QoS Routing Mechanisms and OSPF Extensions

Roch A. Guérin, Ariel Orda, and *Douglas Williams*

Abstract—**This paper presents and discusses path selection algorithms to support QoS routes in IP networks. The work is carried out in the context of extensions to the OSPF protocol, and the initial focus is on unicast flows, although some of the proposed extensions are also applicable to multicast flows. We first review the metrics required to support QoS, and then present and compare several path selection algorithms, which represent different trade-offs between accuracy and computational complexity. We also describe and discuss the associated link advertisement mechanisms, and investigate some options in balancing the requirements for accurate and timely information with the associated control overhead. The overall goal of this study is to identify a framework and possible approaches to allow deployment of QoS routing capabilities with the minimum possible impact to the existing routing infrastructure.**

Keywords—**Routing, QoS, OSPF, Bandwidth, Delay.**

I. Introduction

This paper deals with QoS routing, however rather than discussing its generic issues at large, it focuses on a concrete proposal, based on top of an existing routing protocol. Specifically, we describe a set of proposed additions to the OSPF routing protocol[1] to support Quality-of-Service (QoS) routing in IP. Focusing on unicast flows, we discuss the metrics required to support QoS, the associated link advertisement mechanisms, the path selection algorithm, as well as aspects of route establishment. Our goal is to define an approach which, while achieving the target of improving performance for QoS flows (likelihood to be routed on a path capable of providing the requested QoS), does so with the least possible impact on the existing OSPF protocol. Given the inherent complexity of QoS routing, achieving this goal obviously implies trading-off "optimality" for "simplicity", but we believe this to be required in order to facilitate deployment of QoS routing capabilities.

A. Why QoS-based Routing?

The current routing protocols used in IP networks are typically transparent to any particular Quality-of-Service (QoS) that different packets/flows may have. As a result, and although some protocols have "hooks," routing decisions are currently made without any awareness of resources availability and requirements. This means that flows are often routed over paths that are unable to support their requirements, while alternate paths with sufficient resources are available. This may result in significant deterioration in performance, i.e., in terms of call blocking probability. The goal of QoS routing is to provide routing algorithms that are capable of identifying such paths so as to satisfy the maximum possible number of flows with QoS requirements. In addition, such enhancements should be as synergetic as possible with existing routing protocols, so as to facilitate their introduction. Accordingly, QoS routing in the context of the Internet has been gaining increasing attention in the recent years e.g., [2], [3]. In parallel, an Internet resource reservation protocol, RSVP [4], has been proposed, that enables applications to request specific QoS guarantees across an Internet. However, RSVP has been careful to position itself as not being a routing protocol, and has left the issue of its interactions with QoS routing protocols for further research.

B. Overall Framework

We consider a network that supports both best-effort packets and packets with QoS guarantees. The way in which the network resources are split between the two classes is irrelevant to our proposal, except for the assumption that each QoS capable router in the network is able to dedicate some of its resources to satisfy the requirements of QoS packets. QoS capable routers are also assumed able to identify and advertise the amount of their resources that remain available for additional QoS flows. In addition, we limit ourselves to the case where all the routers involved support the QoS extensions described in this proposal, i.e., we do not consider the problem of establishing a route in an heterogeneous environment with routers that are QoS-capable and others that are not. Finally, as mentioned before, we focus on unicast flows, although many of the additions we define are applicable to multicast flows as well.

We assume that a flow with QoS requirements specifies them in some fashion accessible to the routing protocol. For example, this could correspond to the arrival of an RSVP *PATH* message, whose TSpec is passed to routing together with the destination address. After processing such a request, the routing protocol returns a path that it deems the most suitable given the flow's requirements. Depending on the scope of the path selection process, this returned path could range from simply identifying the best next hop, i.e., a traditional hop-by-hop routing, to specifying all intermediate nodes to the destination, i.e., a source route. Note that this decision impacts the operation of the path selection algorithm as it translates into different requirements to construct and return the appropriate path information. Once a suitable path has been identified, the flow is assigned to it (pinning) and remains assigned to it until it either releases the path (unpinning) or deems that it has become unsuitable.

The focus of this paper is the selection of an appropriate path based on link metrics information and flow requirements. Obviously, a complete proposal for QoS routing requires the specification of many other aspects, as mentioned above, and the reader is referred to [5], [6] for more details.

