Selective Flooding for Better QoS Routing

by

Gangadharan Kannan

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2000

APPROVED:

Professor Mark Claypool, Thesis Advisor

Professor David Finkel, Thesis Reader

Professor Micha Hofri, Head of Department

To Jayamma, Daddy, Hari and Kanchana, who make my WORLD.

Abstract

Quality-of-service (QoS) requirements for the timely delivery of real-time multimedia raise new challenges for the networking world. A key component of QoS is QoS routing which allows the selection of network routes with sufficient resources for requested QoS parameters. Several techniques have been proposed in the literature to compute QoS routes, most of which require dynamic update of link-state information across the Internet. Given the growing size of the Internet, it is becoming increasingly difficult to gather up-to-date state information in a dynamic environment. We propose a new technique to compute QoS routes on the Internet 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 thesis 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.

Acknowledgements

Words fail me in expressing my gratitude to Prof. Mark Claypool. He has been my mentor, colleague and friend all at once. His confidence in me has boosted my morale to no limits. On countless occassions he has gone out of his way to help me with anything and everything. For all these, and more, thank you, Mark.

I would like to thank everyone at PEDS and PERFORM. Their comments, criticisms, suggestions and insights have helped a lot in shaping this thesis. I would like to mention my best friends at WPI, Daniel and Anshul, for being there whenever I needed them. Thanks also to all my other friends here at WPI for their support and motivation. I would like to thank Andreas for all his help with latex. Had it not been for him this document would have been in MS-Word.

Overall the entire process, from the setting of goals to the completion of this report has been a learning one, which I will remember for a long time to come.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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 [7]. Multimedia and real-time applications have stringent bandwidth and timing requirements. With advances in the field of high-speed networks it is now becoming increasingly realistic to expect reasonable performace from such applications. However work still needs to be done in terms of providing guaranteed levels of service to applications. This chapter introduces the idea of Quality-of-Service Routing.

Today's Internet is mainly a connectionless, best-effort network. Data packets from the same application can take different paths to the destination. Further, packets can be lost, duplicated or arrive out of order. Network resources (switches, buffers etc.) are shared uniformly amongst various applications. This infrastructure works fine for conventional text-based applications. However, it does not meet the requirements of integrated service networks (existing ISDN and B-ISDN networks and future ISPN networks [20]). Specifically it does not provide Resource Reservation, which is vital for guaranteed end-to-end performance.

It is therefore predicted that the next generation of the Internet will be connection ori-

ented [10]. This means that for every flow, there would 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.

1.1 QOS-Sensitive Network Services

There has been a general move towards QoS-sensitive network services. In order to support the QoS demands of applications, both the ATM and the IP community have defined service classes that provided per-flow guarantees to applications. At the network layer (*layer 3*) resource reservation can be done using Diff-serv [5] or Multi Protocol Label Switching (MPLS) [1]. The Routing protocol typically used for this is an extension to OSPF called QOSPF [28]. Applications at this level are classified as best-effort, rate-sensitive or delay sensitive and routed appropriately. At the data-link layer (*layer 2*) we have ATM which uses a hierarchical routing protocol (PNNI [12]). Virtual Channels are used to reserve resources from the source to the destination. The QoS metrics used are maxCTD (maximum Cell Transfer Delay), peak-to-peakCDV (Cell Delay Variation) and CLR (Cell Loss Ratio).

Applications can typically specify their QoS requirements as a set of constraints. These could be 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[1] could require that links composing the path have 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

[1]The terms call, connection and request are used interchangeably.

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 (a path that has sufficient residual 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.

1.2 Additional Components of a QoS Network

There are other parts to a QoS-sensitive network besides QoS routing. This section discusses the interactions of QoS routing with these parts.

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. ATM does this through Virtual channels. 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[6]. MPLS and Diff-serv are other emerging techniques to reserve resources in intermediate resources for different flows or classes of service.

Admission Control: Given the limited network resources, it is not possible to provide QoS to every requesting connection. Therefore, we need some sort of admission control, which will accept a call only if the network can support that call at that instance. For every QoS request, we try to reserve the necessary network resources. If we succeed the call is accepted, otherwise it is refused.

QoS negotiation: A QoS routing algorithm may fail to find a path for a prticular 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.

1.3 Issues with QoS Routing

Although conceptually simple, QoS routing suffers from various problems such as diverse QoS specifications, dynamically changing network state and the need to coexist with best-effort traffic [7]. Each one of these is discussed in detail below.

Different distributed applications such as teleconferences, video on demand, Internet phone and Web-based games have different QoS requirements. Applications can specify different QoS constraints such as delay, jitter (variation in delay), bandwidth, loss ratio, etc. It is also possible that they place a constraint on the cost they can afford to get a QoS path. Multiple constraint further complicates the problem. For example, finding a path with two independent path constraints is an NP complete problem.

Future networks are likely to carry both QoS traffic and best-effort traffic. This makes the issue of optimization complicated. This is mainly because their distributions and performance metrics are different. Although QoS traffic will not be affected, due to resource reservation, the throughput of best-effort traffic will suffer if the overall traffic distribution is misjudged. Meeting the QoS requirements of each individual call and reducing the call blocking rate are important metrics for QoS while fairness, overall throughput and average response time are important metrics for best-effort routing. Various techniques have been suggested to integrate QoS with best-effort traffic [21].

In order to compute a QoS route we need accurate information of the network state. 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 (described in Chapter 2) depending on where this information is stored and how it is parsed. Various metrics are 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 a minimum Call-blocking rate. The network cannot accept any more calls because it actually 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 is the time from when the source receives a QoS requst 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 thesis 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. No link state information is updated. 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 thesis, we simulate the Selective Flooding algorithm and evaluate its performance compared to source routing.

The rest of this thesis report is organized as follows. In Chapter 2, we discuss various proposed routing strategies. In Chapter 3, we describe in detail the design of the Selective Flooding routing algorithm. Chapter 4 describes the set-up we use to simulate and evaluate Selective Flooding. Chapter 5 presents results from our experiments. Finally in Chapter 6, we conclude with our findings and future work.

Chapter 2

Related Work

The goals of QoS routing as discussed in the previous chapter are (1) satisfying the QoS requirements for every admitted connection and (2) achieving global efficiency in network utilization.

Routing involves two basic steps: (1) collecting the latest state information at every node on the network, and (2) searching the state information for a feasible path. In order to find an optimal path which satisfies the constraints, the state information about the intermediate links between the source and the destination(s) must be known. The search for feasible paths greatly depends on how the state information is collected and where this information is stored. There are three routing strategies, classified according to where the state information is maintained and how the search of feasible paths is carried out. These are *source* routing, *distributed* routing and *hierarchical* routing.

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 [28, 14]) 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.

In hierarchical routing, nodes are clustered into groups, which are recursively clustered into higher level groups, creating a multi-level hierarchy. Each physical node maintains an aggregated global state, which contains the detailed state information about the nodes in the same group and the aggregated state information about the other groups. Source routing is used to find a feasible path on which some nodes are logical nodes representing groups. A control message is then sent along this path to establish the connection. When the border node of a group represented by a logical node receives the message, it uses source routing to expand the path through the group.

In the next few sections each one of these routing strategies is discussed in detail with specific examples of routing algorithms.

2.1 Source Routing

Source routing achieves its simplicity by transforming a distributed problem into a centralized one. By maintaining a complete global state, the source node calculates the entire path locally. It avoids dealing with the distributed computing problems such as distributed state snapshot, deadlock detection and resolution, and distributed termination problem. Source algorithms are conceptually simple and easy to implement, evaluate,

debug and upgrade. In addition, it is much easier to design centralized heuristics for some NP-complete routing problems than to design distributed ones.

Source routing has several problems [7]. First, the global network state maintained at every node has to be updated frequently enough to cope with the dynamics of network parameters such as bandwidth and delay. Second, the link state algorithm can only provide approximate global state due to non-negligible propagation delay of state messages. This imprecision can cause QoS routing to fail. [26] discuss in detail the effects of stale link-state information and random fluctuations in traffic load on the routing and signalling overheads of source routing. Third, the computation overhead at the source is excessively high, more so for multicast or multi-constraint routing. In summary, source routing suffers from scalability problems.

