Implementation and Performance Measurements of QoS Routing Extensions to OSPF

G. Apostolopoulos
georgeap@cs.umd.edu
U. Maryland
College Park
Maryland, MD 20742

R. Guérin
guerin@ee.upenn.edu
U. Pennsylvania
200 S. 33rd Street
Philadelphia, PA 19104

S. Kamat
sanjay@watson.ibm.com
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights
New York, NY 10598

Abstract—**We discuss an implementation of QoS routing extensions to the OSPF routing protocol and evaluate its performance over a wide range of operating conditions. Our evaluations are aimed at assessing the cost and feasibility of QoS routing in IP networks. The results provide insight into the respective weights of the two major components of QoS routing costs, processing cost and protocol overhead and establish strong empirical evidence that the cost of QoS routing is well within the limits of modern technology and can be justified by the performance improvements.**

I. Introduction

Quality of Service (QoS) routing has recently received substantial attention in the context of its possible use in an integrated services IP network. Most of the current proposals in this context rely on the *link state* approach, e.g., see [2] for an overview. The benefits of QoS routing, in terms of improved network utilization and user service levels have been established through recent research studies [1], [7], [8], [9], [11]. However, despite these benefits, doubts regarding the feasibility of implementing QoS routing protocols in IP networks remain because of the potential additional costs that support for QoS routing entails. These added costs have two major components: *computational cost* and *protocol overhead*. The former is due to the more sophisticated and more frequent route computations, while the latter is caused by the need to distribute updates on the state of network resources that are of relevance to route computation, e.g., available link bandwidth. Such updates add to network traffic and processing overhead at routers.

Several recent works have aimed at shedding some light on the costs inherent to QoS routing. In particular, different variations of path pre-computation [3], [14], [20], [24] and path caching [21], [27] have been investigated to explore the possibility of reducing the processing cost of QoS path computation. Similarly, a variety of link cost metrics and update triggering techniques [23], [26] or path selection techniques [26] have been proposed to lower the protocol overhead of QoS routing without significantly affecting its ability to compute efficient paths. However, these works rely primarily on simulations, and as a result are not able to fully capture some of the more implementation specific issues associated with QoS routing. This work seeks to fill this gap by providing a detailed report and assessment of a complete implementation of a QoS routing protocol for IP networks.

Our implementation is based on the Open Shortest Path First (OSPF) [17] routing protocol, an Internet standard for intradomain routing that is based on a link state approach. Specifically, we added QoS routing extensions to the OSPF implementation that is available on most Unix systems as part of the the `gate daemon (gated)` [5] program. `gated` is a very popular routing protocol implementation platform, and variations of it are used in many commercial and experimental networks. We first discuss a number of important issues and design choices we faced when implementing these extensions. Next we report on the measurements made based on this implementation to obtain realistic estimates of the cost of various QoS routing operations such as path computation, link state advertisement generation and reception, etc. Furthermore, we compare the cost of our QoS enhanced version of OSPF to that of the standard OSPF protocol. Finally, we combine simulation data and the findings based on our implementation, in order to emulate the operation of a router that is part of a large QoS enabled network and get some insight into the amount of load that an "off-the-shelf," `gated` based QoS router can handle.

In Section II, we present background information on OSPF, the QoS extensions we added, and the `gated` program. In Section III, we discuss our implementation of the QoS routing extensions to the `gated` OSPF code base. In Section IV, we report on the performance measurements, and finally in Section V, we summarize our findings.

II. Background

We first provide a brief review of the current OSPF standard, focusing on the aspects that are of relevance to our QoS routing extensions. Next, we discuss the proposed QoS routing extensions of [10] that we have currently implemented. Finally, we provide a short description of the `gated` environment on which our implementation is based.

A. The OSPF Protocol

Open Shortest Path First (OSPF) [17] is a widely deployed link state routing protocol that has been an Internet standard for some time. An important characteristic of link state routing protocols is that each router maintains the full topology of the network in a *link state database*. The OSPF standard specifies that routers implementing the protocol run a shortest path Dijkstra computation on their local link state database, and determine the shortest paths to all other nodes in the network. The database is constructed and updated by means of link state advertisements, that are generated by each router and propagated to all other routers using reliable flooding.

0-7803-5417-6/99/$10.00 ©1999 IEEE.

The flooding procedure utilizes a variety of packet types: *Link State Update* (LSU) packets contain information about changes in the topology, and are used to carry multiple *Link State Advertisements* (LSAs). *Link State Acknowledgment* packets are used to acknowledge receipt of link state advertisements. Finally, *Database Description* and *Link State Request* packets are used to synchronize the link state databases of neighboring routers. There are also several types of link state advertisements (see [17] for details), with router and network link advertisements being the most relevant ones for our purpose. Router LSAs contain information about a router and **all** its interfaces, while network LSAs describe the set of routers attached to a given network. Link state advertisements are either generated periodically or are triggered by topology changes such as link failures or recoveries. These advertisements contain cost metrics that are used to compute the shortest paths.

In order to handle the scalability problems associated with both flooding and maintaining a complete network link state database, OSPF allows for a two level hierarchy of *areas* within the routing domain. Furthermore, the OSPF standard mandates a variety of constants that control the frequency of the operations related to the flooding of LSAs. In particular, the constant *MinLSInterval* specifies the minimum time between any two consecutive originations of a given LSA by a router. The default value of *MinLSInterval* is 5 seconds. Another similar constant is *MinLSArrival,* which limits the frequency at which new instances of a given LSA can be accepted. If two consecutive instances of an LSA are less than *MinLSArrival* apart, the second is not processed and simply discarded.