C. Simplifying Assumptions

In order to achieve our goal of minimum impact to the existing routing infrastructure, we impose certain restrictions on the range of requirements the QoS path selection algorithm needs to deal with directly. Specifically, the path selection algorithm

R. A. Guérin is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. guerin@watson.ibm.com.

A. Orda is with the Dept. Elec. Eng., Technion - I.I.T., Haifa, 32000 - ISRAEL. ariel@ee.technion.ac.il. Part of this work was done while he was visiting the IBM T.J. Watson Research Center.

D. Williams is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598. dougw@watson.ibm.com.

[1] The additions are built on top of OSPF V2 [1].

focuses on selecting a path that is capable of satisfying the bandwidth requirement of the flow, while at the same time trying to minimize the amount of network resources that need to be allocated to support the flow, i.e., minimize the number of hops used. This focus on bandwidth is adequate in most instances, but does not fully capture the complete range of potential QoS requirements. For example, a delay-sensitive flow of an interactive application could be put on a path using a satellite link, if that link provided a direct path and had plenty of unused bandwidth. This would clearly not be a desirable choice. Our approach to preventing such poor choices, is to assign delay-sensitive flows to a policy that eliminates from the network all links with high delay, e.g., satellite links, before invoking the path selection algorithm. A policy based approach can be extended to other criteria than delay, e.g., security, and in general, each existing policy would then present to the path selection algorithm a correspondingly pruned network topology, with the same algorithm used to generate an appropriate path. While a policy based approach has the benefit of simplicity, it is not always the most effective. In the paper, we also show how in the case of delay, a simple yet effective solution can be constructed when delay can be assumed to be of a particular form.

Another important aspect in minimizing the impact of QoS routing is to develop a solution that has the smallest possible computational overhead. Additional computations are unavoidable, but it is desirable to keep the total cost of QoS routing at a level comparable to that of traditional routing algorithms. In this paper, we describe several alternatives to the path selection algorithm, that represent different trade-offs between simplicity, accuracy, and computational cost. In particular, we specify algorithms that generate exact solutions based either on pre-computations or on-demand computations. We also describe algorithms that allow pre-computations at the cost of some loss in accuracy, but with possibly lower complexity or greater ease of implementation.

The rest of this paper is structured as follows. In Section II, we describe the path computation process. After describing the information that this process relies on, we present several path selection algorithms and discuss their complexity. The section concludes with an extension for handling delay requirements. In Section III, we consider the link state advertisement mechanism and discuss several related approaches. Concluding remarks appear in Section IV. Due to space limits, many technical details, as well as the discussion of source-routing *vs.* hop-by-hop alternatives, are omitted from this version, and they can be found in [5].

II. Path Selection Information and Algorithms

This section describes several path selection algorithms that can be used to generate QoS capable routes based on different trade-offs between accuracy, computational complexity, and ease of implementation. In addition, the section also covers aspects related to the type of information, i.e., metrics, on which the algorithms operate. The discussion focuses on the design principles, while specific, low-level, details are provided in [5].

A. Metrics

As stated earlier, the process of selecting a path that can satisfy the QoS requirements of a new flow relies on both the knowledge of the flow's requirements and characteristics, and information about the availability of resources in the network. In addition, for purposes of efficiency, it is also important for the algorithm to account for the total amount of resources consumed to support a new flow. In general, the network prefers to select the "cheapest" among all paths suitable for a new flow. Furthermore, the network may also decide to reject a new flow when the cost of its path is deemed too high. Such issues are more in the realm of call admission than path selection, but providing the ability to account for these aspects influences several metrics on which the path selection process is based. As a result, we consider the following metrics:

- *Link available bandwidth*: As mentioned earlier, we assume that most QoS requirements are derivable from a rate-related quantity, termed "bandwidth". Furthermore, since for a link to be capable of accepting a new flow with given bandwidth requirements, that much bandwidth must at least be still available on the link, the link metric we select is the (current) amount of available (i.e., unallocated) bandwidth.
- *Hop-count:* This quantity is used as a measure of the path cost to the network. A path with a smaller number of hops (that can support a requested connection) is preferable, since it consumes fewer network resources.
- *Policy:* Policies are used to prune from the network links that are incompatible, performance or characteristics wise, with the requirements of a flow. The use of policies to handle specific requirements allows considerable simplification in the optimization task to be performed by the path selection algorithm.