Various algorithms have been proposed for both unicast and multicast source routing. All of them require a global state to be maintained at every node. Most algorithms for unicast source routing transform the routing problem to a shortest path problem and then solve it by Djikstra's or Bellman-Ford algorithm.The most popular among source routing alogorithms is the proposed QoS extensions to the OSPF known as QOSPF [28, 14]. QOSPF includes an "explicit routing" mechanism for source-directed routing. However as with other Source routing protocols, QOSPF imposes a significant bandwidth and processing load on the network, since each switch must maintain its own view of the available link resources, distribute link-state information to other switches, and finally compute and establish routes for every connection.

The source routing algorithm is typically executed at connection arrival on a per-connection basis, increasing the runtime computation overhead. [25] study path precomputation and caching and suggest the following techniques to improve the efficiency of

source routing:

- Coarse-grain link costs: Path selection is based on link-cost metrics, which are a function of link-state information. Limiting link costs to a small number of values reduces the computational complexity of the path selection algorithm. Coarse grain link costs do not significantly degrade performance, and increase the likelihood of having more than one minimum cost route to a destination.

- Precomputation of minimum-cost graph: Each switch or router precomputes a compact data structure that stores all minimum cost routes to each destination. Instead of storing the precomputed paths in a cache, route extraction is postponed until connection arrival.

- Route extraction with feasibility check: As part of route extraction, the source checks the feasibility of each link, based on the most recent link state information and the bandwidth requirement of the new connection. The first route that satisfies the common case is extracted.

- Reranking of multiple routes: As part of route extraction, the source can rerank the links to improve the path selection process for the next connection. This provides a simple framework for a number of alternate routing policies.

These mechanisms enable a wider range of policies for when to compute new routes, how many candidate routes to try for a new connection, and how often to update link-state information, thus causing a significant reduction of processing overhead and set-up delay, in comparision to traditional on-demand source routing algorithms.

As with Source Routing, Selective Flooding computes the entire path to the destination at the source itself. However Selective Flooding does not need any link-state information propogation. Since we use the most-recent link-state information, we hope to do

better than Source-Routing in terms of call-blocking rate, as in our case calls will not be refused due to stale link-state information. It should be noted that Selective Flooding has a per-call network routing overhead while Source Routing typically has network overhead once every link-state update period. While [25] do path precomputation based on link-state information, we do path precomputation based totally on the topology information. Further they precompute just one feasible path from the source to the destination while we precompute multiple possible paths to the destination. In Chapter 4 we compare Selective Flooding to a source routing algorithm, measuring the call-blocking rate and the network cost.

2.2 Distributed Routing

In the case of Distributed Routing, the path computation is distributed among the intermediate nodes between the source and the destination. Hence, the routing response time can be made shorter and the algorithm more scalable. Typically a Distance Vector protocol is used for routing. Here, through continuous state updates, each node knows for every destination the next hop on the best path. Also, searching multiple paths in parallel for a feasible path is made possible, which increases the chance of success. Most distributed routing algorithms require each node to maintain a global network state, based on which the routing decision is made on a hop-by-hop basis. Flooding algorithms, on the other hand, do not require any global state to be maintained.

Distributed algorithms that depend on the global state have similar problems to that of source routing algorithms. In addition they also need to address distributed computation problems such as distributed state snapshot, deadlock detection and resolution, and the distributed termination problem. Inconsistent global states at different nodes can cause loops to occur. It is also difficult to come up with efficient heuristics for NP-complete

routing problems.

Different distributed techniques have been presented in the literature. Kweon and Shin [17] suggest a technique of bounded flooding where routing messages are flooded into the network for a certain hop count in all directions. No global state is required to be maintained at any node. However, it is possible that with certain sets of constraints, the hop count might not include the optimal path in the search space. A flooding-based routing techinique was proposed by Hou [15] for routing in ATM networks. Another algorithm which uses flooding was proposed by Chen and Nahrstedt [8]. They suggest a distributed routing framework based on selective probing. Unlike in [17] probes are forwarded only to a subset of outgoing links selected based on the topological distances to the destination. Further the probes proceed only when the nodes and the links on the way have sufficient resources. This reduces the overhead as compared to [17]

Some algorithms couple routing with reservation. Cidon et al [9] propose reserving resources in parallel along multiple routes. Here every node still maintain a picture of the whole topology and the status on every link, and for every connection request, multiple paths are tried in parallel. Along each path resources are reserved for the connection request. After the best route is selected resources previously reserved along other routes are released. Reserving resources on multiple paths makes routing more resilient to changes in the network state. But it also increases the call-blocking ratio as connections might be denied requested paths even though the network can actually support their requirements. [9] improves its performance by continuing its reservation on a path only if the path can support the QoS requested.

Selective Flooding is essentially a distributed routing algorithm. Unlike other distributed algorithms though, we precompute at the source itself a list of possible paths to

the destination and then flood only along these routes. Current link-state information of the path to the destination is collected in a distributed fashion. Selective Flooding has much less overhead than other flooding algorithms as we flood only over links which definitely lie on a path to the destination.

2.3 Hierarchical Routing

Hierarchical Routing helps in overcoming the scalability problem which source routing faces. This is because each node only maintains a partial global state where groups of nodes are aggregated into logical nodes. The size of such an aggregated state is logarithmic in the size of the complete global state. The source routing algorithms are then directly used at each level to find feasible paths based on the aggregated states maintained at nodes. Thus hierarchical routing retains most of the advantages of source routing. It has also some advantages of distributed routing because many nodes share the routing computation.

However aggregation of nodes introduces further imprecision in the link state information. Also when multiple QoS constraints are involved, different paths may exist within each logical node to optimize different constraints. There may not exist a path with the best properties for all constraints. How to aggregate such information is still an open problem.

There has been work done on how to scale Source Routing algorithms by considering a hierarchical architecture [2, 3, 4]. At the data link layer, the PNNI(Private Network-Network Interface) [12] standard for routing in ATM networks is hierarchical. PNNI uses a crankback mechanism to search multiple paths sequentially. When the selected path does not meet the requirement, the routing process is cranked back and resumes with an

alternative path. While this works fine, it has a longer routing time.

Selective Flooding easily lends itself to Hierarchical Routing thus making it scalable. In this case the source will compute and store multiple hierarchical routes to the destination. Each such route would consist of physical and logical (aggregated) nodes. Thus while PNNI checks multiple routes from the source to the destination in a serial manner, Hierarchical Selective Flooding does the same in parallel. Hierarchical Selective Flooding is explained in the next chapter.

In this chapter, we have analysed different routing strategies and compared them in brief to Selective Flooding. In the next chapter we describe in detail the Selective Flooding QoS routing algorithm.

Chapter 3

Selective Flooding

Source routing stores two important pieces of information at every source. One, the network topology (nodes and presence of links between them). Two, the latest link-state information about the whole network. The topology is relatively static and does not cost much in terms of network overhead. Transmitting link-state information however is expensive, needing 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.

3.1 Approach

The basic idea of Selective Flooding is to check 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. No link-state information is necessary. 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) (either ICMP or TCP packets) 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 can be computed and resources along this path can be 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 us a list of feasible paths from the source to the destination for the particular QoS request. We can then compute the best path from among these feasible paths. As in [8] 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.

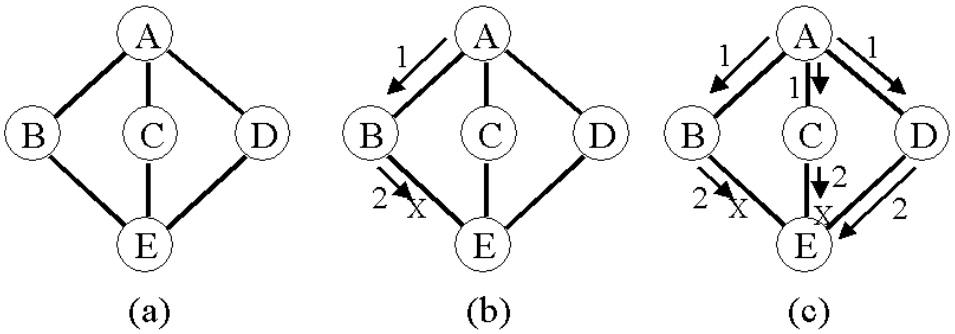


Figure 3.1: QoS Routing Illustration (a)-Topology (b)-Source Routing (c)-Selective Flooding. The letters are nodes. The numbers indicate sequential steps. The X's indicate routing failures.

To better illustrate Selective Flooding consider the sample topology in Figure 3.1(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 3.1(c)). These packets continue along the path only if the path has the requested resources. Thus unlike in Source Routing (Figure 3.1(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 and corresponds to the length of the longest path probed.

