Quality of Service Based Routing: A Performance Perspective

George Apostolopoulos
Computer Science Department
University of Maryland
College Park, MD 20742

Roch Guérin, Sanjay Kamat
IBM T. J. Watson
Research Center
Yorktown Heights, NY 10598

Satish K. Tripathi
Bourns College of Engineering
University of California
Riverside, CA 92521

Abstract

Recent studies provide evidence that Quality of Service (QoS) routing can provide increased network utilization compared to routing that is not sensitive to QoS requirements of traffic. However, there are still strong concerns about the increased cost of QoS routing, both in terms of more complex and frequent computations and increased routing protocol overhead. The main goals of this paper are to study these two cost components, and propose solutions that achieve good routing performance with reduced processing cost. First, we identify the parameters that determine the protocol traffic overhead, namely (a) policy for triggering updates, (b) sensitivity of this policy, and (c) clamp down timers that limit the rate of updates. Using simulation, we study the relative significance of these factors and investigate the relationship between routing performance and the amount of update traffic. In addition, we explore a range of design options to reduce the processing cost of QoS routing algorithms, and study their effect on routing performance. Based on the conclusions of these studies, we develop extensions to the basic QoS routing, that can achieve good routing performance with limited update generation rates. The paper also addresses the impact on the results of a number of secondary factors such as topology, high level admission control, and characteristics of network traffic.

Keywords: QoS routing, Performance evaluation, Link state routing, Path pre-computation

1 Introduction

1.1 Background

What is QoS routing, what are its goals, and what are the issues it faces? These are some of the questions we will try to address in this paper, and in particular the latter. QoS routing is the process of selecting the path to be used by the packets of a flow based on its QoS requirements, e.g., bandwidth or delay. The exact definition of a flow is not important as long as it involves the same ingress and egress points from the network. The motivation for using a path selection that is sensitive to these requirements is the hope that it will help improve both the service received by users and the overall network efficiency. The improvement to the service received by users is in the form of an increased likelihood of finding a path that meets their QoS requirements. Conversely, the improvement to network efficiency is usually in terms of increase in *revenue*, where revenue is typically a function of the number of flows or the amount of bandwidth carried by the network.

Although a number of works, e.g., [6, 9, 10, 11, 12, 14] present evidence that QoS routing can potentially achieve these goals, there are still practical concerns about the value and feasibility of implementing QoS routing protocols in a network. Specifically, is the increase in performance worth the added cost? The added cost of QoS routing has two major components: *computational cost* and *protocol overhead*. The former is due to the more sophisticated and more frequent path selection computations, and the latter is caused by the need to distribute updates on the state of network resources that are of relevance to path selection, e.g., available link bandwidth. Such updates translate into additional network traffic and processing. Note that we assume here the use of a *link state* protocol. This is consistent with many of the proposals being currently put forward for QoS routing, e.g., see [3] for an overview.

While increases in computational complexity can usually be offset by leveraging the technology curve, i.e., faster processors and bigger memories, an increased volume of protocol traffic contributes to higher cost along multiple dimensions, e.g., bandwidth, storage, update processing, and the associated context switching overheads. Thus, the cost increase due to higher protocol overhead is harder to contain and, therefore, often considered a primary inhibitor to the deployment of QoS routing. A major goal of this paper is to understand how much that cost component can be reduced while preserving most of the performance gains of QoS routing.

The efficiency gains of QoS routing can be measured by comparing the network revenue (e.g., carried traffic) using QoS routing to that obtained using a simpler routing protocol that is oblivious to QoS requirements. There are several parameters that influence the outcome of such a comparison. The first is the path selection algorithm itself, i.e., its ability to select "good" paths. The second is the accuracy of the information on which the path selection algorithm operates, i.e., the state of the network resources. These two components are directly related to the previous aspects of computational cost and protocol overhead, respectively. As a result, we focus the investigations of this paper on understanding the issues and trade-offs they involve.

Clearly, there are several other parameters that influence the overall performance of a QoS routing solution. A very important one is the characteristics of the network traffic. QoS sensitive routing attempts to improve network utilization by diverting traffic to paths that would have not been discovered by traditional, non QoS sensitive routing. This may be impossible if due to traffic patterns and network dimensioning all paths are equally loaded. We believe that QoS routing will be more useful and more effective in environments where traffic and network capacity are mismatched and alternate paths with lower load exist. In practice, both network element failures and changing traffic patterns will very often create such mismatches. Our experiments are designed so that we can investigate the performance of our solutions under such conditions.

Other parameters that can affect the operation and performance of QoS routing include the network topology and the choice of high level admission control policies. Different network topologies may be better suited to certain algorithms, thereby affecting their performance. Admission control is of significance as it is often desirable to reject a request, even when a feasible path has been found, if admitting the request will lead to inefficient use of network resources. Failure to put in such protections can result in throughput degradation in cases of overload [1, 2].

Ignoring these aspects can distort conclusions on the relative performance of QoS routing schemes, and it is important to make sure they are accounted for when comparing different solutions. However, they represent factors that are, to some extent, "external" to the core QoS routing issues, namely computational cost and protocol overhead. Hence, our investigation will initially focus on the impact of parameters directly linked to these two primary issues, and assume fixed network topology and high level admission control strategy. Sensitivity to variations along these dimensions will be explored, but only after having narrowed down the field of solutions based on the outcome of our primary investigation.

We list below parameters of interest to characterize the behavior and properties of different QoS routing solutions along the dimensions of computational cost and protocol overhead, and also try to identify the relations that exist between them.

1.2 Computational Cost Parameters

The following parameters strongly influence the computational cost of a QoS routing solution:

- Path selection criteria: e.g., minimize hop count, widest path, etc. Sophisticated path selection algorithms attempt to optimize multiple criteria as well as satisfy several additional constraints. The trade-off is between the ability to identify paths that are cheaper and/or better matched to the requirements of a request, and the computational complexity of the algorithm.

- Trigger for path selection computations: e.g., for each request, periodically, upon receipt of a network state update, etc. The trade-off is between the amortized per request computational cost and the "goodness" of the selected paths. For example, computing a new path for each request allows its selection to be based on the most recent network state information. However, if a large number of requests are received between consecutive network state updates, it may be more cost effective to pre-compute paths.

- Flexibility in supporting alternate path selection choices: e.g., maintaining equal cost choices and selecting among them, accounting for inaccuracy in network state information, etc. In general, the unavoidable inaccuracy in the network state information has implications for the path selection process. In particular, it may be desirable to maintain and alternate between several choices in order to avoid being "stuck" with a single bad choice caused by inaccurate information. Similarly, relying on strict cost minimization criteria may not be justified when cost information is relatively inaccurate. In such cases, relaxing the optimization criteria of the path selection process may yield more robust solutions. However, allowing such options can affect the overall computational cost.

1.3 Protocol Overhead Parameters

There are two main parameters that seriously impact the protocol overhead cost. The first is the triggers for network state updates, i.e., when does a node decide to inform the rest of the network about changes in the state of one or more of its links. The mechanisms used to trigger such updates can be classified as follows:

- Relative change or threshold based triggers: An update is triggered when the relative difference between the current and the previously advertised link state exceeds a certain threshold, e.g., 50%.

- Absolute change or class based triggers: Link bandwidth is divided into adjacent bandwidth classes and an update is triggered when the current link state value crosses a class boundary. Such schemes can be further classified based on the spacing between class boundaries, e.g., fixed-size partitioning or exponentially distributed class sizes.