B. Path Selection Algorithms

There are several aspects to the path selection algorithms. The main ones include the optimization criteria it relies on, the exact topology on which it is run, and when it is invoked.

As mentioned before, invocation of the path selection algorithm can be either per flow, or when warranted by changes in link states when the algorithm used allows precomputation of paths (more on this below).

The topology on which the algorithm is run is, as with the standard OSPF path selection, a directed graph where vertices[2] consist of routers and networks (transit vertices) as well as stub networks (non-transit vertices). When computing a path, stub networks are added as a post-processing step similar to that of the standard OSPF algorithm (see [5] for details).

The optimization criteria used by the path selection are reflected in the costs associated with each interface in the topology and how those costs are accounted for in the algorithm itself. As mentioned before, the cost of a path is a function of both its hop count and the amount of available bandwidth. As a result, each interface has associated with it a metric, that corresponds to the amount of bandwidth which remains available on this interface. This metric is combined with hop count information to provide a cost value used in the path selection algorithm. How

[2] In this paper, we use the terms node and vertex interchangeably.

it is used depends on the exact form of the algorithm, but all the different alternatives described in the paper share a common goal. They all aim at picking a path with the minimum possible number of hops among those that can support the requested bandwidth. When several such paths are available, the preference is for the path whose available bandwidth (i.e., the smallest value on any of the links in the path) is maximal. The rationale for the above rule is the following: we focus on feasible paths (as accounted by the available bandwidth metric) that consume a minimal amount of network resources (as accounted by the hop-count metric); and the rule for selecting among these paths aims at balancing load as well as maximizing the likelihood that the required bandwidth will indeed be available.

It should be mentioned that some care needs to exercised to ensure a proper correspondence between edges in the topology database and hop counts. This is because, as mentioned earlier, networks also correspond to vertices in the topology. Hence they are associated with **edges** but should only be counted as **one** hop. This problem can be readily handled through the use of *zero-hop* edges in the algorithm. For simplicity, we limit ourselves in the paper to the case where edges and hop count coincide, i.e., point-to-point links connecting routers, and the reader is referred to [5] for details in handling zero-hop edges.

It should be noted that standard routing algorithms are typically single objective optimizations, i.e., they may minimize the hop-count, or maximize the path bandwidth, but not both. Double objective path optimization is a more complex task, and, in general, it is an intractable problem [7]. Nevertheless, as we will see, because of the specific nature of the two objectives being optimized, the complexity of our proposed algorithm(s) is similar to that of standard single-objective algorithms.

B.1 Algorithm for exact pre-computed QoS paths

In this section, we outline a path selection algorithm, that for a given network topology and link metrics, allows us to pre-compute all possible QoS paths, and also has a reasonably low computational complexity. Specifically, the algorithm allows us to pre-compute for any destination a minimum hop count path with maximum bandwidth, and has a computational complexity comparable to that of a standard shortest path algorithm. The path selection algorithm is based on a *Bellman-Ford* (BF) shortest path algorithm, which is adapted to compute paths of maximum available bandwidth for all hop counts. It is a property of the BF algorithm that, at its h-th iteration, it identifies the optimal (in our context: maximal bandwidth) path between the source and each destination, *among paths of at most h hops*. Therefore, we also take advantage of the fact that the BF algorithm progresses by increasing hop count, to essentially get for free the hop count of a path as a second optimization criteria.

Specifically, at the kth (hop count) iteration of the algorithm, the maximum bandwidth available to all destinations on a path of no more than k hops is recorded, together with the corresponding routing information. After the algorithm terminates, this information enables us to identify for all destinations and bandwidth requirements, the path with the smallest possible number of hops and sufficient bandwidth to accommodate the new request. Furthermore, this path is also the one with the maximal available bandwidth among all the feasible paths with this minimum number of hops. For further details, the reader is referred to [5].

B.2 Algorithm for on-demand computation of QoS paths

In the previous section, we described an algorithm that allows pre-computation of QoS routes. However, it may be feasible in some instances, e.g., limited number of requests for QoS routes, to instead perform such computations on-demand, i.e., upon receipt of a request for a QoS route. The benefit of such an approach is that depending on how often recomputation of pre-computed routes is triggered, on-demand route computation can yield better routes by using the most recent link metrics available. Another benefit of on-demand path computation is the associated storage saving, i.e., there is no need for a QoS routing table. This is the standard trade-off between memory and processing cycles. In this section, we briefly describe how a standard Dijkstra algorithm can, for a given destination and bandwidth requirement, generate a minimum hop path that can accommodate the required bandwidth and also has maximum bandwidth. Because the Dijkstra algorithm is already used in the current OSPF route computation, only differences from the standard algorithm are described.