The best path means the path among all feasible paths which maximizes call admittance into the network and improves the network performance. 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. We could choose either the shortest feasible path in terms of number of hops or choose the path which maximises 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. 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.

3.2 Analysis

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 all this 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 has a high Call-blocking rate.

There is a network cost involved with every call. The network overhead in Selective Flooding is per-call whearas 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 we probe 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 we also need to consider the computation cost at each router when a routing packet arrive. 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 consumes a lot of computa-

tion power 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 a lot of precious 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 static and done only once at boot time.

In Chapter5 we present results from our experimental evaluation of Selective Flooding.

3.3 Extensions

Several heuristics can be applied to further speed up the path selection process and to reduce network resource utilization.

While theoretically we could store all possible paths from every source to every destination, it is obvious that this is not required. As mentioned before the network cost incurred doing Selective Flooding is proportional to the number of paths we probe. Based on the network topology we could compute (through experiments) MAX-PATHS as the maximum number of paths we need to store from a source to every destination. MAX-PATHS thus represents the threshold beyond which storing additional paths will not give us added benefit in terms of further reducing call-blocking rate. As we will show later, this MAX-PATHS is typically a small number thus significantly reducing the network

overhead. An upper limit can be set on the hop count of all the paths stored initially, reducing the number of control messages. Based on past link-utilization patterns, we can restrict flooding to only those routes that have a high probability of satisfying the QoS.

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. In all, Hierarchical Selective Flooding (HSF) can result in substantial gains in terms of reduced network utilization and faster route computation. In this context ATMs crank-back PNNI routing technique [12] can be visualized as a serialized version of Hierarchical Selective Flooding.

In the next chapter we describe how we have evaluated the Selective Flooding algorithm.

Chapter 4

Implementation

As mentioned in Chapter 1, the performance of any QoS routing algorithm can be measured by two main performance parameters: call-blocking rate and network overhead. Our primary goal in evaluating Selectve Flooding was to compare its performance to that of source routing, the most common form of QoS routing [26]. We evaluated Selective Flooding through simulation as using an analytical model would be too coarse given all the possible parameters and an actual implementation is difficult for reasons of interoperability.

4.1 Simulating QoS Routing

We have used *routesim* as the simulator to conduct our tests [24]. *routesim* is an open-source, event driven simulator used to study QoS routing in large networks. It is written in C and was developed by Anees Shaikh at the University of Michigan.

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

```
topology-type file
network-model SINGLE
arrival-spec ARR_UNIFORM
arrival-dist POISSON
arrival-scale .100
arrival-shape 1.00
traffic-multiplier 1.0
holding-time-spec EXP
holding-time-shape 1.0
holding-time-scale 10.00
mean-bandwidth 0.060
bandwidth-spread 1.99000
update-trigger 100000.0000
refresh-interval 10.000000
refresh-skew 0.05
min-update-interval 0.000000
min-requests 500000
min-warmup 100000
min-sim-time 300.0
warmup-proportion 0.25
blocking-policy BLOCKING
routing-policy ON_DEMAND
routing-algorithm WIDE_SHORT
distance-function HOPCOUNT
hopcount-threshold 1
wide-short-maxhops 10
alternate-routing NO_ALT
multi-route-policy UNIQ
uniq-route-policy RANDOM
max-sig-attempts 1
prune-policy NO_PRUNE
link-state-accuracy STALE
random-seed 1013553325
confidence-level 99
sim-tolerance .05
warmup-tolerance .075
```

Figure 4.1: Sample Configuration File for *routesim*

simulator. Typically all these parameters are specified using a "configuration file". A sample configuration file is shown in Figure 4.1.

A major reason in choosing *routesim* for our simulations was its design and implementation. It had a fairly flexible architecture. The code for the most part was well documented and clean, thus making it easy for us to introduce the changes necessary to implement Selective Flooding. Another big advantage is that this simulator has been used to compare other Source-Directed Quality of Service Routing techniques [26], [25], [23] . Thus, we could borrow results from these experiments for ours, and also validate our use of the simulator by comparing our results from the modified simulator to his original results. *routesim* is available on Solaris, Irix64 and Linux platforms. We ran all our tests on the Sun Solaris platform.

4.1.1 *routesim* Features

In the next few paragraphs we discuss some important parameters in *routesim*.

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. For a user defined topology the "rstopology.dat" file is used to describe an arbitrary network topology. Each line in this file defines a unidirectional network link. For each such unidirectional link we can specify the link capacity, propogation delay and the maximum number of calls it can support. Additionally we can also provide an arbitrary administrative weight for each link. The first line in the file contains the number of nodes in the network. The format of a topology file is as follows:

```
num nodes
from to capacity propdelay adminweight maxcalls
.
.
.
```

In our experiments, we assumed that all links had equal capacities and propogation delays. We used 3 different topologies for our simulations.

- Sparse 19-node MCI backbone (average degree - 3.4)
- 50-node Random Topology (average degree - 7.0)
- 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. The "rstraffic.dat" file in *routesim* is used to define a routesim traffic matrix. This traffic matrix specifies the arrival rate between each source-destination pair. It also allows different values for different times of the day. Network traffic distribution is typically modelled using a Poisson distribution [16]. For all our experiments, we assumed an uniform traffic pattern with Poisson flow interarrival distribution.

Routing Algorithm and Policy: Various QoS routing policies have been implemented on the simulator. These include 2 link-state routing protocols, one using Dijkstra'a algorithm and the other Bellman-Ford. Both the algorithms provide almost same performance. The routing policy can also be set to "precompute" to simulate QoS routing using pre-computation [25]. In our experients 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: We can specify 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. Further we 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.

Statistics: On completion of the simulation *routesim* provides numerous useful statistics about the simulation. These include the number of routing and signalling 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.

4.1.2 Constraints to *routesim*

routesim has a restrictive event model. It has only 3 events STARTCALL, ENDCALL and PRECOMP. The STARTCALL event is responsible for routing, extracting a feasible path and signalling. 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 signalling. The bandwidth cost of link-state updates in source routing was not considered, nor was the route computation cost at the source. Also, it was not possible for us to measure accurately the costs incurred by our technique in terms of routing time. The topology model could not be viewed in a hierarchical manner. Hence we did not implement the heuristic of Hierarchical Selective Flooding (Section 3.3).

4.2 Implementing Selective Flooding on *routesim*

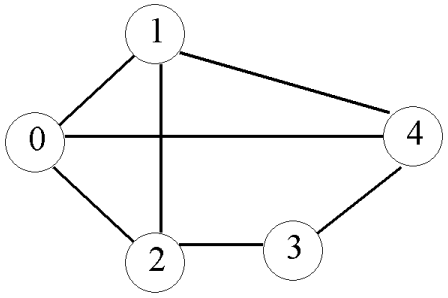
Implementing Selective Flooding had 3 major parts. First, to implement a function which would, given any topology, return multiple paths from any source to destination. Ths second was to implement the actual Selective Flooding algorithm. Finally we had to integrate these parts within *routesim* and add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

4.2.1 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. The technique which we have used was developed by David Eppstein [11]. This method 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 started off by borrowing from a sample implementation of the technique provided by Graehl [13]. We then made changes to it 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.

We tested the program 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 Fig. 4.2 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



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 4.2: Sample test for multiple paths algorithm. Table shows the routes stored at Source 0 for various destinations as calculated by our program

additional field in the call structure to store the multiple possible routes from the source to the destination of the call. Each call would then at startup make a function call to compute "k" 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 was 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 detemined by the size and connectivity of the topology being considered.

4.2.2 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

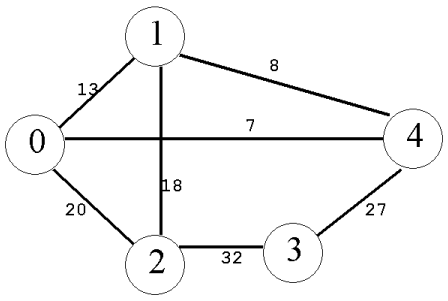
```
for each path from source to destination {
   feasible = true;
   for each link on the path {
      if (available bandwidth on the link < requested bandwidth) {
          feasible = false;
          break;
      }
   }
   if (feasible) {
      set this as the path on which to signal
      update current_time
      return;
   }
}
No feasible path exists.
Declare Routing failure.
Update current_time.
return;
```

Figure 4.3: Pseudocode for the Selective Flooding algorithm.