B. QoS Routing Extensions

Originally, the OSPF specification allowed for Type of Service (TOS) based routing in order to support the five different Types of Service that were specified for IP datagrams. This meant that routers could advertise TOS specific cost metrics in their link state advertisements that could be used to compute multiple TOS specific routing tables. Recently, the requirement for TOS-based routing was dropped due to lack of deployment. However, in order to avoid potential backward compatibility problems, routers can still advertise TOS specific metrics in their link state advertisements. This has provided an opportunity to experiment with QoS routing as an extension of the TOS features provided by standard OSPF.

The QoS routing extensions to OSPF are based on two key ideas: 1) enhancing the link state advertisements and the topology database to include network resource information (such as available bandwidth), and 2) using an alternate route computation algorithm to compute routes that take this resource information into account. Our implementation is based on the approach described in [10], which is similar to several other proposals [3], [24] for supporting QoS routing. It is limited to handling requests with bandwidth requirements, and as a result link bandwidth is the only metric extension that has been implemented. Although, one can argue that this approach is simplistic and ignores other important metrics, such as delay, we believe that the above bandwidth based model is adequate for most realistic network scenarios, where delay can be expected to be low. Furthermore, delay constraints can be handled with mechanisms such as policy, that can be used to ensure that low delay links for example are avoided. In addition, QoS routes are discovered only within a single OSPF area. Handling hierarchies of areas raises important issues that are topics for further research and beyond the scope of this paper.

[10] identifies several possible variations for QoS routing extensions, that include on-demand computation and pre-computation of QoS routes as well as both explicit and hop-by-hop routing modes. Our implementation performs path pre-computation and assumes a hop-by-hop routing mode. We chose to implement path pre-computation because of its potentially significant gains in terms of processing load, e.g., see [25]. Similarly, we opted for a hop-by-hop routing mode, simply because it can be accommodated without major changes to RSVP [12], the signaling protocol that we assume is used to request QoS guarantees, e.g., see [15].

Our implementation computes QoS paths using the *widest-shortest* path selection criterion described in [10]. At a router, a modified Bellman-Ford algorithm is used to pre-compute paths from the router to all destinations in the network. For each destination, the algorithm computes paths of all possible bandwidth values, and uses them to build a QoS routing table which is kept separate from the standard OSPF routing table (more on this later). The QoS routing table generated by the algorithm can be conceptually viewed as a matrix, where a row corresponds to a destination (entry in the IP routing table), and the ith column corresponds to paths that are no longer than i hops and have the largest amount of bandwidth available among all such paths to the specific destination. The information stored in a matrix entry includes the next hop(s) and the available bandwidth on such paths.

The information in the QoS routing table is used to identify paths capable of satisfying the bandwidth requirements of new requests. This is accomplished by comparing the amount of bandwidth requested by a new flow to the available bandwidth in successive entries in the row associated with the flow's destination. The search stops at the first entry with an available bandwidth value larger than the requested one, at which point the corresponding next hop is returned and used to determine the next hop on which to forward the request. If there is more than one next hop, our implementation chooses one of them at random with a probability that is weighted by the available bandwidth on the associated local interface corresponding to the next hop.

In addition to the changes required to both the routing table and the path computation, the OSPF protocol and code also need to be modified to support the propagation of appropriately extended link state advertisements. In particular, information about available bandwidth needs to be added to the link state database and updated through link state advertisements. This information is encoded using a new TOS field. However, only 16 bits are available to advertise the value of the metric. While 16 bits are sufficient for advertising link costs for best-effort routing, advertising bandwidth values for links ranging from few kilobits per second to many terabits per second requires more careful encoding. One such encoding scheme is described in [10] and was used in our implementation.

C. Gate Daemon (gated)

gated [5] is a popular, public domain[1] program that provides a platform for implementing routing protocols on machines running the Unix operating system. The distribution of the gated software also includes implementations of many popular routing protocols, including the OSPF protocol. The gated environment offers a variety of services useful for implementing a routing protocol. These services also facilitated implementation of some of the extensions that were required to support QoS routing. These gated services include: a) support for creation and management of timers, b) memory management, c) a simple scheduling mechanism, d) interfaces for manipulating the routing table and accessing the network, and e) route management (e.g., route prioritization and route exchange between protocols).

All gated processing is done within a single Unix process, and routing protocols are implemented as one or several *tasks*. A gated task is a collection of code associated with a Unix socket. The socket is used for the input and output of the task. gated maintains a single routing table that contains routes discovered by all the active routing protocols. A radix tree is used for fast access to routing table entries. Also, the OSPF link state database is implemented using a radix tree for fast access to a particular link state record.

III. Implementation Issues

A. Design Objectives and Scope

Our objective was to gain substantial experience with a functionally complete QoS routing implementation while also limiting the overall implementation complexity. Hence, we chose a modular architecture aimed at maximizing reuse of the existing OSPF code, minimizing changes to it, localizing the QoS extensions to specific modules and keeping their interaction with existing OSPF code to a minimum. Besides reducing the development and testing effort, this approach also facilitated experimentation with different alternatives for implementing the QoS specific features such as triggering policies for QoS related link state updates and QoS route table computation.