- Timer based triggers: Timer based triggers may be used to generate updates at fixed intervals or used to enforce a minimum spacing between two consecutive updates. The latter, referred as *clamp-down* or *hold-down* timers, are often used in conjunction with one of the above "change" based schemes to control the volume of updates when network state oscillates in a narrow range.

The trade-off in each of the above schemes is between the volume of updates and the accuracy of state information available to path selection.

The other parameter that signifantly affects QoS routing protocol overhead is the update contents. This refers to the scope of an update message and the type of value advertised for metrics. Specifically, whenever a trigger is activated, the node's update message can cover only the specific link involved or all of the node's links. The trade-off is between adding unnecessary traffic and processing overhead when there is little change in the state of other links, and amortizing the cost of processing updates over many links and eliminating the need for some future updates. Additionally, when updating the state of a link, there is a choice of advertising the exact state at the time of the update, or some quantized value from a fixed set of values. Quantizing the advertised value influences when the next update takes place, but more importantly, it can affect the path selection process. Specifically, limiting updates to quantized values on one hand impacts the accuracy of link state information, but on the other hand, it increases the number of equal cost paths. The former can affect the ability of path selection to identify a "good" path, but the latter can help the path selection avoid being stuck with a single "bad" choice. The outcome of such a trade-off depends on many of the above parameters, and is one of the aspects investigated in the rest of the paper.

The nature of inaccuracy introduced by various choices for the triggers of the network state updates can be very different. In the absence of a clamp down timer, inaccuracy is primarily determined by the sensitivity of the triggering policy (number and size of classes or the threshold value). In this case, since the details of the triggering policy are known, it is possible to infer a reasonable range for the actual link metric value at any instant given its last advertised value. Due to this property we call this type of inaccuracy, *systematic* inaccuracy. This property can be exploited by a clever path selection algorithm to suitably account for the inaccuracy and thereby increase the probability of success (see [8] for some examples). However, often a large clamp down timer is used since that is the only direct means of controlling the protocol overhead and limiting it to a tolerable level. Unfortunately, the degree of inaccuracy in this case is much larger and almost impossible to estimate. We will term such inaccuracy *random* inaccuracy. A different approach is then needed to cope with this type of inaccuracy. In this paper, we focus primarily on "low cost" solutions where relatively large clamp-down timer values are used to significantly lower the protocol overheads. Hence, we address primarily the issue of random inaccuracy and the means of coping with it.

1.4 Paper Outline

The rest of the paper is organized as follows. In Section 2, we describe the simulation environment we used for our studies. Section 3 presents a classification of update trigger policies and their evaluation based on the protocol overhead they incur. In Section 4, we investigate the routing performance of different policies when large clamp-down timers are used to reduce their protocol overhead. Section 5 discusses methods to further improve the performance of these policies by introducing mechanisms to cope with the random inaccuracy due to the large clamp-down timer values. The dimension of computational cost is investigated next in Section 6, for each of the solutions of Section 5. Finally, Section 7 summarizes our main results.

2 Evaluation Environment

The focus of our work is on evaluating the performance of unicast Quality of Service based routing and its sensitivity to various factors. We use simulation as the main tool for performing this evaluation. The simulator used is based on the Maryland Routing Simulator (MaRS) [16]. MaRS was designed as an event driven routing simulator for both link state and distance vector routing protocols and has been extended appropriately for handling QoS routing.

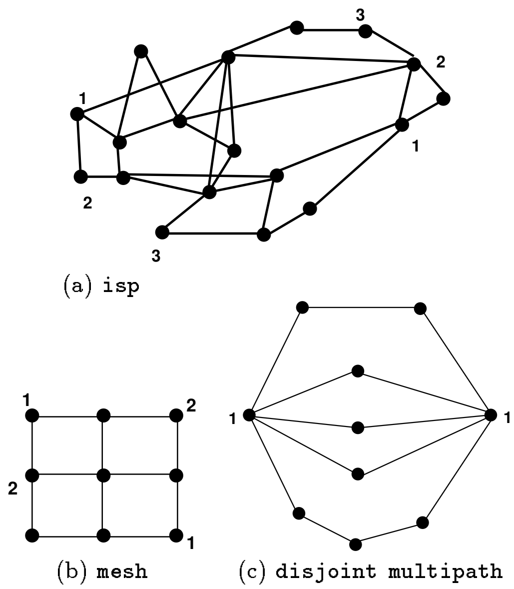


Figure 1: Topologies used in the experiments

2.1 Network Topologies

Figure 1 shows the topologies that we will be using for our experiments. Links in all topologies have propagation delay of 1 millisecond and never fail. The topology in 1(a) is similar to the one used in [7, 14] and typical of a large ISP's network. The `mesh` topology of 1(b), although somewhat contrived, is of interest due to its regularity and large number of equal hop length multi-paths. In addition, the topology of 1(c) will be used in Section 5 to evaluate the impact of topology on certain modifications to the widest-shortest path selection algorithm we consider to cope with inaccuracies in link state metrics. Unlike the previous two topologies, this topology exhibits multiple equal cost paths that do not share links. The bulk of the results reported in this paper will be for the `isp` topology. Whenever results strongly depend on the characteristics of the network topology, this will be clearly noted.

2.2 Traffic Model

We model traffic in terms of requests for setting up flows with specific bandwidth requirements. Thus, a request is characterized by its source, destination and bandwidth requirement. Requests are assumed to arrive independently at each node, following a Poisson distribution. Request duration, i.e., the holding time of a flow, is assumed to be an exponentially distributed random variable. Requested bandwidth is uniformly distributed in the interval $[L, U]$ where L is 64 Kbps and U is varied for different simulation runs to obtain different load conditions. While we concern ourselves with arrival and departure of flows, we do not model the data traffic of the flows; the amount of available bandwidth on an interface is determined based on the bandwidth reservations on this particular interface.

For a given mean arrival rate of requests, traffic load can be considered proportional to the mean bandwidth request size and the mean duration of flows. We use different combinations of these values to evaluate the effects of these parameters. U is set to 1, 5 and 25 Mbits/sec. The value for mean request duration time is for most experiments set to 3 minutes. When requests are very large, there is an increased chance of loss of revenue due to bandwidth fragmentation. On the other hand, with small requests, the impact of a bad routing decision is likely to be smaller, potentially decreasing the sensitivity of routing performance to the various parameters we are investigating. For this reason we will mainly concentrate on cases of medium requests sizes, i.e., 5 Mbits/sec although results for smaller requests are also shown.

When deciding on the type of traffic patterns to be used, the relationship of the traffic pattern and the topology must be considered. In order to emulate conditions of mismatch between traffic and network topology we introduce non-uniform traffic with increased traffic levels between particular hot spot nodes. Nodes that participate in a hot spot pair exchange increased levels of traffic with the other pair member. Although there are multiple pairs of hot-spot nodes at any instance there is only one active pair in the network. The active pair alternates over the period of the experiment to reduce the dependencies on the network topology. The pairs of hot-spot traffic nodes for each topology are shown in Figure 1, marked with the number of the hot-spot pair they belong to. In the case of non-uniform traffic there are two request arrival rates. The *background* rate for traffic between non hot-spot nodes and the *foreground* rate for the hot-spot nodes. Clearly, the higher the levels of the background traffic the closer we get to a uniform traffic model. If on the other hand the background traffic is very low we are operating under very unrealistic conditions.