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 [25], we used the hop-count as a metric for the best path. It should be noted that this metric is a paramter 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 signalling. This part is built into routesim and is common for all the routing algorithms. Pseudocode for the

Selective Flooding algorithm is presented in Figure 4.3.

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 signalling failures when using Selective Flooding. In contrast, there can be signalling failures in source-routing algorithms, as it is possible that the link-state using which the path was selected may be stale.



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 4.4: Sample test of Selective Flooding algorithm Implementation

We tested our implementation of the algorithm to check that it worked correctly. An example of our tests is shown in Fig. 4.4. 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 10Mb. 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.

4.2.3 Changes to *routesim*

Selective Flooding is basically a routing algorithm which is used to find the path along which resources should be reserved for a connection. *routesim* has within it implemented a couple of other routing algorithms, namely, shortest-path tree (Dijkstra) and wide-short algorithm (Bellman-Ford). Whenever the routing algorithm is needed, a switch statement is used to perform the appropriate action depending on which routing algorithm is being simulated. So we had to insert the code for Selective Flooding in the appropriate places. Also an added feature was that the signalling (or reserving of resources) was done independently of the routing algorithm. Since the routing algorithm is used to detemine the route which should be used for signalling, the rest of the simulation is the same irrespective of which algorithm is being used for routing. So, our main focus was to implement the Selective Flooding algorithm.

We modified the existing statistics counters to behave appropriately for Selective Flooding. We also added code to measure the number of packets sent into the network by both Source Routing and Selective Flooding.

After these changes were implemented, we ran tests with the other routing algorithms using both the new version and the original version of *routesim*. This was to make sure that we had not changed the original behaviour of the system. While system resources external to the simulation, consumed for the simulation were different in both cases (as was expected) the simulation results were the same.

For example, we tried to recreate one of the results from a previously published paper [26] which used *routesim*. Figure 4.5a shows for a 100 node random topology, the effect of the link-state update frequency on the call-blocking rate of source routing. Figure 4.5b shows a picture of the results previously published [26] from the same topology. We

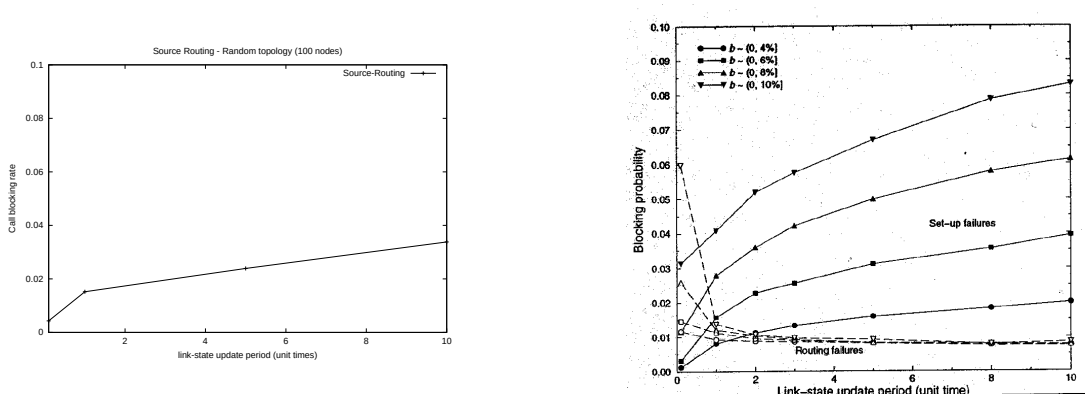


Figure 4.5: Call-blocking vs link-state update period - Random topology (100 nodes) (a)-Test run (b)-Previously published [26]

were able to reproduce the exact same results after we had implemented the changes to *routesim* thus confirming that we had not broken the simulator.

In the next chapter we present results from our simulations and our evaluation of the Selective Flooding algorithm.

Chapter 5

Evaluation

This chapter discusses some of the results obtained and inferences drawn from our simulations.

Our main focus was to compare the performance of Selective Flooding with that of Source QoS Routing. As discussed in Chapter 1, various metrics can be used to compare the performance of different routing algorithms. The important ones we could use are: 1) Call-blocking rate - The percentage of calls blocked; 2) Network Overhead - This would involve 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 rest analytical results are presented.

5.1 Settings

We used 3 main sets of topologies for our simulations. As, is common with backbone networks [29] we consider topologies with relatively high connectivity that support a dense traffic matrix and are resilient to link failures. Table 5.1 lists the topologies we used.

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 5.1: Topologies used for Simulations.

The random graphs were generated using Waxman's model [27]. 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 [26, 18, 19].

As is typical in actual networks and other simulations [22], 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.

5.2 Call-Blocking

Figures 5.1, 5.2 and 5.3 compare 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 this value remains constant and less than the best possible by source-routing. This is because Selective Flooding does not need any link-state updates. It checks 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 it will find the path. Source routing on the other hand depends on stale link-state information in making its routing decisions and hence could be wrong at times.

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 5.4 summarizes the results across all the topologies we tested. In all cases Selective Flooding has the best call blocking rate possible for the topology-traffic combination.

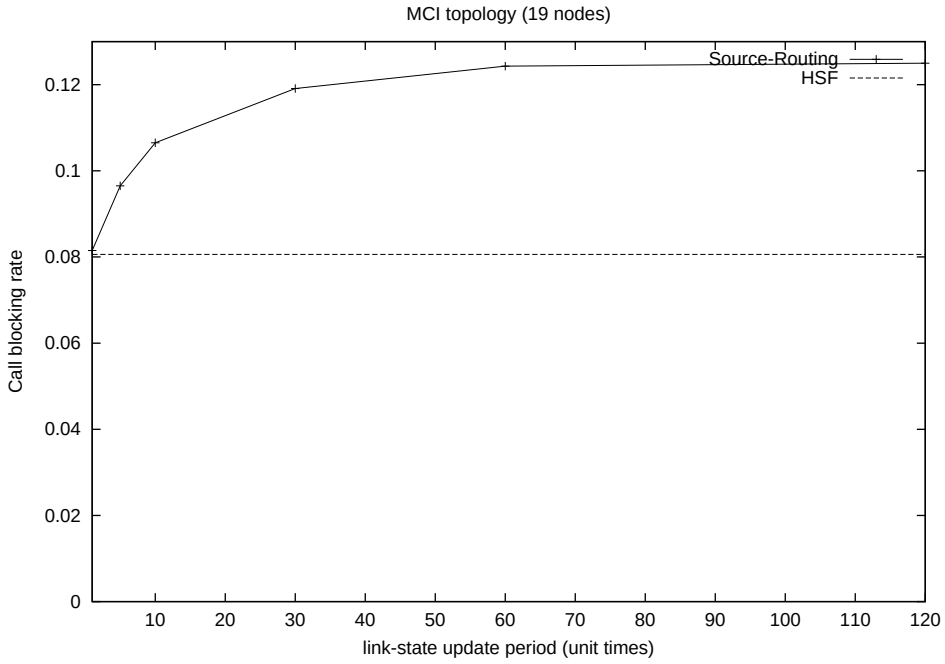


Figure 5.1: Call-blocking vs link-state update period - MCI topology

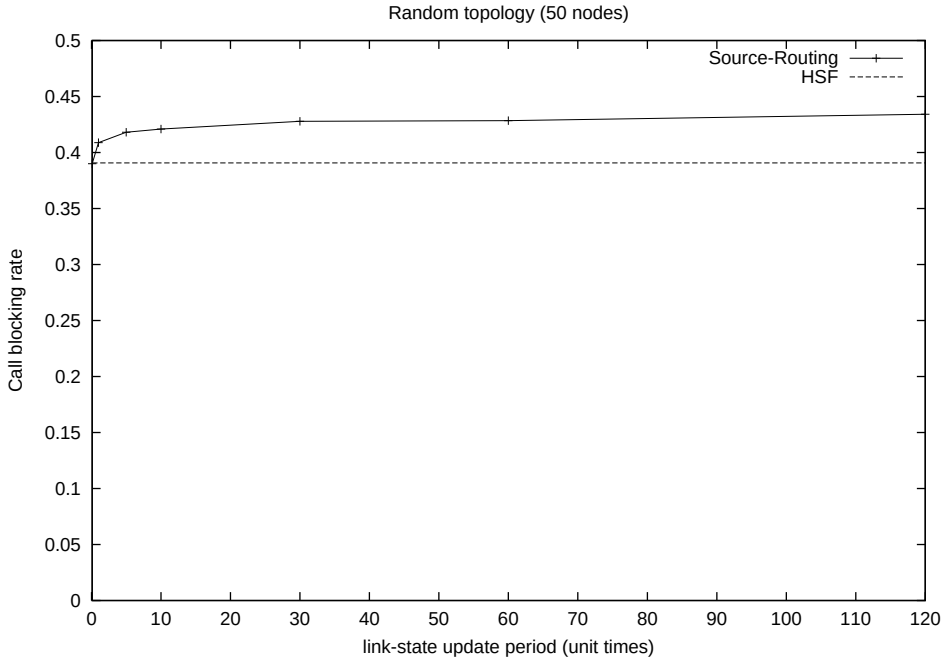


Figure 5.2: Call-blocking vs link-state update period - Random topology (50 nodes)