Several of the design choices were also influenced by our assumptions regarding the core functionalities that an early prototype implementation of QoS routing must demonstrate. Some of the important assumptions are:

- Support for hop-by-hop routing only. This affected the path structure in the QoS routing table as it only needs to store next hop information.
- Support for path pre-computation. This required the creation of a separate QoS routing table, and was motivated by the need to minimize processing overhead.
- Full integration of the QoS extensions into the gated framework, including configuration support, error logging, etc. This was required to ensure a fully functional implementation.
- Modularity to allow experimentation with different approaches, e.g., use of different update and pre-computation triggering policies with support for selection and parameterization of these policies from the gated configuration file.
- Decoupling from local traffic and resource management components, i.e., packet classifiers and schedulers and local call admission. This is supported by providing an API between QoS routing and the local traffic management module, that hides all internal details or mechanisms.
- Interface to RSVP. The implementation assumes that RSVP [12] is used to request routes with specific QoS requirements. Such requests are communicated through an interface based on [22], and we used the RSVP code developed at ISI, specifically, version 4.2a2 [13].

In addition, our implementation also relies on the following simplifying assumptions made in [10].

- The scope of QoS route computation is limited to a single area.
- All routers within the area run the QoS enabled version of OSPF.
- All interfaces on a router are QoS capable.

B. Architecture

Figure 1 depicts the software architecture of the system. There are three major components: the signaling component (RSVP in our case); the QoS routing component; and the traffic manager. In the rest of this paper we concentrate on the structure and operation of the QoS routing component. As can be seen in Figure 1, the QoS routing extensions are further divided into the following modules:

- **Update trigger module** determines when to advertise local link state updates.
- **Pre-computation trigger module** determines when to perform QoS path pre-computation.
- **Path pre-computation module** computes the QoS routing table based on the QoS specific link state information.
- **Path selection and management module** selects a path for a request with particular QoS requirements, and manages it once selected, i.e., reacts to link or reservation failures.
- **QoS routing table module** implements the QoS specific routing table, which is maintained independently of the other gated routing table.
- **Tspec mapping module** maps QoS requirements that are part of RSVP messages into the bandwidth requirements that QoS routing uses.

In the rest of this section, we outline the main functions of each of these modules.

C. QoS Routing Table and Path Pre-Computation Modules

QoS paths are pre-computed and stored in the QoS routing table as outlined in Section II-B. However, the "conceptual" matrix format described earlier is implemented differently for efficiency. Each row of the QoS routing table consists of a *path structure* in the form of a linked list. Each entry in the list corresponds to a different hop count and bandwidth value, arranged in increasing order, i.e., an entry for a given hop count is created only if it has a larger bandwidth than previous entries with a smaller hop count. This avoids having to allocate memory for entries associated with hop count values that do not correspond

[1] Access to some of the more recent versions of the gated is restricted to the GateD consortium members.

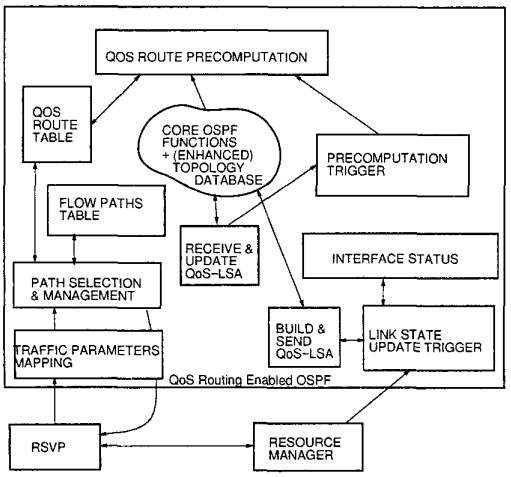


Fig. 1. The software architecture

to an increase in bandwidth. Locating the path entry for a given destination and bandwidth requirement is accomplished through a radix tree. We chose to reuse the existing radix tree structures and code of `gated` for this purpose.

An important aspect of the QoS routing table is the overhead involved in building it during the Bellman-Ford computation. During this construction phase, it is necessary to associate the link state database entities (vertices) that are being expanded by the Bellman-Ford algorithm with the corresponding path structures that contain the paths discovered so far for this vertex. This can be accomplished by using the radix tree of the QoS routing table to search for the address contained in the LSA associated with the vertex being expanded (router id in the case of a router LSA, and network id in the case of a network LSA). This provides a general, albeit inefficient solution, as it requires a full lookup in the radix tree each time a vertex is expanded in the Bellman-Ford computation. In order to avoid such a penalty in our implementation, we added a pointer directly to the path structure inside the vertex structure in the link state database. This required a small additional modification to the existing vertex structure, which had to be modified anyway to support the QoS extensions.

The last issue of significance in the construction of the QoS routing table, is allocation and de-allocation of memory. Currently, both the radix tree and the path structures are freed before a new QoS routing table is computed. This full de-allocation of the QoS routing table is potentially wasteful, especially since memory allocation and de-allocation is an expensive operation. Furthermore, because path pre-computations are typically not triggered by changes in topology, the set of destinations will usually remain the same and correspond to an unchanged radix tree. A natural optimization would then be to de-allocate only the path structures and maintain the radix tree. A further enhancement would be to maintain the path structures as well, and attempt to incrementally update them only when required because of a different number of paths with distinct hop counts and bandwidth values. However, despite the potential gains, these optimizations have not been included in our initial implementation. The main reason is that they involve subtle and numerous checks to ensure the integrity of the overall data structure at all times, e.g., correctly remove failed destinations from the radix tree and update the tree accordingly.

D. Update and Pre-computation Trigger Modules

The update trigger module determines when a router floods a new LSA to advertise changes in its link metrics. The pre-computation trigger module determines when to initiate the computation of a new QoS routing table. In order to allow for experimentation, these two modules support a number of options that can be chosen through configuration.