The algorithm essentially performs a minimum hop path computation, on a graph from which all edges, whose available bandwidth is less than that requested by the flow triggering the computation, have been removed. This can be performed either through a pre-processing step, or while running the algorithm by checking the available bandwidth value for any edge that is being considered. Another modification to a standard Dijkstra based minimum hop count path computation, is that the list of equal cost next (previous) hops which is maintained as the algorithm proceeds, needs to be sorted according to available bandwidth. This is to allow selection of the minimum hop path with maximum available bandwidth. Alternatively, the algorithm could also be modified to, at each step, only keep among equal hop count paths the one with maximum available bandwidth. This essentially amounts to considering a cost that is function of both hop count and available bandwidth.

B.3 Algorithm for approximate pre-computed QoS paths

This section outlines a Dijkstra-based algorithm that allows pre-computation of QoS routes for all destinations and bandwidth values. As mentioned before, the benefit of using a Dijkstra-based algorithm is a greater synergy with existing OSPF implementations, while allowing pre-computation of paths helps lower computational overhead. The cost is, however, a loss in the "accuracy" of the pre-computed paths, i.e., the paths being generated may be of a larger hop count than needed. This loss in accuracy comes from the need to rely on *quantized* bandwidth values, that are used when computing a minimum hop count path. In other words, the range of possible bandwidth values that can be requested by a new flow is now mapped into a fixed number of quantized values, and minimum hop count paths are generated for each quantized value. For example, one could assume that bandwidth values are quantized as *low*, *medium*, and *high*, and minimum hop count paths are computed for each of these three values. A new flow is then assigned to the minimum hop path that can carry the smallest quantized value, larger than or equal to what it requested.

The algorithm operates again on a directed graph where vertices correspond to routers and transit networks. The metric associated with each edge in the graph is as before the bandwidth available on the corresponding interface, where $b_{n,m}$ is the available bandwidth on the edge between vertices n and m. The vertex corresponding to the router where the algorithm is being run is selected as the source node for the purpose of path selection, and the algorithm proceeds to compute paths to all other nodes (destinations). Starting with the highest quantization index, Q, the algorithm considers the indices consecutively, in decreasing order. For each index q, the algorithm deletes from the original network topology all links (n, m) for which $b_{n,m} < bw[q]$, where $bw[q]$ is the qth quantized bandwidth value, and then runs on the remaining topology a Dijkstra-based minimum hop count algorithm [3] between the source node and all other nodes (vertices) in the graph. Note that as with the Dijkstra used for on-demand path computation, the elimination of links such that $b_{n,m} < bw[q]$ could also be performed while running the algorithm. For further details, the reader is referred to [5].

C. Computational Complexity

One generic aspect of the algorithmic complexity of computing QoS paths is the efficiency of the shortest path algorithm used. Specifically, in this paper, we have described approaches based on both Bellman-Ford and Dijkstra shortest paths algorithms. Dijkstra's algorithm has traditionally been considered more efficient for standard shortest path computations because of its lower worst case complexity. However, in this section we briefly review a number of considerations, in particular in the context of multi-criteria QoS paths, which indicate that a BF approach may often provide a lower complexity solution.

The asymptotic worst-case complexity of a binary-heap implementation of the Dijkstra algorithm is $O(M \log N)$, where N is the number of vertices in the graph, and M the number of edges.[4]. The asymptotic worst-case bound for the BF algorithm is $O(H \cdot M)$, where M is again the number of edges in the graph, and H, which is the maximum number of iterations of the algorithm, is an upper-bound on the number of hops in a shortest path. Although, theoretically, H can be as large as $N - 1$, in practice it is usually considerably smaller than N. Moreover, in some network scenarios an upper-bound $\widehat{H}$ of small size (i.e., $\widehat{H} << N$) is imposed on the allowed number of hops, e.g., as part of a call admission scheme, in which case the complexity reduces to $O(\widehat{H} \cdot M)$, i.e., effectively to $O(M)$. As a consequence, as noted in [9] and experienced in [10], in practical networking scenarios, the BF algorithm offers an efficient solution to the shortest path problem, one that often outperforms the Dijkstra algorithm.