For each of the topologies of Figure 1 links are dimensioned for uniform traffic between nodes and they are in the range of 30-80 Mbits/sec for the `isp` topology and 100-140 Mbits/sec for `mesh`. For the `disjoint multipath` topology all links are 100 Mbits/second. For these link dimensions and a request duration of 3 minutes, for hot-spot traffic over the `isp` topology we chose an average offered load of 150 and 192 Mbits/sec for 5 and 1 Mbits/sec requests respectively. The For the `mesh` topology (again for 3 minute request duration) and hot-spot traffic, the average offered load was 675 and 761 Mbits/sec. These loads were selected so that the networks were operated in a realistic region, i.e., with small but non-zero blocking.

2.3 Path Selection

In this study, we primarily focus on the *widest-shortest* path selection algorithm described in [13] and some of its variants. We use an implementation based on the Bellman-Ford algorithm although the Dijkstra algorithm could also be used. The basic idea behind the algorithm is as follows. Consider a source node s and a destination node d. Let $S_d =< S_d^1, S_d^2, \ldots, S_d^n >$ be an ordered list of sets where S_d^h is the set of paths from s to d of hop length h that have the largest (among all paths of hop length h) bottleneck bandwidth. Conceptually, the path selection algorithm can be thought of as considering candidate paths from the sets S_d in increasing order of hop count and choosing one that has adequate bandwidth to meet the flow's requirement.

For each request, an *explicit route* that describes the whole path to the destination is generated. Initially, we use the above algorithm in an on-demand mode. Later in the paper, we consider variations such as pre-computing paths in an effort to reduce computational cost and relaxing the strict order of investigating paths in S_d as a means of coping with inaccuracy in link metrics.

An issue that needs some further elaboration is that of selecting a particular path from a set of equal cost multipaths. In order to achieve some degree of load balancing, we use the following approach. Consider the process of constructing an explicit path as a sequence of choosing the appropriate nodes and links. When we are at a node at which we have more than one equivalent choices for the next hop, we favor choosing one that is reached by a higher capacity link from the current node. This is achieved by selecting the next node at random with a probability that is weighted by the available bandwidth of the link connecting the current node to the next hop.

2.4 Update Policies

An update policy determines when a network node triggers link state updates and the specific contents of such an update. We assume that the scope of a node's update extends to all its incident links, i.e., bandwidth values for all the interfaces of the node are advertised even when the update is triggered by just one link. This is consistent with the behavior of routing protocols such as Open Shortest Path First (OSPF) [15] that only generate link state updates that contain information about all the links attached to a router.

All policies attempt to trigger an update only when the current value of a link metric differs *significantly* from the previously advertised value. We investigate the following update trigger policies which differ in how they determine the significance of a change in the available bandwidth metric.

Threshold based updates: This policy is characterized by a constant threshold value (th). At some node, if bw_i^o is the last advertised value of available bandwidth for interface i and bw_i^c is the current value, an update is triggered when $\|(bw_i^o - bw_i^c)\|/bw_i^o > th$. This policy tends to provide more detailed information when operating in the low available bandwidth range and becomes progressively less accurate for larger values of available bandwidth. In the graphs and the text these policies will be referred to as T/th, with th replaced with the actual threshold value.

Equal class based updates: This policy is characterized by a constant B which is used to partition the available bandwidth operating region of a link into multiple equal size classes: $(0, B)$, $(B, 2B)$, $(2B, 3B)$, $\ldots$, etc. An update is triggered when the available bandwidth on an interface changes so that it belongs to a class that is different from the one to which it belonged at the time of the previous update. This policy has the same degree of accuracy for all ranges of available bandwidth and will be referred to as E/B.

Exponential class based updates: This policy is characterized by two constants B and f ($f > 1$) which are used to define unequal size classes: $(0, B)$, $(B, (f+1)B)$, $((f+1)B, (f^2+f+1)B)$, $\ldots$, etc. Unlike the previous policy, the class sizes grow geometrically by the factor f. Updates are triggered as before, i.e., when a class boundary is crossed. This policy has fewer and larger classes in the high available bandwidth operating region and more and smaller classes when available bandwidth is low. Consequently, it tends to provide a more detailed and accurate state description for the low bandwidth region. These policies will be referred to as X/B/f.

With class based policies, triggering an update each time the available bandwidth value crosses a class boundary has the undesirable effect of generating overly frequent and somewhat meaningless updates when the available bandwidth fluctuates around a class boundary. In order to dampen such oscillatory behavior, class based policies are augmented by a hysteresis mechanism. This mechanism suppresses generation of an update until the available bandwidth reduces sufficiently to fall below the middle value of the new class. No such rule is applied when the available bandwidth increases and crosses a class boundary.

We advertise quantized values for available bandwidth only in the class based policies. The value advertised corresponds to the bandwidth at the middle of the new available bandwidth class. For exponential classes we consider 2 and 4 as two possible values for f. The accuracy of class based policies is varied by changing the base (smallest) class size B. In our experiments, we use four different values for the base class size B: 25%, 50%, 100%, and 200% of the request bandwidth range. These values result in progressively less accurate information in the low available bandwidth region. For example, when equal classes are used, a class size of 25% results into 4 classes in the region of bandwidth that a single flow can request. On the other hand, a value of 200% in conjunction with advertising a quantized value that is the middle value of the class range, results in always advertising available bandwidth values that are larger than any individual flow's requested bandwidth. We call such policies *non-pruning* policies as they effectively cause the widest-shortest path selection algorithm to select only minimum hop paths with the available bandwidth information being used solely for load balancing purposes. Combinations of triggering policies and type of advertised value that can result in link pruning will be called *pruning* policies. Our classification of triggering policies into pruning and non-pruning assumes that the maximum value of bandwidth that may be requested is known so that the triggering policy can be tuned to never advertise smaller values for available bandwidth. This may not be possible in a realistic situation. As a result, triggering policies will exhibit a hybrid behavior with the distribution of the requested bandwidth values determining which of the two modes will be dominant. We add clamp-down timers to all of the above policies to further control the protocol overhead.

2.5 Higher Level Admission Control

It is well known [1, 2] that excessive alternate routing can actually reduce routing performance in conditions of high load, since traffic following alternate routes can interfere with minimum hop traffic competing for the same links. This effect was observed in our study for the `mesh` topology under uniform traffic. An example of its manifestation can be seen in Figure 2. The routing performance of on-demand routing using the threshold policy is shown for varying triggering threshold values; there were no clamp down timers. The routing performance appears to improve with larger triggering threshold values, i.e., decreasing accuracy of network state information. At the same time, simulation showed that with increasing threshold values, less alternate paths were used. As a result, for very small thresholds, which provide a very accurate network state information, the routing algorithm could locate and use many alternate paths, and actually reduce routing performance.