The **update trigger module** implements:

- A variety of triggering policies. Currently, only threshold based policies are implemented: an update is triggered when the difference between the current and previously advertised values of the available link bandwidth is larger than a configurable threshold.
- Periodic update generation.

The **pre-computation trigger module** implements:

- Periodic pre-computation.
- Triggered pre-computation each time N distinct link state advertisements have been received.

In order to implement the above functionalities, both the modules need to maintain and post their own timers and be able to receive notification of certain events of interest. Both modules need two types of timers: a) hold down timers, that are used to limit the frequency of actions (eg., metrics updates) and b) timers for periodic operation. An event significant for both modules is the change in status (up/down) of a local interface. Modules can be notified of such an event through a small addition to the OSPF code that handles the interface status changes. The pre-computation module also needs to be notified of the arrival of a link state advertisement or a timer expiration event. This can be accomplished by inserting hooks into the OSPF code responsible for receiving and processing LSAs. Similarly, the update triggering module needs to be informed of changes in the available bandwidth on local links. This is needed to ensure that the correct value is sent in the next LSA, and in some instances to determine if an update is needed. This is accomplished through the use of a simple messaging interface, that allows the resource manager to notify the update triggering module of such changes.

Note that the regular OSPF update triggering rules can interfere with the triggering policy implemented by the update triggering module. Specifically, the periodic (*RxmtInterval*) and own hold down timers (*MinLSInterval* and *MinLSArrival*) of OSPF may need to be disabled or bypassed in order to avoid interfering with the generation of QoS LSAs. One option is to disable the existing OSPF hold-down mechanisms in the case of QoS related LSAs. However, it remains necessary to implement a similar mechanism to ensure stability of the protocol during periods of overload. As a result, we opted for the simple approach of decreasing the value of *MinLSInterval* to allow more frequent QoS updates. Note that since the current resolution of `gated` timers is in seconds, one second is the smallest possible value we can specify, and it is the one we currently use in the implementation.

Delayed acknowledgments of LSAs is an OSPF feature designed to allow aggregation of acknowledgments for multiple LSAs. Since achieving a meaningful level of aggregation for acknowledgments appears to require a delay value that could interfere with QoS updates, we chose in our current implementation to bypass this mechanism altogether and immediately acknowledge LSAs received from neighboring routers. Another approach which we considered but did not implement is to make QoS LSAs unreliable, i.e., eliminate their acknowledgments, so as to avoid any potential interference. The rationale for such a design is that QoS LSAs are transmitted much more frequently and an occasional loss of a QoS LSA may only degrade the quality of a QoS route temporarily, but would not interfere with the regular OSPF operation. We plan on implementing and experimenting with such an option in the future.

E. Tspec Mapping Module

This module simply extracts information from the RSVP Tspec that describes the QoS requirements of a request, and maps it to the QoS model supported by the system, i.e., bandwidth. Currently, we support only a simple mapping where the token rate of the request is used as the bandwidth requirement of the request. Other more sophisticated mappings can be added later without affecting the rest of the system. The integrated services data structures are the same ones used by the ISI RSVP, version 4.2a2 [13].

F. Path Selection Module

The path selection module is responsible for handling incoming requests for QoS routes, e.g., triggered by the receipt of an RSVP PATH message. This is done by first using the destination information provided in the request, to search through the radix tree of the QoS routing table. This search identifies the path structure associated with the destination as discussed in Section II-B. A path is selected by stepping through the path structure, and stopping at the first entry which contains a bandwidth value larger than or equal to the requested amount. The next hop stored in this entry is then returned. If more than one next hop exists, one is chosen randomly with probability proportional to the available bandwidth on the link to the next hop.

IV. Performance Evaluation

A. Methodology

In this section, we attempt to evaluate the cost of QoS routing, when using our implementation. We explore three different dimensions in our comparisons: a) processing cost, b) message generation and reception cost, and c) memory requirements. For processing cost, we further subdivide it into path pre-computation and path selection costs.

Most of the above costs can be measured individually or as stand-alone operations. For example, the time needed for a single path pre-computation, or the size of the QoS routing table can be estimated based on a single router, whose link state topology database has been populated using some external mechanism. The same holds for measuring the time it takes to select a path. Even the cost of receiving or originating LSAs can be measured reasonably accurately by using only two routers after they form an adjacency and start exchanging LSAs. Thus, it is possible, with a minimum amount of equipment, to obtain good atomic estimates of the cost of all individual operations of interest. We refer to this type of performance measurements as "stand-alone" evaluation mode. Stand-alone performance results alone are, however, not sufficient to provide a complete assessment of the impact of QoS routing on a router's operation. This is because, while this accurately estimates the intrinsic cost of QoS related operations, it does not fully capture the many dependencies and interactions that take place in a real operational environment. These affect performance, if only because they determine the frequency and timing of many of those operations, and these parameters are difficult to estimate without a full scale network environment.

In order to address this shortcoming of the stand-alone measurement mode, we propose to combine its results with simulations that we use to create conditions that appear in a large network. Specifically, we define a simulation environment that allows us to specify an operational network with the following parameters: a) network topology, b) traffic characteristics such as size of requests, arrival rates and distribution of request sources and destinations, and c) choice of path pre-computation and link state update generation trigger policies in the routers. Each of the above parameters is "tuned" based on our previous experience with this simulation environment, so as to correspond to representative and realistic operational conditions as described below.

While performing a simulation run, we generate a log that contains the time at which each of the following operations occurred at each node: a) generation of an LSA, b) reception of an LSA, c) initiation of path pre-computation and, d) initiation of path selection. The information gathered in the simulation logs is then used to derive operational costs of the test-node router. This is accomplished by using the individual operation costs derived from the stand-alone experiments to compute a cumulative cost at the node.