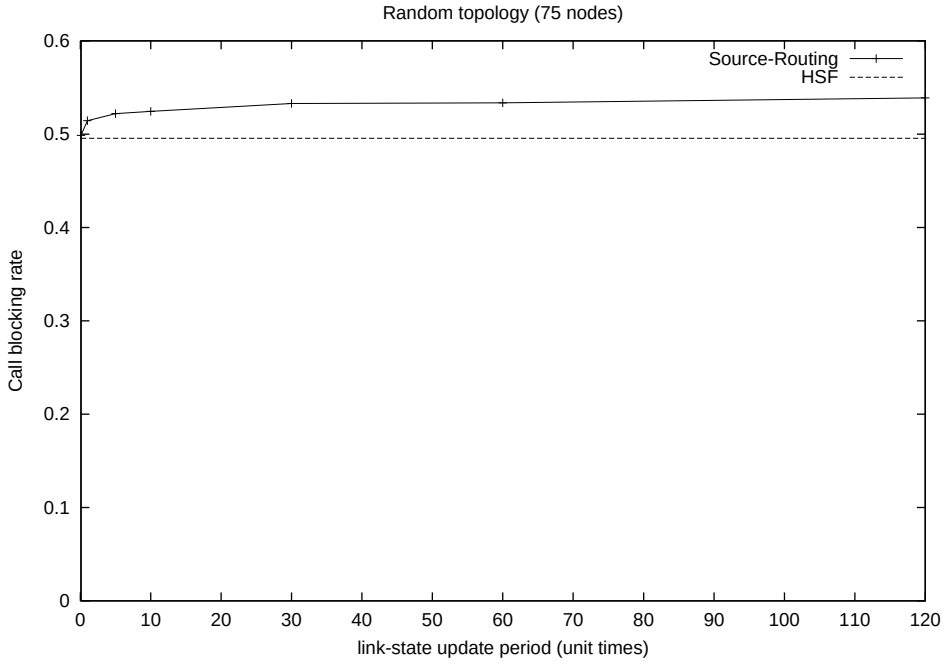


Figure 5.3: Call-blocking vs link-state update period - Random topology (75 nodes)

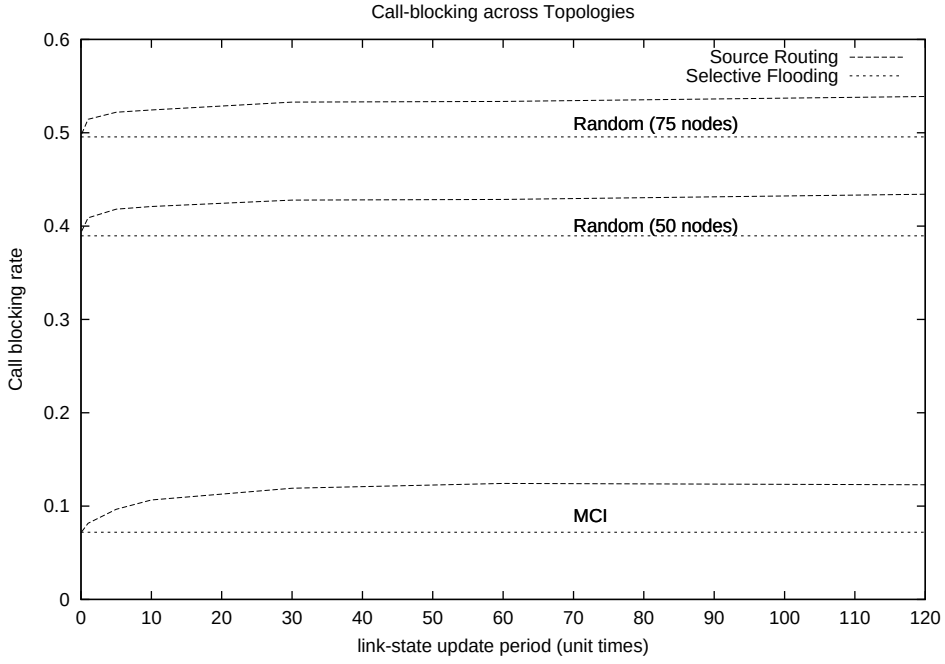


Figure 5.4: Call-blocking vs link-state update period - All Topologies

5.3 Network Overhead

routesim within itself did not have any support to measure the network overhead of any routing algorithm. Its main metric was the call-blocking rate. We added an additional metric to measure the network cost incurred by both source routing and Selective Flooding. This 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. In the case of Source Routing it measures the number of link-state update packets, whereas in Selective Flooding it measures the number of probe packets (control messages) sent into the network.

Figures 5.5, 5.6 and 5.7 show 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.

In the case of source routing, the cost being measured is the number of packets required by the network for link-state update. 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 straight line for Selective Flooding in Figures 5.5, 5.6 and 5.7. 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. As discussed in Section 3.3, we need not necessarily check all the paths from the source to the destination. The fewer the number of paths we check the lower the network cost incurred. If “k” is the number of paths being checked per call and “l” the average length of the path then the number of control packets required to route a call are k*l.

Figure 5.8 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.

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 [29]. 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 needs to update its picture of the network. It then needs to recompute the shortest path to all the nodes and update its routing tables. Finally, it needs to route 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. Higher the link-state update frequency more the number of link-state update packets and

more the overhead incurred at the router. Also this presents a scalability problem for bigger 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 saves a lot of computing time on the routers as compared to source routing, making it more scalable.