In order to address this problem, we investigate high level admission control policies similar to trunk reservation. Assuming that explicit routing is used, we propose a trunk reservation approach that may result in rejecting requests routed over alternate paths during the resource reservation

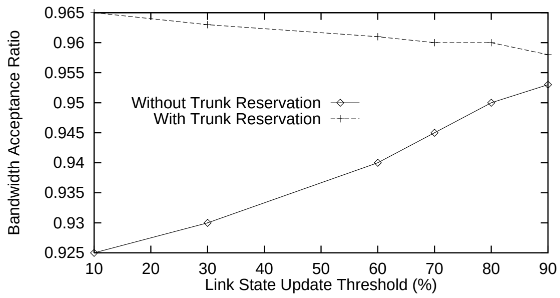


Figure 2: Effects of trunk reservation in the `mesh` topology (1 Mbits/sec, 3 min requests)

phase, even when there are sufficient resources to satisfy the request. A local per node check determines if the request is allowed to continue reserving resources over the path depending on both the resources that remain available on the link after the reservation and the relative length of the path, i.e., how much longer it is compared to the minimum hop path. Minimum hop path lengths can be easily computed in a separate step. When a reservation is attempted through a node, the quantity $(b_i^{avail} - b_{req})/b_i^{capacity}$ is calculated for the outgoing link i, where b_i^{avail} is the amount of available bandwidth on link i, and $b_i^{capacity}$ is the capacity of link i. The resource reservation for the request is allowed to continue only if this fraction is larger than a trunk reservation level which depends on the length of the path. If the request fails this test, it is rejected. Computing the trunk reservation level based on the request's requirements and the residual capacity of the link allows us to reject requests only when they really would have resulted in overloading a link. Having different trunk reservation levels for increasingly longer paths allows us to penalize longer paths more, and better control alternate routing.

The effectiveness of the above technique is demonstrated in Figure 2. When using high level admission routing performance is improved and there is no un-intuitive dependency on the threshold setting. For this experiment, the trunk reservation levels were set to 5% for one hop longer paths, 10% for two hops longer, and 20% for all paths more than two hops longer. All the results shown in the rest of this paper, were derived with this trunk reservation mechanism in effect using the above thresholds. In the `isp` topology, the effects of the above mechanism are less dramatic and can actually result in some routing performance loss due to the smaller number of alternate paths. Still, in all the experiments we use higher level admission control since we believe that similar mechanisms will have to be used in a real production network.

We must note that since non-pruning triggering policies never use alternate paths, there is no point in using high level admission control combined with a non-pruning triggering policy. Nevertheless, as was discussed earlier, in realistic situations, due to lack of knowledge of the range of bandwidth requests, it will be hard to implement a purely non-pruning policy; high level admission control will therefore still be needed in practice.

3 Protocol Overhead

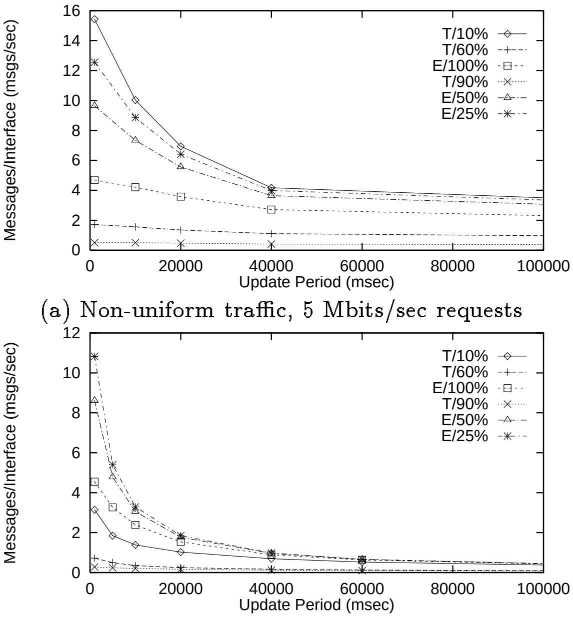
Protocol traffic is affected by three different factors: (a) the triggering policy used, (b) the sensitivity level of the particular triggering policy that is determined by its specific control parameter (threshold value th, base class size B, class growth factor f) and, (c) the use of a non-zero clamp down timer. In this section, we investigate and characterize the protocol overhead resulting from different combinations of the above factors.

All the results reported in this section are for the `isp` topology of Figure 1(a) under non-uniform traffic, with the widest-shortest path algorithm of [13] used to construct on-demand explicit paths at the origin node of each flow. We measure the protocol overhead by counting the number of network state update messages generated over a simulation run. Since we operate in a link state environment, each network state update results in multiple messages as it is flooded to all other nodes in the network; our count includes all these messages. Simulations were run until 100,000 requests were made, where the first 30,000 requests were used to warm up the network, and their effects are not included in the results. The duration of a simulation depends only on the request arrival rate. As a result, the total number of updates generated during a simulation run can be compared with other runs having the same request arrival rate. For all the values for the total update traffic volume reported, the 95% confidence intervals, computed using the t distribution, were under 4%.

In Figure 3(a) we show the number of protocol messages generated in a typical simulation run for some of the triggering policies for request sizes up to 5 Mbits/sec and mean flow duration of 3 minutes. While we consider both quantized metric advertisement and exact metric advertisement, we present results only for advertising exact values. Our experiments showed that the type of advertised value has negligible effect on the volume of update traffic. However, as we show later, the routing performance can depend significantly on the type of advertised value.

We observe that, for small or zero clamp down values, the choice of the triggering policy and its sensitivity level can result in large differences in the volume of update traffic. Increasing values for the clamp down timer results in reduced update traffic volumes for all combinations of triggering policies and sensitivity levels. For very large clamp down values (larger than those shown in the figure), the number of updates is primarily determined by the clamp down timer and all triggering policies and sensitivity levels produce similar amounts of routing traffic. In general, the amount of routing protocol traffic increases as the average request size increases. For comparison purposes, the OSPF link state routing protocol [15] enforces a minimum spacing of 5 seconds between successive floodings of the same LSA from a router and 1 second between processing arriving LSAs. This means that in general the message rate per interface will be bounded by 1 message/second (the actual value is hard to estimate since OSPF also acknowledges the reception of updates and can combine multiple updates into a single network packet). Some of the policies shown in Figure 3 are within these limits.

As can be expected, the number of classes affects the message overhead of class based policies as is illustrated in Figure 3 for the equal class policies. As can be seen in Figure 3(a), some combinations of triggering policy and clamp down values (such as the threshold policy with small threshold values and the equal class policy with a base class of 25% or 50% of the maximum request size) result in significantly larger numbers of messages than most of the other combinations of clamp down and triggering policy that tend to produce less messages. Even less routing messages are produced when the value of the clamp down timer is further increased. We expect the above behavior to be insensitive to the traffic characteristics. This is indeed the case as can be seen in Figure 3(b) that presents the volume of protocol traffic for 1 Mbits/sec requests. In all cases, three different levels of update volumes (and related cost), that correspond to different combinations of the three factors affecting update traffic volume can be clearly distinguished: (a) the *High Cost* environment with zero clamp down and sensitive triggering policies, (b) the *Medium Cost* environment with a combination of zero or small clamp down values and policies with fewer classes or reduced sensitivity and, (c) the *Low Cost* environment with very large clamp down values. As discussed in the introduction, in the rest of this work we primarily investigate the characteristics and routing performance of the low cost environment, i.e., large clamp down timers.



(a) Non-uniform traffic, 5 Mbits/sec requests

(b) Non-uniform traffic, 1 Mbits/sec requests

Figure 3: Update traffic volume comparison

Summarizing, the study of the volume of update traffic generated by various combinations of triggering policy and clamp down showed that: a) the type of advertised value (quantized versus non-quantized) does not significantly affect the number of updates generated; b) for large clamp down values the amount of update traffic is largely independent of the triggering policy used.