It is important to note certain approximations inherent in the above method. First, the simulator (a modified version of MaRS [6] built for previous studies [26]) used to derive the operations and timing logs, does not exactly mimic the behavior of the OSPF protocol. The main difference is that OSPF implements two types of LSAs, router and network LSAs, and the simulator assumes that only router LSAs are sent. However, router LSAs represent the bulk of LSAs when operating a QoS routing enabled domain, with network LSAs being originated only in case of topology changes which we anyhow do not consider in our simulation. Another discrepancy between the simulator and the real implementation, is the minimum spacing of 1 second that the implementation imposes between the transmission of two consecutive LSAs. This affects the arrival patterns of LSAs and is also likely to cause the combination of multiple LSAs into a single OSPF network packet, which lowers their transmission overhead. These effects are not captured in the simulator, which does not impose any minimum spacing between consecutive LSAs, and further assumes that each LSA is transmitted in its own OSPF packet. This can lead to slightly larger estimates for operational costs since the reception of multiple individual LSAs is likely to be more expensive than the reception of a sin-

gle OSPF packet containing multiple LSAs.

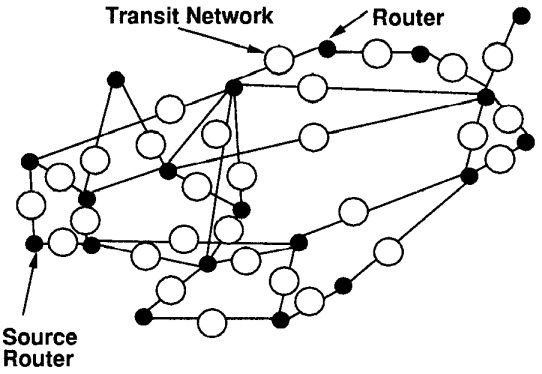
In all experiments, the test systems used are IBM Intellistation PCs each with a Pentium Pro 266 MHz processor, 32 Mbytes of real memory, 3.4 Gbytes of disk, running FreeBSD 2.2.7-RELEASE and `gated 4` software. The Ethernet adapters used in the tests are 100 Mbit/second Intel PCI.

B. Stand-Alone Cost

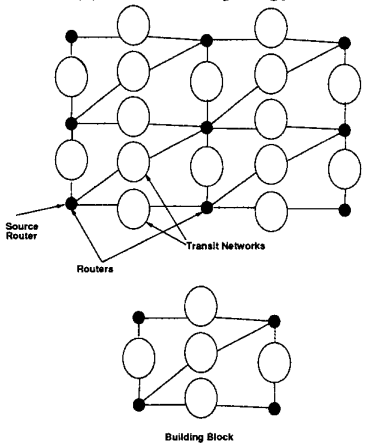
It is important to note that QoS routing specific costs depend not only on the network topology, as is the case for the standard OSPF protocol, but also on the distribution of available bandwidth on links. This is because path pre-computation maintains alternate paths with bottleneck bandwidth larger than that of minimum hop paths. As a result, the number of distinct paths to a given destination varies according to the distribution of available bandwidth on network links.

One possible approach to factoring this dependence in our cost evaluations is to attempt to identify, for a particular network topology, an assignment of link bandwidths that results in the maximum distinct number of paths being generated when computing the QoS routing table. Unfortunately, such a direct approach is infeasible since it requires the enumeration of an exponential number of paths. This makes the problem difficult even in small topologies. As an intermediate solution, we propose to compute both best and average cases for those costs that depend on link bandwidth distribution. Specifically, we obtain the best case for a given topology by defaulting the path pre-computation algorithm to a minimum hop count algorithm. This yields at most one path to each destination. Next, we obtain an estimate of the average case by executing the path pre-computation algorithm for a sequence of random link bandwidth assignments, and averaging the resulting costs.

Finally, in order to perform any of the above measurements, we need to first artificially populate the link state topology database of the routers with the corresponding set of entries. We consider two types of topologies. The first one is a topology that has been used in a number of previous studies, and is representative of the network topology of a typical Internet service provider in the US. This is the `isp` topology shown in Figure 2(a), where all nodes correspond to router nodes interconnected by transit network links. In the `isp` topology, the maximum path length for the Bellman-Ford computation was set to 16 hops. The topology is dimensioned for uniform traffic, assuming minimum hop routing, with link capacities range between 20 and 70 Mbits/second. The second type of topology we consider is an artificial mesh like topology that is constructed by repeating a basic building block. The basic building block which consists of 4 routers and 5 transit networks is depicted in Figure 2(b). An $N \times N$ mesh topology is constructed by repeating the building block along two dimensions. Figure 2(b), also illustrates 2×2 mesh topology. In this topology, all links are assumed to have a capacity of 45 Mbits/second. In our experiments we use instances of this topology ranging from 1×1 to 10×10. For a topology of size $N \times N$, the maximum hop limit for the Bellman-Ford computation was set to $N + 2$. In both cases, path pre-computation were performed with the node indicated in Figure 2(a) as source. Variations of the measured pre-computation times when different nodes were used as sources were small.