In the context of QoS path selection, the potential benefits of the BF algorithm are even stronger. As mentioned before, efficient selection of a suitable path for flows with QoS requirements cannot usually be handled using a single-objective optimization criterion. While multi-objective path selection is known to be an intractable problem [7], the BF algorithm allows us to handle a second objective, namely the hop-count, which is reflective of network resources, at no additional cost in terms of complexity. On the other hand, the Dijkstra algorithm requires some modifications (or approximations, e.g., bandwidth quantization) in order to be able to deal with hop count as a second objective. For example, if QoS paths are precomputed using a Dijkstra algorithm with Q quantized bandwidth values, the corresponding worst case asymptotic complexity is $O(Q \cdot (M \log N))$, which is comparable to the complexity of the BF algorithm[5], but at the expense of sub-optimal solutions. We mention, though, that when on-demand computations of QoS paths are practical, then a standard Dijkstra algorithm provides a (exact) solution of complexity $O(M \log N)$.

D. Extension: Handling Delay Requirements

In general, the framework proposed for path selection does not allow us to explicitly account for link propagation and queueing delays. As mentioned, this aspect is dealt with through a policy mechanism. However, it is worth pointing out that a simple extension to the proposed path selection algorithm allows us to directly account for delay if we can assume that the end-to-end delay can be expressed in a particular form. Specifically, assuming schedulers consistent with those of [11], end-to-end delay can be represented by the following expression:

$$D(\mathbf{p}) = \frac{\alpha(h(\mathbf{p}))}{b} + \sum_{l \in \mathbf{p}} d_l \tag{1}$$

where $\mathbf{p}$ is the path traversed, $D(\mathbf{p})$ is the guaranteed (upper-bound) end to end delay, $h(\mathbf{p})$ is the number of hops, b is the reserved bandwidth, d_l is the (fixed) propagation delay of a link l, and $\alpha(h)$ is a parameter that grows with h (a typical value is $\alpha(h) = \sigma + h \cdot c$, where σ is the burst size and c is the maximum packet size).

Assuming further that we restrict ourselves to intra-domain routing, and since links with prohibitively high propagation delays can be filtered out by policing, it can be assumed that the propagation delays d_l of all links can be reasonably upper-bounded by a single value d. Expression (1) then implies that an end to end delay requirement D can be translated into a bandwidth requirement $b(h)$ through the following expression:

$$b(h) = \frac{\alpha(h)}{D - h \cdot d} \tag{2}$$

where h is the number of hops on the path established for the connection.

III. Advertisement of Link State Information

It is assumed that each router maintains an updated database of the network topology, including the current state (available bandwidth) of each link. The distribution of link state (metrics) information is based on extending OSPF mechanisms (see

[3] Note that a Breadth-First-Search (BFS) algorithm [8] could also be used. It has a lower complexity, but would not allow reuse of existing code in an OSPF implementation.

[4] Although the Fibonnaci-heap implementation has a lower complexity, $O(N \log N + M)$, in practice it usually incurs a larger number of steps, due to the large constants involved [8].

[5] The exact merits of each of the two algorithms depend on the respective values of H, Q and N.

[5]). However, in addition to how link state information is distributed, another important aspect is when such distribution is to take place.

Ideally, one would want routers to have the most current view of the bandwidth available on all links in the network, so that they can make the most accurate decision on which path to select. Unfortunately, this then calls for very frequent updates, e.g., close to every time the available bandwidth of a link changes, which is neither scalable nor practical. Alternatively, one may opt for a simple mechanism based on periodic updates, where the period of updates is determined based on a tolerable corresponding load on the network and the routers. The main disadvantage of such an approach is that major changes in the bandwidth available on a link could remain unknown for a full period and, therefore, result in many incorrect routing decisions.

A better alternative is to use a simple hybrid update mechanism, which attempts to reconcile accuracy of link state information with the need for the smallest possible overhead. Such a scheme can be concretized and refined in various ways, some of which are discussed below.