4 Routing Performance Under Large Clamp Down Timers

The examples of the previous section illustrate, among other things, the impact of clamp down timer values on routing protocol overhead. In this section, we study the routing performance of different policies, primarily for large clamp down timer values. We use a simple *static* version of the

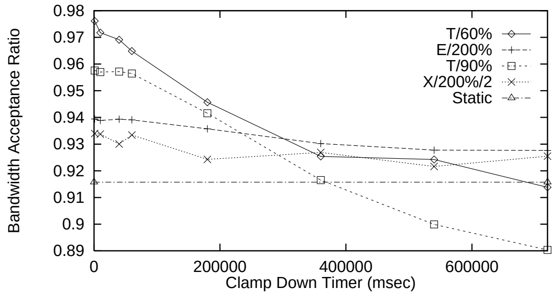
QoS routing algorithm as a *baseline* solution to compare the performance gains of other solutions. The baseline solution pre-computes a set of paths using the same widest-shortest path algorithm but does it only once and uses the *raw link capacity* information. Choosing among equal cost multi-paths at the time of path selection is done in the same way as the dynamic algorithm. Since this algorithm has no recurring path computation or protocol overhead costs, any dynamic QoS routing solution must perform significantly better than the static one in order to justify its increased cost. We are mainly interested in identifying combinations of triggering policies and advertised value types that can achieve good performance and low overhead and investigate how their routing performance compares to static routing.

We measure routing performance using the bandwidth acceptance ratio introduced in [7], defined as the percentage of the total requested bandwidth that was routed successfully. Note that this measure ignores the duration of calls and does not take into account the length of paths or other measures of efficiency such as bandwidth fragmentation or fairness. Still, it is a reasonable measure for studying the sensitivity of the *same* routing algorithm to varying network conditions. For all the results on bandwidth acceptance ratio reported here, the 95% confidence interval, as computed using the t distribution, was less than 0.5%. Although the trade-off between routing performance and volume of update traffic is probably the most appropriate criterion, in the following discussion, we compare only routing performance; the use of a large clamp down value results in a small number of network state updates that are relatively insensitive to the triggering policy used.

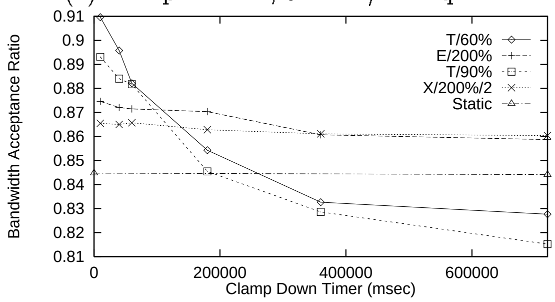
The results of our experiments show that the pruning or non-pruning nature of the triggering policy has important implications for the routing performance for both small and large clamp down timer values. Since non-pruning policies use only minimum hop paths for routing requests we would expect their routing performance to be worse than pruning policies that can also use alternate paths. On the other hand, pruning policies depend on network state information to determine these alternate paths and potential inaccuracies in this state can penalize their performance. Indeed, if paths are determined based on outdated information it is possible to end up using (potentially for extended periods of time due to large clamp down timer settings) inefficient alternate paths, wasting network resources. In contrast, the pruning policies use network state information only when choosing between multiple minimum hop paths to a destination and we would expect them to be less sensitive to network state inaccuracies. These effects can indeed be observed in Figure 4.

In this figure we show examples of the routing performance of both exponential and equal class based non-pruning policies compared to a threshold policy with a threshold value of 60% and 90% (both pruning policies), and the baseline algorithm for different types of traffic. In parts (a) and (b) of the figure all the non-pruning policies have rather low performance for small values of clamp down but for much larger clamp down values they do better than the threshold policies. The type of traffic (uniform versus hot-spot) does not seem to affect this behavior.

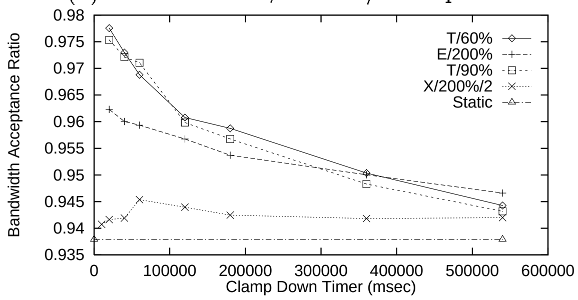
Advertising quantized values, apart from the non-pruning effect that it can achieve, if combined with the appropriate triggering policy, also facilitates the discovery of equal hop multipaths that can provide more alternatives for routing requests. Our experiments showed that even for base class size smaller that 200% (so that link pruning will be performed)



(a) Hot-spot traffic, 5 Mbits/sec requests



(b) Uniform traffic, 5 Mbits/sec requests



(c) Hot-spot traffic, 1 Mbits/sec requests

Figure 4: Routing performance of non-pruning policies

advertising a quantized value improves routing performance.

In Figure 4, we can see that, although non-pruning policies operate in a fashion quite similar to the static algorithm, even very sparse network state updates can result in reasonable routing performance improvement over static. It must be understood that the relationship between the routing performance between static routing, pruning, and non-pruning policies depends on a variety of factors such as topology, clamp down value, arrival rate as well as size and duration of the requests. Request size and arrival rate determine how rapidly the link load fluctuates, while the clamp down timer determines how often these changes can be communicated to the routing algorithm. Although it may be possible to model these relationships, it is a very complex task if it is to be performed for arbitrary topologies.

An example of the effects of the request size for the `isp` topology is shown in Figure 4. In part (c) of the figure, we can see that for small request sizes, even for large clamp down values, all policies can do better than static routing. Pruning policies perform consistently better than the non-pruning ones and do not become worse even for large clamp down timer values. As discussed earlier, since the requests have very small size, the effects of the bad routing decisions due to imprecise link state information are reduced.

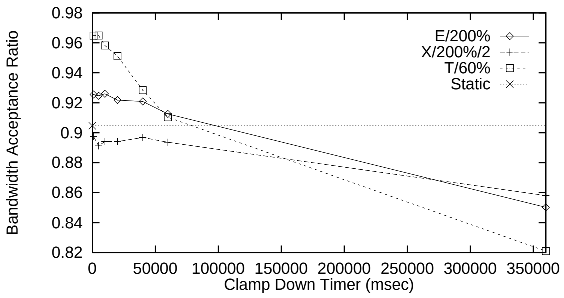


Figure 5: Routing performance of non-pruning policies in the `mesh` topology

Network topology can also have a significant effect on the relative performance of different routing solutions. As can be seen in Figure 5, in the `mesh` topology, the routing performance of non-pruning policies can become worse than static for larger clamp down periods. In this topology there is a large number of minimum hop multipaths between the sources and the destinations of hot-spot traffic. This gives an advantage to static routing that is able to use all of these paths while both pruning and non-pruning policies will always attempt to only use the widest of the multipaths. Even coarsening of the advertised values of available bandwidth routing performance does not help much in this case.

Summarizing, in this section we showed that: a) the routing performance of pruning policies degrades significantly with increasing clamp down values and can become worse than static routing; b) the routing performance of non-pruning policies can be significantly better for large clamp down values; c) in some cases even very infrequent updates can maintain the routing performance of non-pruning policies at levels better than static routing; d) advertising quantized values improves routing performance since it allows the discovery of more equal hop multipaths; e) the request size has significant impact on the routing performance of various policies; for small requests the performance degradation of pruning policies is less while large requests penalize routing performance more.