(a) The `isp` topology



(b) The variable size mesh topology

Fig. 2. Topologies used in cost measurements

B.1 QoS Routing Table Computation

Varying the size of the mesh topology allows us to observe how the computation cost varies with network size. A recent survey [18] of vendors who have deployed OSPF in real networks, reports that the figure for the number of routers in one area ranges from 20 to 350 with 100 being the median and 160 being the mean. We show results for networks of up to 400 nodes (that include both routers and transit networks). The time needed for computing the standard SPF tree and the QoS routing table for the `mesh` topology are shown in Figure 3. In the case of QoS routing, the pre-computation times include the cost of de-allocating the previous QoS routing table, and results are shown for the best and average case. In both cases, the processing cost of QoS path pre-computation is not significantly larger than that of the SPF computation. However, one should remember that the frequency of QoS path pre-computation would be significantly larger than that of the SPF computation, and this is an aspect which we investigate in Section IV-C.

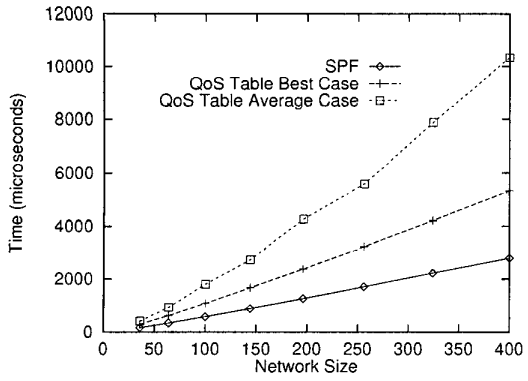


Fig. 3. Processing time for path computation

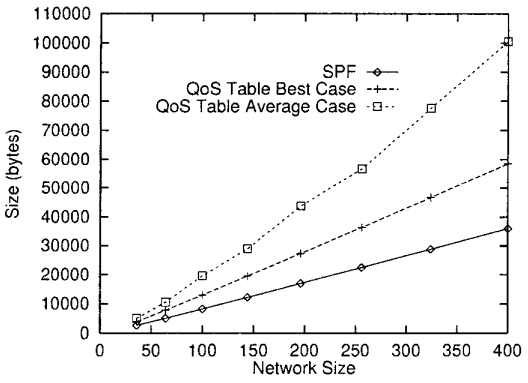


Fig. 4. Comparison of memory requirements

B.2 Memory Requirements of the QoS Routing Table

Figure 4 illustrates the differences in memory requirements between the QoS routing table and the standard `gated` routing table (with OSPF as the only configured active routing protocol) for different network sizes. As before, the `mesh` topology is used to generate networks of varying size. Specifically, while the memory requirements for QoS routing are clearly higher, given the cost and availability of memory, the difference is again not extremely significant, e.g., about a factor of 3 for the average case. This difference was expected since both tables contain a radix tree of all destinations, and the QoS routing table requires additional storage for the path structures.

B.3 Cost of Path Selection

Path selection consists of accessing the QoS routing table for a given destination and bandwidth value and returning a suitable path (next hop). Accessing the routing table requires first a lookup in the radix tree based on the destination, and second a search of the path structure associated with the destination until a path capable of satisfying the requested bandwidth is found. The cost of these operations is shown in Figure 5 for both the best and average cases. As mentioned earlier, the best case is obtained by forcing all path structures to contain only one entry - the one corresponding to the minimum hop count path. As far as the average case is concerned, there are two dimensions

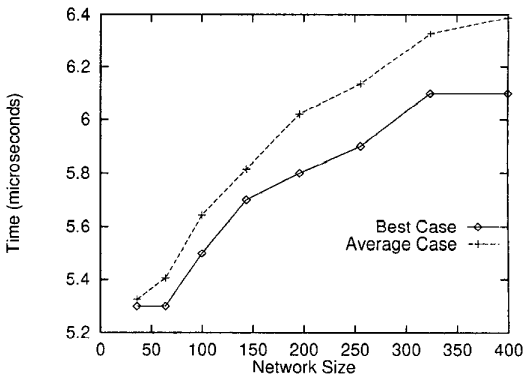


Fig. 5. Cost of path selection

along which averaging can take place. For a given topology and distribution of link bandwidth, averaging can be done over the destination nodes or the possible values for the requested bandwidth. The destination affects not only the cost of the lookup in the radix tree, but also the potential depth of the search in the path structure associated with the destination, as the number of entries in the path structure is likely to differ from destination to destination. The search in the path structure is also affected by the requested bandwidth as large values will typically require stepping through more entries in the path structure. In our measurements, we average based only on destinations, and the average is computed across all possible destinations in the network. Averaging based on bandwidth is avoided by forcing all requests to be for an amount of bandwidth larger than the capacity of the network links. This results in the maximum search time through the path structure, so that we are in effect measuring an average worst case.

B.4 Link State Advertisements Generation and Reception

The last set of parameters, whose costs we want to estimate, are common to the standard and QoS routing versions of OSPF. They consist of the cost of generating and receiving LSAs, that are incurred in both cases.

The time required for generating and receiving LSAs was measured using two similar machines connected to each other, and running `gated` with our QoS routing enabled version of OSPF. The machines were configured so as to form an OSPF adjacency, and then exchange LSAs between them. The OSPF hold down mechanism based on *MinLSInterval* and *MinLSArrival* was disabled, in order to get accurate measurements of the LSA generation and reception costs. According to our measurements, either generating or receiving an LSA takes about 900 microseconds in our test system.

C. Operational Cost

In this section, we apply the method described in Section IV-A to estimate the actual operational behavior of a router running our QoS routing enabled version of OSPF. Our first task consists, therefore, of simulating a complete network and recording operation and timing logs at a test router in the simulation. These logs are then used to derive estimates of the test router's

performance. In order to obtain sample points representative of different network sizes and topologies, we run two simulations, one for the `isp` topology and the other for an 8 × 8 `mesh`. The `isp` topology is characteristic of a typical medium sized ISP network and the 8 × 8 `mesh` topology with its 284 nodes provides an example of a fairly large network.