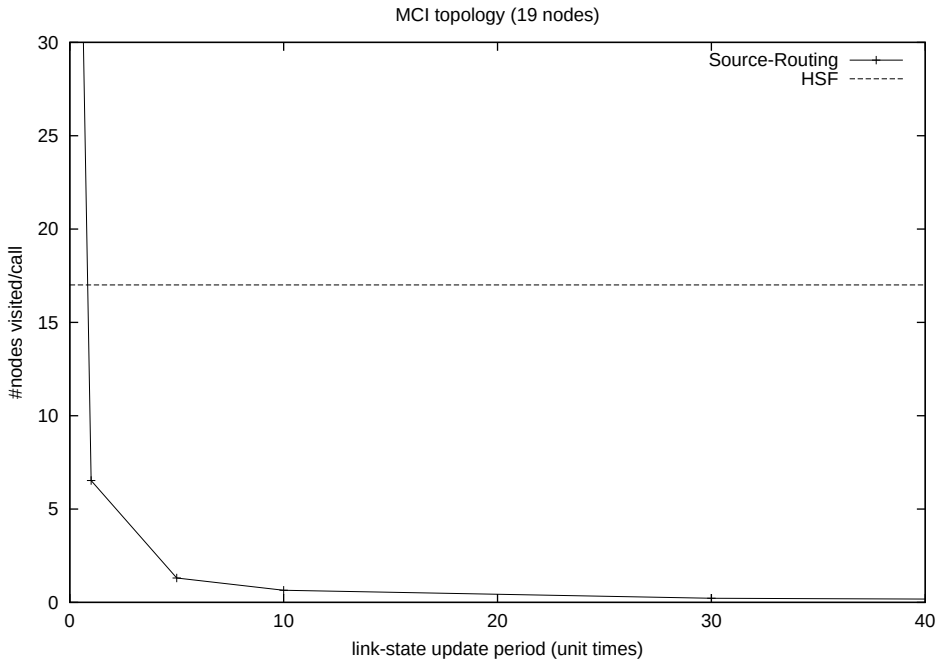


Figure 5.5: Network Overhead vs link-state update period - MCI topology

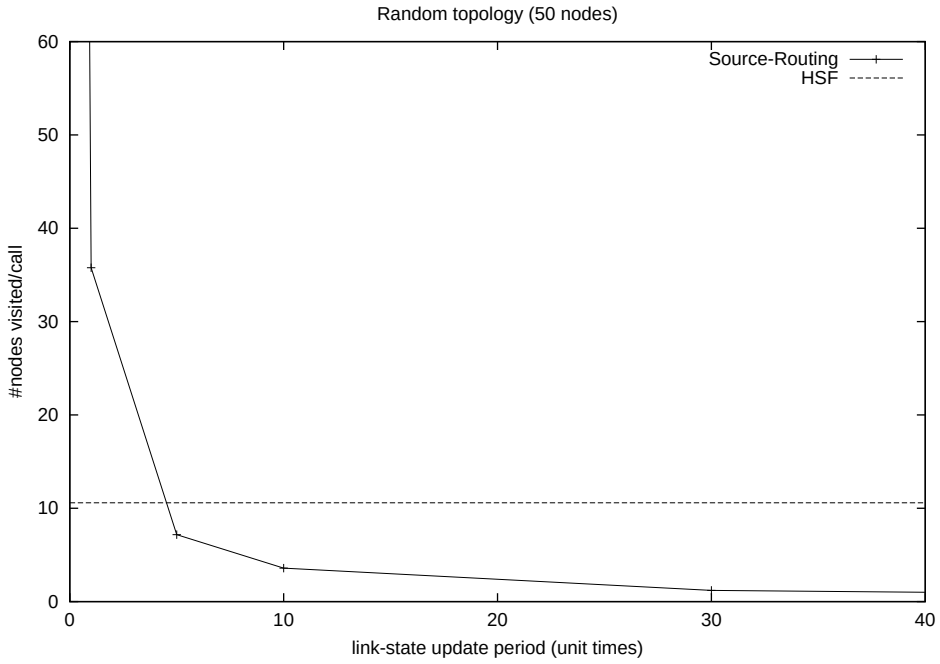


Figure 5.6: Network Overhead vs link-state update period - Random (50 nodes)

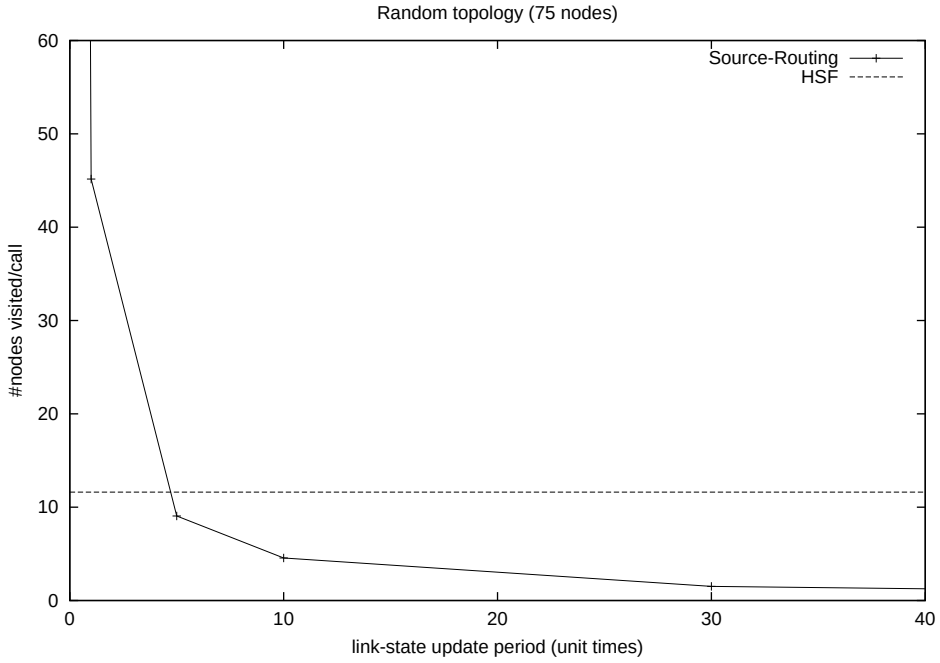


Figure 5.7: Network Overhead vs link-state update period - Random (75 nodes)

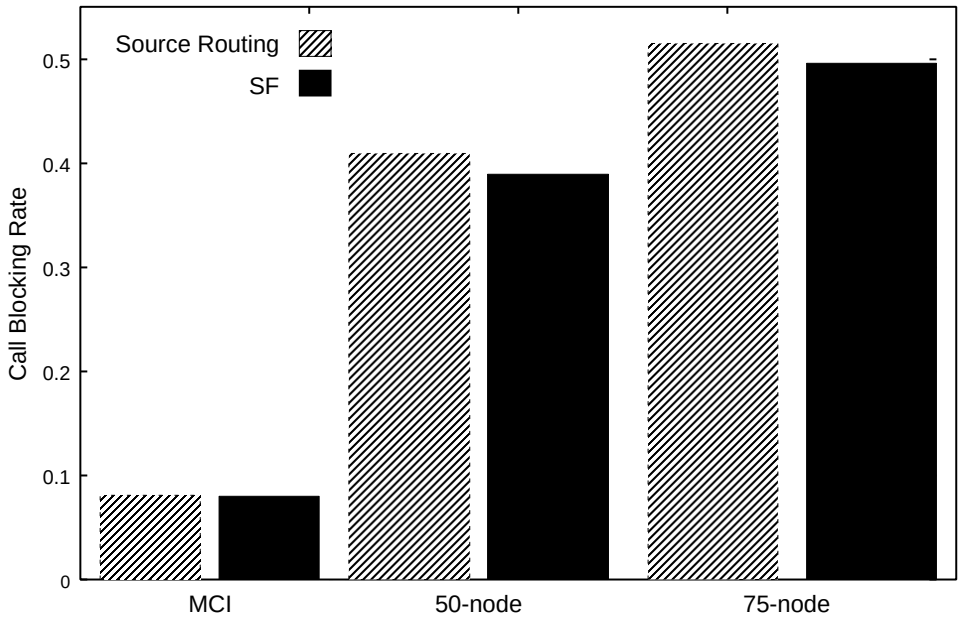


Figure 5.8: Call blocking rate for different topologies at the same network cost

5.4 Value of k

As mentioned before, for Selective Flooding, it is not necessary to store and check all paths from a soure to a destination. Based on network topology and traffic distribution we could store a limited number of paths for each source-destination pair. The number of paths stored ("k") determines the call-blocking rate. Ideally we would like to have "k" as infinity. i.e check all possible paths from any source to any destination, but then this has the highest network bandwidth cost. However, having an inappropriately small value for "k" increases the call-blocking rate substantially, since we now try to reserve all calls from a source to destination along a limited number of routes. Once links on these routes get full, Selective Flooding reports a routing failure. Source-Routing in this case could possibly find a route from those not stored by selective flooding and thus perform better.

Figure 5.9 shows the effect of the value of "k" on the call-blocking rate for different topologies. As expected, the call-blocking rate goes down as we increase the number of