5 Improving Routing Performance

So far, we have shown how routing performance for large clamp down timers can be improved by proper selection of a triggering policy and type of advertised value. Next we attempt to modify the routing algorithm itself to compensate for the inaccuracy due to the large clamp down timers. Since network state updates are generated very infrequently, on-demand computation is wasteful since most of the time the topology database is not updated between consecutive requests. For this reason, we depart from the on-demand operating model used so far and focus on pre-computation based solutions; paths to all destinations are pre-computed each time a network state update is received. A more detailed discussion of path pre-computation and its benefits will be presented in a later section.

Under extreme inaccuracy resulting from large clamp down values, the information used for path pre-computation and path selection can not be entirely trusted. In particular, during the path pre-computation phase paths may incorrectly be ignored since they appear to have smaller bottleneck capacity than other equal length or shorter paths. The path selection phase is also affected by stale advertised values; paths that were initially accurately computed, may become stale by the time they are selected for a given request.

We propose to cope with random inaccuracy by using the link state information only as a hint to the decisions of the routing algorithm. For example, during path selection, longer paths will be occasionally selected for a request although a shorter path may appear feasible. This should help avoid systematically selecting a path that due to inaccuracies may not be as good as it appears. Clearly, alternate paths must be used carefully. The length of the alternate path as well as the size of the requests should be considered, so that very inefficient use of network resources is avoided. In order to protect the network from such degradations we rely on high level admission control as described in Section 2.5 to limit the usage of alternate paths.

In the rest of the section we present extensions to both the path pre-computation and selection phases of the routing algorithm that attempt to reduce their sensitivity to network state inaccuracies. Although these extensions are not specific to a particular triggering policy, since, as shown in the previous sections, the non-pruning policies perform well when large clamp down values are used, we mainly use these policies in the evaluation of the proposed extensions.

5.1 Randomized Routing

First, we consider the implications of network state inaccuracy on the path pre-computation phase. Stale values of available bandwidth can result in erroneously ignoring some paths since they appear to have less available bandwidth than others. A low sensitivity triggering policy advertising highly quantized bandwidth values can help reduce this effect since there are few distinct bandwidth values advertised and there are more chances for ties between paths. In addition, the path pre-computation algorithm can be modified so that it makes decisions about which paths to maintain with a varying degree of independence from the advertised bandwidth values. A simple modification is to maintain alternate paths to destinations even if they have bottleneck bandwidth equal (and not only larger) than other shorter paths to this destination. In order to make sure that the path computation phase will terminate we impose a maximum hop length limit, for the topologies in this work this limit is set to 16 hops. In the results presented in the rest of this section, longer paths with a bottleneck capacity equal to that of shorter paths were maintained during path pre-computation.

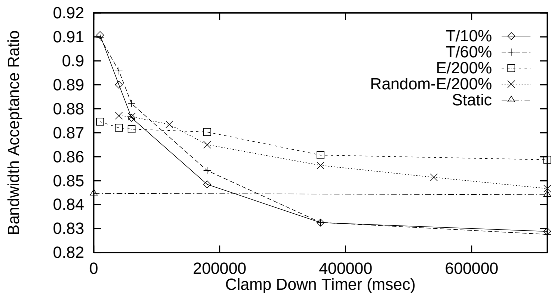
After paths are pre-computed, the path selection phase picks a path for routing a particular request. As discussed in Section 2.3, the path computation phase outputs for each destination d a list of paths $S_d =< S_d^1, S_d^2, \ldots, S_d^n >$. Since some of the S_d^i will be empty, let us assume that the list of non-empty path sets is $S_d =< S_d^{h_{min}}, S_d^{h_2}, \ldots, S_d^{h_{max}} >$. Each set $S_d^{h_i}$ contains paths of hop length h_i and common bottleneck capacity $B_d^{h_i}$. The standard algorithm deterministically selects the $S_d^{h_i}$ with the smallest h_i that has enough bandwidth to take the request. Instead, we propose to randomize the selection of $S_d^{h_i}$ by associating with each a weighted selection probability $P_d^{h_i}$.

One reasonable model for determining the selection probabilities is the following. The selection probability for a given hop count is set in proportion to a weight W_{h_i} associated with each $S_d^{h_i}$ so that $P_d^{h_i} = W_{h_i} / \sum_{h_k=h_{min}}^{h_{max}} W_{h_k}$. Each W_{h_i} clearly should depend on the hop length of the path (h_i) with respect to the minimum hop path that is feasible (according to the available link state information) for the call to be routed. Assume that the hop length of this path is H. The value of H is also affected by the inaccuracy of link state information and randomization should also consider paths with $h_i < H$ as candidates for routing a call. For paths with $h_i > H$ longer paths should be less probable for selection; W_{h_i} for these cases could be inversely proportional to hop length. Similarly, for paths $h_i < H$ it should be less desirable to choose a path with hop length much smaller than H; W_{h_i} should be proportional to h_i. We chose not to let W_{h_i} depend on the number of paths with hop length h_i. To avoid very large deviations from H we discount W_{h_i} by a factor $D_{h_i} = C^{|h_i - H|}$ with $0 \le C \le 1$. C is defined as $C = 1 - \theta * b_{req}/b_{max}$ with $0 \le \theta \le 1$ b_{req} the amount of bandwidth requested and b_{max} the maximum amount of bandwidth that can be requested; smaller requests will result in smaller values of C and smaller discounts for longer hop paths. This is desirable since the effects of misrouting small requests are less serious. The value of θ can be used to control how aggressive the use of longer (or shorter) alternate paths will be. A weight for each $S_d^{h_i}$ is computed as $W_{h_i} = B_d^{h_i} * D_{h_i}$ incorporating in this way the network state information into the value of the weight assigned to a particular $S_d^{h_i}$. For longer paths that have significantly larger available bandwidth, the large value of $B_d^{h_i}$ will offset the large discount and result in a large weight and increased probability of selection.

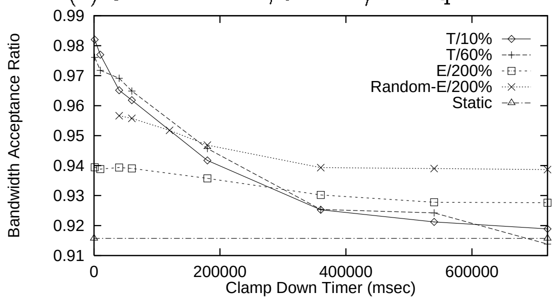
The effectiveness of the above randomization approach clearly depends on the underlying network topology. If there is high amount of link sharing among primary and alternate paths the randomized algorithm will not be able to increase performance very much. Although it will be using a larger variety of paths, it will still end up using and loading the same few bottleneck links that lead to a destination, failing to achieve improved routing performance over non-randomized routing. In addition, the traffic characteristics also affect the effectiveness of randomized routing. Under uniform traffic, alternate paths will be highly loaded and randomizing the routing of requests over them will not be particularly effective. If on the other hand, as a result of hot-spot traffic, there are alternate paths with lower load, randomization may be able to use these paths more effectively than non-randomized routing. We experimented with two different settings for the threshold θ; in the *conservative* case θ was chosen so that C varied between 0.7 and 0.3 depending on the size of the requests while in the *aggressive* case it was chosen so that C varied between 0.6 and 0.9.