In all simulations, we generate a workload assuming that requests arrive according to a Poisson distribution, and are independent and uniformly distributed across source-destination pairs. The mean request inter-arrival time at a node is set to 15 seconds, and bandwidth requirements are uniformly distributed between a minimum of 64 Kbits/sec and a maximum of 5 Mbits/sec. The duration of requests is assumed to be exponentially distributed with a mean of 3 minutes. The resulting workload corresponds to reasonably realistic operating conditions of moderate overload, with a small but non-zero blocking probability (less than 5% for the `isp`, 12% for the `mesh` topology). In addition to this workload configuration, the simulations also assume specific values for several operational parameters of the protocol. In particular, path pre-computations are performed periodically, for period values of 1 second (the minimum allowed by our implementation, given the available timer precision), 5 seconds, and 50 seconds. Similarly, a threshold based mechanism was chosen to trigger the generation of LSAs, and two different threshold values of 10% and 80% were used. A value of 10% provides very precise link state information but corresponds to a larger level of LSA traffic. A value of 80% substantially reduces the amount of LSA traffic, at the cost of greater inaccuracy in link state information. We believe that these combinations of parameter settings provide a reasonably comprehensive coverage of different operational environments.

C.1 Router Load

The measured processing load is reported in Table I for both the `isp` and `mesh` topologies, and for the different combinations of path pre-computation period and link state update threshold mentioned above. In addition to router load, the table also gives the bandwidth blocking ratio for each combination, denoted as BR. We measured the load on all the routers in intervals of 50 milliseconds. For each router we kept track of the average utilization of its CPU due to routing related processing and also the peak utilization that was observed in a single 50 millisecond period. For each of these two quantities, we show the maximum and minimum values that were observed over all routers in the network.

The results seem to indicate that given today's processor technology, QoS routing should not be viewed as a costly enhancement, at least not in terms of its processing requirements. Although the peak load can be relative high (but under 84% even for the large `mesh` topology), the average load is very low, indicating that the periods of high load are very few.

For large pre-computation periods, increasing the link state update threshold from 10% to 80% reduces the average processing load by a factor between 5 and 10, while the corresponding loss in performance, i.e., the increase in blocking probability, is by a factor of less than 2. This means that when operating with large pre-computation periods, increasing the update threshold is a cost effective trade-off. This is in line with the findings of [26]. On the other hand, for small pre-computation periods, processing cost savings are smaller and have the same magnitude as routing performance loss. When the update triggering threshold is 10%, the contribution of the link state update processing to the overall processing cost is very significant.

	Update Triggering Threshold		
Period	10%		
	Max	Avg	BR
1 sec	36%/9%	.27%/.06%	1.5%
5 sec	36%/7%	.26%/.05%	1.6%
50 sec	30%/7%	.24%/.05%	3.5%
	80%		
	Max	Avg	BR
1 sec	31%/8%	.02%/.01%	2.7%
5 sec	31%/8%	.02%/.01%	3.2%
50 sec	30%/7%	.02%/.01%	4.3%

`isp`

Period	10%		
	Max	Avg	BR
1 sec	84%/36%	1.60%/.71%	5%
5 sec	84%/36%	1.27%/.53%	5.4%
50 sec	84%/36%	1.26%/.53%	12.2%
	80%		
	Max	Avg	BR
1 sec	82%/35%	.22%/.15%	10%
5 sec	82%/33%	.15%/.06%	10.2%
50 sec	80%/31%	.12%/.05%	14%

8 × 8 `mesh`

TABLE I

ROUTER UTILIZATION AND BANDWIDTH BLOCKING

	Update Triggering Threshold			
Period	10%		80%	
(sec)	Peak	Avg	Peak	Avg
1	67.7/28.6	4/1.5	52/17.4	.353/.131
5	60.7/23.5	4/1.4	51.7/16.5	.349/.128
50	49.8/16.9	3.8/1.4	49/16.5	.327/.128

`isp`

1	169/109	29.4/15.4	155/107	2.9/1.5
5	174/110	29.7/15.3	157/107	2.9/1.5
50	161/110	25.3/13.2	157/107	2.9/1.5

8 × 8 `mesh`

TABLE II

LSA BANDWIDTH CONSUMPTION (KBYTES/SECOND)

C.2 Bandwidth Requirements of LSAs

The last important cost component is the amount of link bandwidth that is consumed by LSAs. Not only do LSAs contribute to the processing load of the router, but the bandwidth needed to transmit them cannot be used by regular data traffic. This bandwidth consumption can be computed for each link using

the timing information contained in the simulation logs and our knowledge of the LSA format. A router LSA in OSPF has a size of $88 + 16 \times l$ bytes, where l is the number of links of the originating router, and is acknowledged with a link state acknowledgment packet that has a size of 28 bytes.

In Table II we show the peak and average bandwidth consumption (in Kbytes/second) that was observed due to LSA traffic. This traffic was again measured in periods of 50 milliseconds and the peak and minimum values across all network links are reported. As expected, a large update threshold results in lower LSA traffic. The path pre-computation period does not significantly affect the average volume of update traffic. Variations that appear for the peak volume are mainly due to the lower confidence intervals inherent to reporting the maximum and minimum numbers across all network links.

The above numbers indicate that even with the more frequent updates that QoS routing requires, the bandwidth consumption of protocol traffic should not be of significant concern, at least not for the kind of link bandwidth we have assumed. Even the maximum volume of update traffic does not exceed 174 Kbytes/sec, a value quite small for links of capacities in the tens of megabits range. On the average, the volume of update traffic is much smaller, and does not exceed 29.4 Kbytes/sec for the cases we tested.