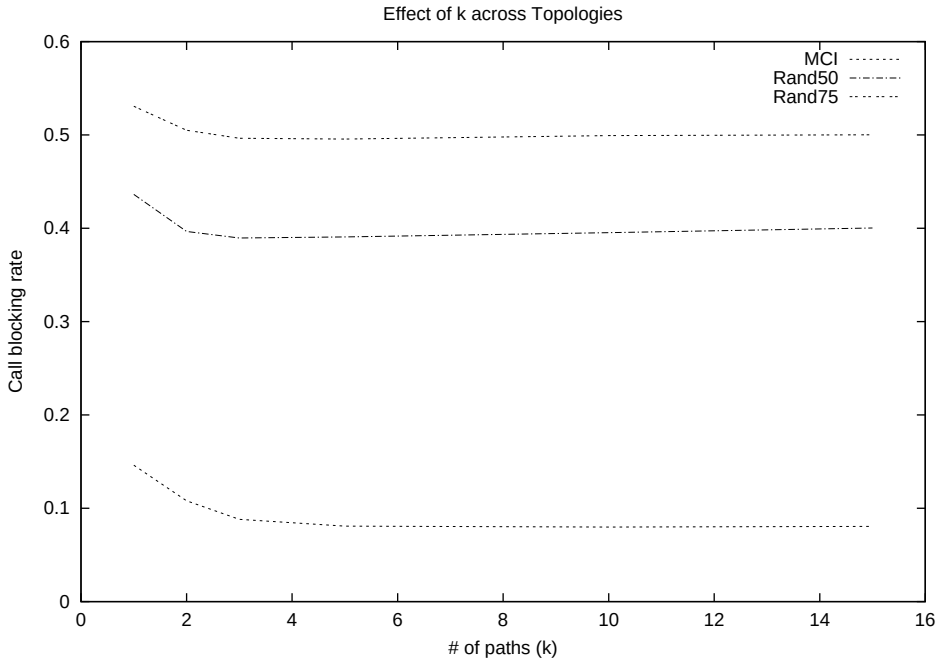


Figure 5.9: Effect of Number of paths probed (k) on call blocking rate across topologies

paths being probed in parallel, until we obtain a call-blocking rate equal to the best possible for the given topology and traffic distribution. Ideally we could set MAX-PATHS to the value at the end of the knee on each curve. For all three topologies studied, this value falls between 3 and 5. This means that we need to check only the first 3 best paths in parallel to get a call-blocking rate better than that of source routing. The total network overhead is thus drastically reduced as compared to probing all possible paths. Figure 5.10 shows for the MCI topology the effect on the call blocking rate and the network overhead as we vary the value of k. Figure 5.11 compares for the 50 node random topology the call blocking rate of source routing and Selective Flooding at the same network cost as we vary the value of "k". As can be seen even at a value of k=3 we get a better call blocking rate for Selective Flooding than for source routing at the same network cost.

The value of MAX-PATHS can be determined apriori for any topology and traffic distributions by running simulations. Further it is possible that we could have different

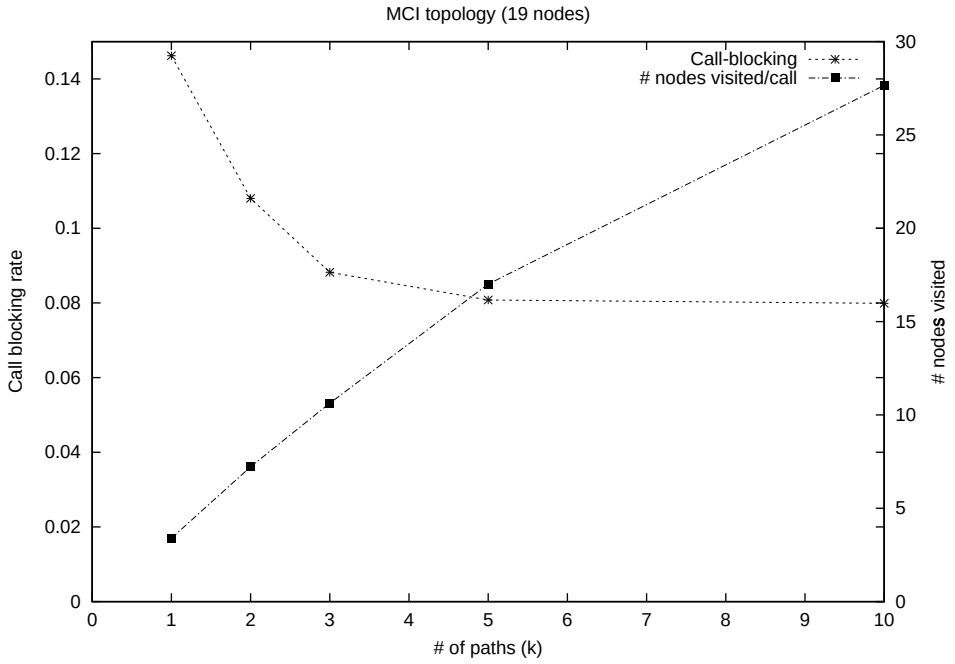


Figure 5.10: Effect of Number of paths probed (k) on Call-blocking and Network cost-MCI topology

values of k at different sources for better performance.

5.5 Other Costs

This section presents an analyses 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

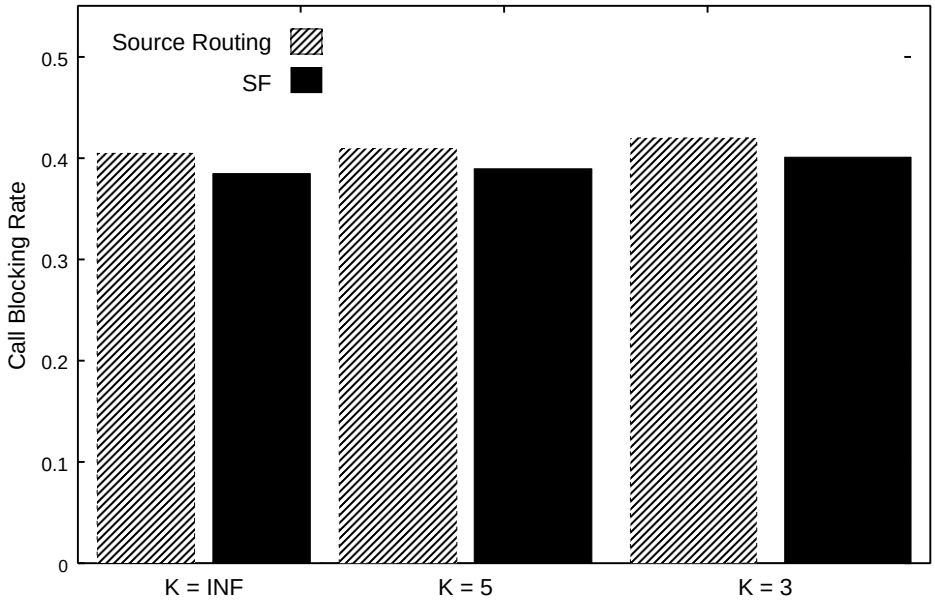


Figure 5.11: Call blocking at the same Network cost for different values of k - Random Topology (50 node)

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. Further to reduce the call setup time, we want this information to be easily retrievable at run-time.

5.6 Effect of Bandwidth

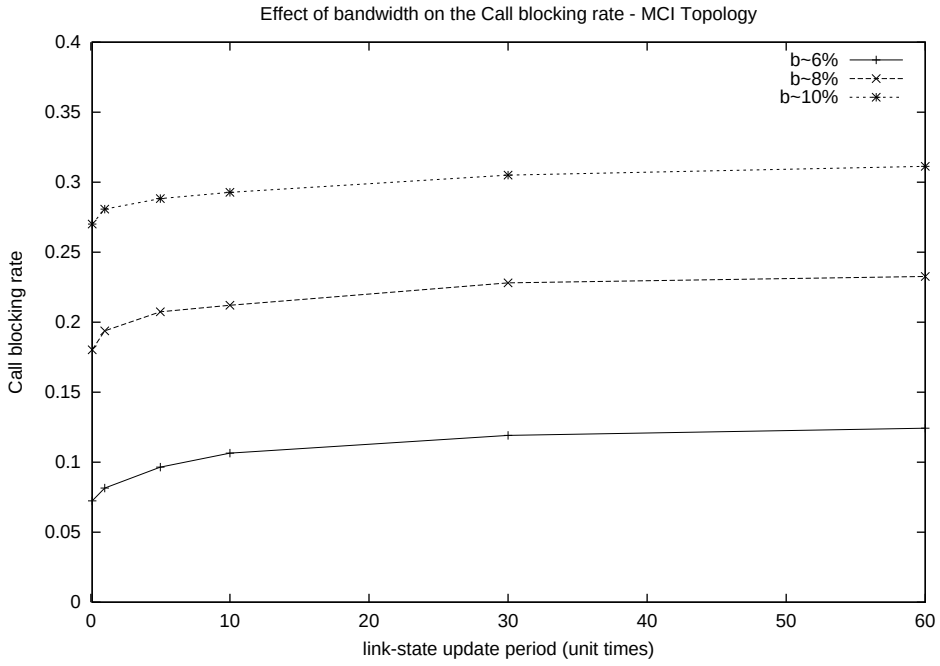


Figure 5.12: Effect of bandwidth on the Call blocking rate (Source Routing)- MCI

We studied the effect of varying the call bandwidth on the performance of Selective Flooding. 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 5.12 shows the effect of bandwidth