The `isp` topology allows us to illustrate the above points in Figure 6. In this figure we compare, for both exponential and equal class based non-pruning policies, the routing performance of the randomized and *on-demand* routing using the same triggering policy. Under uniform traffic (Figure 6(a)) we do not observe any improvement in routing performance when the randomized version of the routing algorithm is used. Under hot-spot traffic though (Figure 6(b)), randomization was able to improve routing performance of the non-pruning policies for very large clamp down values, above that of other more accurate update generation policies both pruning and non-pruning. The setting of the weights did not have any significant effect in the routing performance in the `isp` topology.

To better investigate the dynamics of the proposed ran-



(a) Uniform traffic, 5 Mbits/sec requests



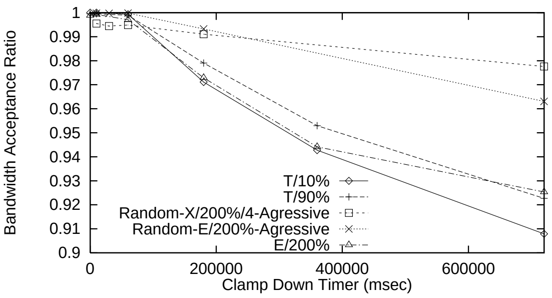
(b) Hot-spot traffic, 5 Mbits/sec requests

Figure 6: Performance of randomized routing in the `isp` topology

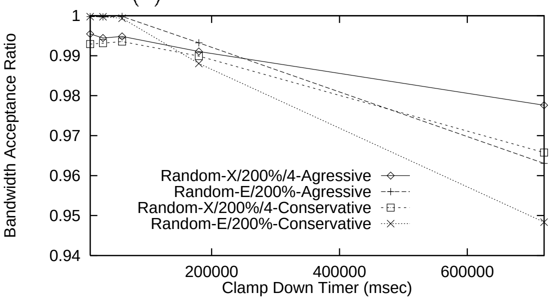
domized routing algorithm, we use the topology shown in Figure 1(c). This topology, unlike `isp`, contains several disjoint alternate paths that can potentially be used for alternate routing, and can give a better idea of the potential of the randomization approach. In this topology, the randomized algorithm improves routing performance substantially for large values of clamp down. This is shown in Figure 7(a), which compares the routing performance of randomization and on-demand routing using the same triggering policies. In Figure 7(b) we show the effects of the weight settings when an exponential and an equal class based non-pruning triggering policy is used. In the `disjoint multipath` topology the two alternate paths between the hot-spot source and the destination nodes are less loaded than the minimum hop paths. As a result, it is usually a good idea to be aggressive and prefer an alternate path more often. In this Figure, the triggering policy X/200%/2 performs better than policy E/200%. The fewer classes of policy X/200%/2 lead to better performance since there are more chances for ties between paths resulting in more paths maintained during path pre-computation.

In Figure 7(c) we evaluate the utility of extending the path pre-computation phase to maintain paths with bottleneck capacity equal to that of shorter paths. When only strictly longer alternate paths were maintained routing performance was reduced for all triggering policies and weight settings. In this particular topology, because of the triggering policies used and the quantized values being advertised, it is quite likely that longer alternate paths have capacity equal to that of the shorter direct path(s). Not keeping these longer alternate paths, at least for the nodes with increased traffic levels, appears to severely impact routing performance.

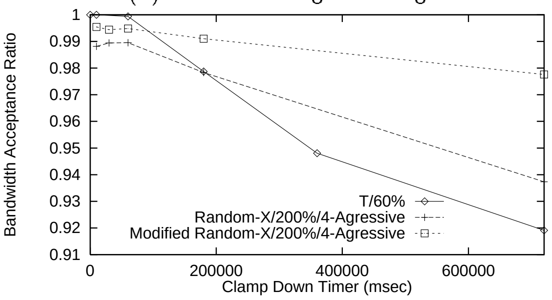
Randomized routing in the `mesh` topology did not achieve any important improvement in routing performance for large



(a) Effects of randomization



(b) Effects of weight settings



(c) Effects of modified path computation phase

Figure 7: Performance of randomized routing

clamp down values for both uniform and hot-spot traffic. This is because, unlike the `isp` topology, `mesh` exhibits high link sharing between minimum hop and alternate paths. Even for the selection of hot-spot node pairs shown in Figure 1(b) both primary and alternate paths have to share the two incident links in at least one of the pair's nodes. This renders randomization ineffective since even alternate routed traffic will have to compete for resources on the few common links.

Summarizing, a) randomized routing when combined with a non-pruning policy is effective at improving routing performance for large clamp down values when the underlying topology has alternate paths that are relatively disjoint and less loaded than the minimum hop paths and/or traffic is non-uniform. Under such conditions, randomization can achieve significantly better routing performance than the non-randomized algorithm using any pruning and non-pruning policy; b) modifying the path pre-computation phase to maintain more paths is in most cases also advantageous.

6 Processing Cost

In this section, we investigate the processing cost of the different solutions presented in the previous sections. With the exception of the randomized routing algorithm, the presentation so far assumed that paths were computed in an on-demand fashion. This may result in large processing load, proportional to the number of the requests placed on the network. In some cases, as for example when large clamp down values are used, the contents of the topology database do not change very often, and performing a new path computation for each request can be wasteful since the same paths will be re-discovered. In such cases, pre-computing paths is a more reasonable alternative. Path pre-computation may be necessary even in cases where there is no large clamp down timer, if the processors in the routers can not cope with the processing load of on-demand path computation.

6.1 Measuring the Processing Cost

The operations performed by the routing algorithms discussed in this paper can be distinguished into several categories. In *Initialization*, the QoS routing table is initialized. The *QoS routing table maintenance* operations perform updates to the QoS routing table by adding/deleting and replacing paths during the path computation phase. *Node Operations* access the topology database to retrieve information about a particular network element. *Per Iteration Operations* maintain data needed by the Bellman-Ford algorithm, mainly a queue of vertices to be expanded in the next iteration. Finally, *Path Selection* determines a path for a particular request using information in the QoS routing table.

In order to meaningfully compare the processing cost of the various routing alternatives considered in this paper, we estimate the cost of each of the operation types described above using specially designed benchmarks. The benchmarks perform a sequence of path computations in an empty network while the execution time of the various operations is monitored using the `pixie` tool, available in the Digital Unix platform used for the simulations. Then in later simulations, we maintain counts for each operation type, and the time spent in routing algorithm operations is computed for the whole network. The costs of the different operations are summarized in Table 1. The numbers are derived for the architecture we used for the simulations: Alphaserver 2100 4/274 servers with 4 Alpha 21064A CPUs at 275 MHz, 256 Mbytes of real memory, and Digital Unix operating system. The routing protocol code used is based on a random access implementation of the link state database, achieving therefore very efficient access to link state information. Path selection cost depends on the number of candidate paths but it is negligible and is not reported here. This is also the case for the randomized algorithm that performs more complex operations during path selection and usually handles more paths.