V. Conclusion

In this paper, we reported on a detailed evaluation of the overhead incurred by extensions needed to support QoS routing in the OSPF protocol. Our investigation, based on an actual implementation of these extensions, showed that the increased processing cost of QoS routing is not excessive, and remains well within the capabilities of medium-range modern processors. In the worst case which we tested, path pre-computation took only 10 milliseconds. In general, we found processor utilization in our test system to be quite low, leaving room to operate at even higher frequencies of path pre-computation than the ones we used. In addition, we verified that while LSA generation and reception costs are indeed a major cost component of QoS routing, they remain tolerable even for large networks. More important, bandwidth consumption associated with LSA traffic was also found to represent only a small fraction of link bandwidth.

References

[1] H. Ahmadi, J. S.-C. Chen, and R. Guérin, "Dynamic Routing and Call Control in High-Speed Integrated Networks." In Proceedings Workshop Sys. Eng. Traf. Eng., ITC'13, 1991.

[2] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A Framework for QoS-based Routing in the Internet." RFC 2386, Category Informational, August 1998.

[3] Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees." In Proceedings IEEE International Conference on Network Protocols, Atlanta, Georgia, October 1997.

[4] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest Paths Algorithms: Theory and Experimental Evaluation." In Proceedings 5th Annual ACM SIAM Symposium on Discrete Algorithms, pages 516–525, Arlington, VA, January 1994,

[5] The GateDaemon (GateD) Project, Merit GateD Consortium, `http://www.gated.org`.

[6] C. Alaettinoglu, A. U. Shankar K. Dussa-Zieger, and I. Matta, "Design and Implementation of MaRS: A Routing Testbed." Journal of Internetworking Research and Experience, 5(1):17-41, 1994.

[7] Z. Wang, and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications." IEEE Journal Selected Areas in Communications, 14(7):1228-1234, 1996.

[8] W. C. Lee, M. G. Hluchyj, and P. A. Humblet, "Routing Subject to Quality of Service Constraints in Integrated Communication Networks." IEEE Networks, pages 46-55, July/August 1995.

[9] R. Widyonon, "The Design and Evaluation of Routing Algorithms for real-time Channels." Technical Report TR-94-024, University of California at Berkeley, June 1994.

[10] G. Apostolopoulos, R. Guérin, S. Kamat, A. Orda, T. Przygienda, and D. Williams, "QoS Routing Mechanisms and OSPF Extensions." Internet Draft, work in progress, January 1998.

[11] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Two Distributed Algorithms for the Constrained Steiner Tree Problem." In Proceedings 2nd International Conference on Computer Communication and Networking, pages 343-349, 1993.

[12] R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP), Version 1, Functional Specification." Internet Engineering Task Force, Request For Comments (Proposed Standard), RFC 2205, September 1997.

[13] The RSVP project, `http://www.isi.edu/div7/rsvp/`.

[14] R. Guérin, A. Orda, and D. Williams, "QoS Routing Mechanisms and OSPF Extensions." In Proceedings GLOBECOM'97, Phoenix, AZ, November, 1997.

[15] R. Guérin, S. Kamat, and S. Herzog. "QoS path management with RSVP." In Proceedings GLOBECOM'97, Phoenix, AZ, November, 1997.

[16] Q. Ma, P. Steenkiste, and H. Zhang, "Routing High-Bandwidth Traffic in Max-Min Fair Share Networks." In Proceedings SIGCOMM'97, 206-217, 1997.

[17] J. Moy, OSPF Version 2, Internet Request for Comments, RFC 2178, July 1997.

[18] J. Moy, OSPF Standardization Report, Internet Request for Comments, RFC 2329, April 1998.

[19] S. Gupta, "Performance Modeling and Management of High-Speed Networks." Ph.D. Thesis, University of Pennsylvania, 1993.

[20] J.-Y. Le Boudec and T. Przygienda, "A Route Precomputation Algorithm for Integrated Services Networks." Journal of Networks and Systems Management, vol. 3, no. 4, pages 427-449, 1995.

[21] M. Peyravian and A. D. Kshemkalyani, "Network Path Caching: Issues, Algorithms, and a Simulation Study." Computer Communications, vol. 20, pages 605-614, 1997.

[22] R. Guérin, S. Kamat, and E. Rosen, "Extended RSVP-Routing Interface." Internet Draft, work in progress, July 1997.

[23] A. Shaikh, J. Rexford, and K. Shin, "Dynamics of quality-of-service routing with inaccurate link-state information." University of Michigan Technical Report CSE-TR-350-97, November 1997. A more recent version is in submission, May 1998.

[24] A. Shaikh, J. Rexford and K. Shin, "Efficient Precomputation of Quality-of-Service Routes.", in Proceedings of Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Cambridge, U.K., July 1998.

[25] G. Apostolopoulos and S. K. Tripathi, "On the Effectiveness of Path Precomputation in Reducing the Processing Cost of On-demand QoS Path Computation." In Proceedings IEEE Symposium on Computer and Communication, Athens, Greece, June 1998.

[26] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi, "QoS Routing: A Performance Perspective." in Proceedings of SIGCOMM, Vancouver, B.C., Canada, September 1998.

[27] G. Apostolopoulos, R. Guérin, S. Kamat, and S. K. Tripathi, "On Reducing the Processing Cost of On-Demand QoS Path Computation." To appear in Journal of High Speed Networking.