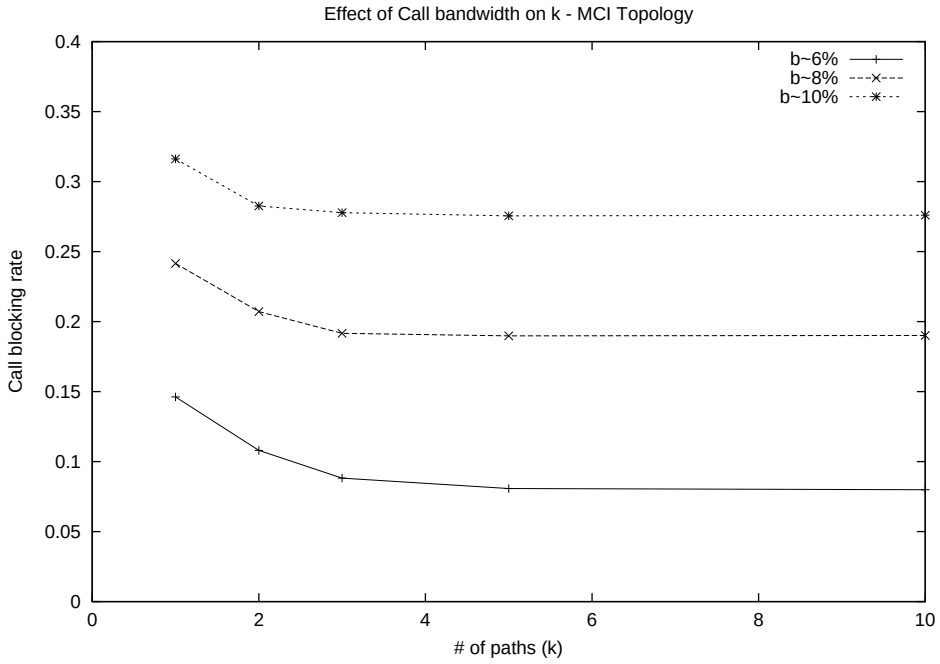


Figure 5.13: Effect of bandwidth on the value of k (Selective Flooding) - MCI

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. Further, for all 3 cases the knee of the curve in Fig 5.13 remains around the same area. This suggests that changes in bandwidth requirements of applications does not affect the performance of Selective Flooding.

In this chapter we have reported results from our experimental evaluation of Selective Flooding and compared its performance to source routing. We conclude in the next chapter by summarizing our findings and mentioning future work in the field.

Chapter 6

Conclusions

In Chapter 5 we presented results from our tests on Selective Flooding and source routing. In this chapter we summarize these results and identify future work that needs to be done in this subject.

6.1 Summary

Traditional source QoS routing has a high call blocking rate due to stale link-state information. Continuously updating the link-state information on the other hand 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. The source code for the same will soon be

made publicly available.

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 computing 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.

For all the tested topologies the actual number of paths needed to be stored in Selective Flooding for every source-destination pair is often much less than the maximum. This further reduces the network overhead while providing the same call blocking rate. The performance of Selective Flooding does not seem to be affected by the average bandwidth requested by applications. Based on tests on 3 topologies Selective Flooding seems to scale well with the network size.

Overall, the contributions of this thesis 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

6.2 Future Work

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

Floooding needs to be compared with other proposed QoS routing techniques.

routesim, the simulator we used to test source routing is limited in its ability to measure run-time computation costs at routers. This is a short-coming with most network simulators. We require a way by which we can model these run-time computation costs at routers for various routing algorithms.

Selective Flooding does not require all the paths from the source to the destination for a reduced call blocking rate. Extensive tests are required to be able to model and predict "k" as a funtion based on topology size, connectivity and traffic pattern.

Hierarchical Selective Flooding (HSF) promises to reduce the network overhead even further as compared to Selective Flooding (Section 3.3). Future work would include implementing HSF and comparing it to other hierarchical algorithms.

Another possible extension to Selective Flooding would be to combine it wth source routing to account for network volatility. Here we could have occasional link-state updates, to detect network topology changes such as link or router failures. Based on this new topology picture we 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 Floooding. For example, we could now 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.

QoS routing is yet at an incipient stage. Lots of further work needs to be done in this area on various fronts before we can have the infrastructure in place. This includes areas like [7] routing with imprecise state information, efficient heuristics to implement

distributed and hierarchical routing algorithms, co-existing with best-effort traffic, QoS negotiation, integrating QoS routing with other network components and testing QoS routing algorithms for generality over different QoS requirements, simplicity and scalability.

Bibliography

[1] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering over MPLS. RFC2702 Network Working Group, September 1999.

[2] B. Awerbuch, Y. Du, B. Khan, and Y. Shavitt. Routing through Teranode Networks with Topology Aggregation. In *ISCC '98*, Athens, Greece, June 1998.

[3] J. Behrens and J. J. Gracia-Luna-Aceves. Hierarchical Routing Using Link Vectors. In *IEEE INFOCOM '98*, March 1998.

[4] J. Bennett and H. Zhang. Hierarchical Packet Fair Queueing Algorithms. In *ACM SIGCOMM '96*, August 1996.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. RFC2475 Network Working Group, December 1998.

[6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S.Jamin. Resource Reservation Protocol (RSVP): Functional Specifications. RFC2205 Network Working Group, September 1997.

[7] Shigang Chen and Klara 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.

[8] Shigang Chen and Klara Nahrstedt. Distributed Quality-of-Service Routing in High-Speed Networks Based on Selective Probing. In *Conference on Local Area Networks (LCN '98)*, October 1998.

[9] Israel Cidon, Raphael Rom, and Yuval Shavitt. Multi-Path Routing Combined with Resource Reservation. In *INFOCOM '97*, pages 92–100, Kobe, Japan, April 1997.

[10] E. Crawley, R. Nair, B.Rajagopalan, and H.Sandick. A Framework for QoS-based Routing in the Internet. RFC2386 Network Working Group, August 1998.

[11] D. Eppstein. Finding the k shortest paths. In *35th IEEE Symp. Foundations of Computer Science*, pages 154–165, February 1994.

[12] Mike Goguen. Private Network-Network Interface Specification Version 1.0. PNNI Specification Working Group ATM forum, March 1996. Document available at ftp://ftp.atmforum.com/pub/approved-specs/afpnni-0055.000.

[13] P. Graehl. www.ics.uci.edu/∼eppstein/pubs/graehl.zip.

[14] 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.

[15] C. Hou. Routing Virtual Circuits with Timing Requirements in Virtual Path Based ATM Networks. In *IEEE INFOCOM'96*, 1996.

[16] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, New York, NY, April 1991.

[17] Seok-Kyu Kweon and Kang G. Shin. Distributed of QoS Routing Using Bounded Flooding. Technical Report CSE-TR-388-99, University of Michigan, 1999.

[18] Q. Ma and P. Steenkiste. On Path Selection for traffic with Bandwidth guarantees. In *IEEE International Conference on Network Protocols*, Atlanta, GA, October 1997.

[19] Q. Ma and P. Steenkiste. Quality-of-Service routing for traffic with performance guarantees. In *IFIP International Workshop on Quality-of-Service*, pages 115–126, New York, May 1997.

[20] Quingming Ma and Peter 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'98*, 1998 July.

[21] Klara Nahrstedt and Shigang Chen. Coexistence of QoS and Best Effort Flows - Routing and Scheduling. In *10th Tyrrhenian International Workshop on Digital Communications*, Multimedia Communications, Ischia, Italy, September 1998.

[22] V. Paxson and S. Floyd. Why we don't know how to simulate the Internet. In *Winter Simulation Conference*, Atlanta, GA, December 1997.

[23] A Shaikh. *Efficient Dynamic Routing in Wide-Area Networks*. PhD thesis, University of Michigan, June 1999.

[24] A. Shaikh and J. Rexford. www.eecs.umich.edu/∼ashaikh/research/software/routesim.html.

[25] 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 '98)*, pages 15–27, Cambridge, England, July 1998.

[26] 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 '98)*, pages 42–51, Austin TX, October 1998.

[27] B. M. Waxman. Routing of Multipoint Connections. *IEEE Journal on Selected Areas in Communications*, December 1988.

[28] B.Salkewicz Z. Zhang, C. Sanchez and E.Crawley. Quality of Service Extensions to OSPF (QOSPF). Internet Draft draft-zhang-qos-ospf-00.txt, Septemer 1997.

[29] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE INFOCOM '96*, pages 594–602, March 1996.