6.2 Reducing Processing Cost Through Path Pre-computation

There are two methods for deciding when to trigger a path pre-computation:

- *Pre-compute paths periodically:* In this case, a reasonable value for the pre-computation period is the clamp down timer value. Periodic path pre-computation does not necessarily need to be synchronous with the clamp down timer used but providing link state information

Operation	isp	mesh
Node (Avg/Node)	5	3
Iteration (Avg/Iteration)	2	1.5
Data Structure (Avg/Operation)	4	4
Initialization (Avg/Path Computation)	120	47

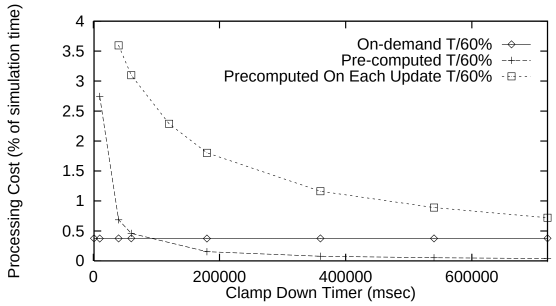
Table 1: Cost of the operations performed by the routing algorithms (in μsec)

more often than path pre-computation needs it is wasteful.

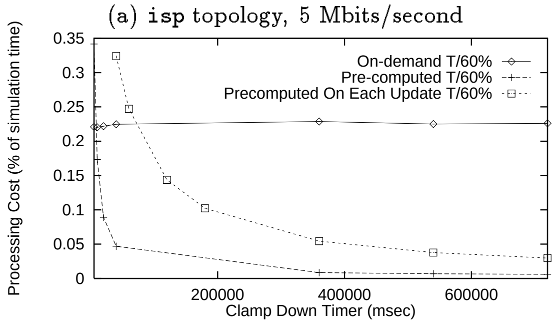
- *Pre-compute on each received update:* This can increase the number of path pre-computations over the previous approach by a factor of N, where N is the number of nodes in the network. Indeed, with a clamp down value C, over a period of time T there will be about T/C path computations per node if the first approach is used, for a total of NT/C path computations for the whole network. In the second approach there will be T/C updates per node that each will trigger NT/C path pre-computations in all the other nodes in the network, for a total of N^2T/C path pre-computations.

An example of the processing cost of these two alternatives is shown in Figure 8 for 5 Mbits/sec requests with duration of 3 minutes. The processing cost in this figure is amortized by dividing the total time spent in routing algorithm computations by the duration of the simulation. Periodic pre-computation can achieve very significant processing cost savings over on-demand computation. Triggering path pre-computation on every update received increases processing cost considerably. The relative cost of path pre-computation compared to on-demand path finding depends on the topology. In general, the larger and less connected the topology is, the larger the average cost of pre-computing paths to all destinations will be compared to the cost of computing a path to a single destination. This is why, the relative costs are very different for the `mesh` topology as can be seen in Figure 8(b). In this topology, due to its small size and high node degrees, computing paths to all destinations involves on the average only a small amount of extra operations over computing a path to a single destination.

While path pre-computation can achieve significant reduction of processing cost it also has important effects on routing performance. In experiments with periodic path pre-computation, the non-pruning policies achieve better routing performance than the other triggering policies for large clamp down values, similar to what was the case for on-demand path computation. Periodic path pre-computation results in some routing performance loss compared to on-demand path computation using the same triggering policy. An example of the relative performance of on-demand and periodic path pre-computation is shown in Figure 9. There is a similar routing performance loss for both pruning and non-pruning policies that increases with increasing clamp down timer values. This difference in routing performance is reduced for smaller request sizes. Similar differences between path pre-computation and on-demand computation can be observed for uniform traffic. The performance difference between periodic and on-demand path computation is eliminated when paths are pre-computed on every update received. In this case, on-demand and path pre-computation

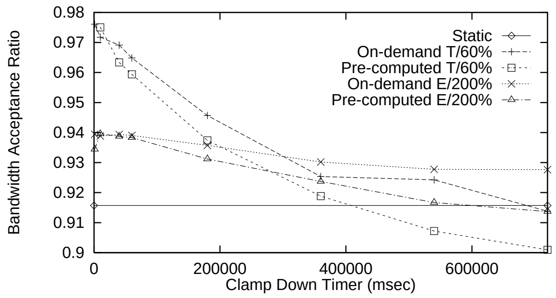


(a) `isp` topology, 5 Mbits/second



(b) `mesh` topology, 5 Mbits/second

Figure 8: Comparison of processing cost



Hot-spot traffic, 5 Mbits/sec

Figure 9: Routing performance of path pre-computation

compute their paths using exactly the same network state information,, and produce similar results with small differences due to load balancing.

Summarizing, a) periodic path pre-computation results in some routing performance loss compared to on-demand path computation; this performance loss disappears if path are pre-computed on each update received although this increases the processing cost considerably; b) periodic path pre-computation considerably reduced processing cost compared to on-demand path computation even for smaller clamp down values; pre-computing on each update is cheaper than on-demand only for large clamp down values; c) non-pruning update triggering policies still provide better routing performance for large clamp down values, even when paths are pre-computed.

7 Conclusions

In this work, we have investigated the cost-performance tradeoff in QoS routing through simulation. First, to cope with the negative effects of excessive alternate routing we developed a simple method for high level admission control and demonstrated its effectiveness in the `mesh` topology.

With a detailed comparison of the routing performance achieved from different combinations of triggering policies and parameter settings, we verified that triggering policies that do not result in pruning of links and advertise quantized values can perform better in conditions of high random inaccuracy, independently of the other details of triggering policies such as number or type of classes.

This behavior is similar for both on-demand and path pre-computation, although periodic path pre-computation results in a small routing performance loss over on-demand. The combination of path pre-computation and non-pruning triggering policies provides an efficient and very cost effective solution for routing when large clamp down timers are used and outperforms other more sensitive triggering policies that operate in either path pre-computation or on-demand mode.

For random inaccuracy resulting from very large clamp down settings, a randomization based path selection approach allows the routing algorithm to partially compensate for the large inaccuracy in the link state information. We demonstrated that this approach can increase routing performance significantly for larger clamp down values, mainly in topologies where there are multiple disjoint alternate paths and non-uniform traffic. The network state inaccuracy may also have negative effects on the path computation phase; we briefly discussed simple modifications that can improve the resiliency of path computation to infrequent updates of network state.

References

[1] R. S. Krupp, "Stabilization of Alternate Routing Networks," in IEEE International Communication Conference, Philadelphia, PA, 1982

[2] R.J. Gibbens, F. P. Kelly, and P. B. Key, "Dynamic Alternate Routing - Modeling and Behaviour," in Teletrafic Science for New Cost-Effective Systems, Networks and Services, ITC-12, Elsevier Science Publishers, 1989

[3] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A Framework for QoS-based Routing in the Internet," Internet Draft, QoS Routing Working Group, Internet Engineering Task Force, expires October 1998

[4] "Interim Inter-Switch Signalling Protocol Version 1," ATM Forum, *af-pnni-0055.000*, March 1996

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

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

[7] Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees," in proceedings of IEEE International Conference on Network Protocols, October 1997

[8] R. Guérin, and A. Orda, "QoS Based Routing in Networks With Inaccurate Information: Theory and Algorithms," in proceedings of INFOCOM, 1997

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

[10] 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

[11] 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

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

[13] R. Guérin, D. Williams, and A. Orda, "QoS Routing Mechanisms and QSPF extensions," in proceedings of GLOBECOM, 1997

[14] 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

[15] J. Moy, "OSPF Version 2," Request For Comments 2178, Internet Engineering Task Force, July 1997

[16] 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