Suppose that each node sends a Link State Advertisement (LSA) only when the ratio between the current value bw of a link and the last reported value is above (or below) some threshold, say 2.[6]. This implies that, when a path with some b units of bandwidth is sought, links with advertised bandwidth values above $2 \cdot b$ are "safe bets", those with values below $\frac{b}{2}$ should be excluded, and all the rest may supply the required bandwidth with various degrees of certainty. This means that a third objective is added to our two standard objectives of bandwidth and hop count, namely *certainty*. Its incorporation in the path selection process can be handled with various degrees of complexity and sophistication, of which we outline a few.

(1) A probabilistic approach:

The bandwidth value of a link l is, for the decision maker, a random variable that takes values in $(\frac{b_l}{2}, 2 \cdot b_l)$, where b_l is the last advertised value. Making some assumptions on the probability distribution of these values, e.g. uniform distributions, one can compute for each bandwidth requirement b the success probability of a link l, say $p_l(b)$, and then run a BF algorithm on the metric $\{w_l\}$, where $w_l = -\log(p_l(b))$ (see [3] for details). However, the problem here is that a different path should be computed *for each* bandwidth value b, hence rendering this approach too complex in the case of pre-computed routes. We are thus behooved to consider a simpler approach.

(2) A simple approach:

Here we run the standard BF algorithm, described in Section II-B.1, obtaining as an output an $N \cdot H$ QoS routing table. Let α, $0.5 \leq \alpha \leq 1$, be a parameter that indicates the "risk proneness" of the decision maker (the lower the value, the higher the risk proneness is). Also, let H_R be a parameter that indicates how many hops the decision maker is willing to trade for safety. Then, upon accommodating a connection request with b values of bandwidth, perform the following:

1. From the routing table, get $h_{\min}$, the minimal number of hops of a path with bandwidth of at least $\alpha \cdot b$ units.
2. From the routing table, get $h_{\max}$, the minimal number of hops of a safe path, i.e., with bandwidth of at least $2 \cdot b$.
3. If $h_{\min} + H_R \geq h_{\max}$: choose the safe path.
4. Otherwise: from the table, choose the path that has the maximal bandwidth among those having at most $h_{\min} + H_R$ hops.

(3) Elaborating on Approach (2):

A major problem of the above simple approach (2), is that some, potentially efficient, paths are disregarded. As a simple example, consider the case of two paths between the source and the destination: the first with two links of 10 units of advertised bandwidth, the second with three links, two of them with 20 units of advertised bandwidth and the third with 10 units of advertised bandwidth. Suppose that a connection request with 10 units of bandwidth is being handled. According to the simple approach (2), we will not make a note of the second path, since it gives the same bandwidth as the first path but with more hops. However, the second path has less unsafe links, hence it might be wiser to choose it. The following approach addresses this issue.

Let $b^1 < b^2 < \ldots < b^K$, where $K \leq M$, be the distinct bandwidth values corresponding to the last advertisement for each link. (as before, M denotes the number of links). The following is performed for each b^i, $1 \leq i \leq K$:

1. Erase the "forbidden" links, i.e., those with advertised bandwidth of less than $0.5 \cdot b^i$.
2. Assign a "cost" of 0 to a "safe" link, i.e. one with advertised bandwidth of at least $2 \cdot b^i$, and a cost of 1 to all other links, i.e., those with advertised bandwidth in the range $0.5 \cdot b^i \ldots 2 \cdot b^i$.
3. Run a BF algorithm, using the above cost as the metric.

As a result, one gets a (huge) table, which identifies, for each ("interesting") bandwidth value and for each number of hops h, a path with no more than h hops that has a minimal number of "non-safe" (and maximal number of "safe") links with respect to that bandwidth value. Then, when accommodating a connection with a bandwidth requirement of b units, such that $b^{(i-1)} < b \leq b^i$ for some $1 \leq i \leq K$, the decision maker can identify the various options for trading between safety and hops for the bandwidth value of b^i, and choose the respective path according to its perception of the trade-off between these two criteria.

Again, the caveat of this approach is complexity, which, in terms of both the running time and the table size, is M times bigger than that of a standard BF algorithm. The following, last, approach aims at handling the problems of the simple approach (2), but without the complexity of (3).

(4) Reducing the complexity of Approach (3):

Denote by $b_{\max}$ the maximal value that a link bandwidth can attain. Since the advertised bandwidth values are exponentially growing, according to the chosen factor (of, say, 2), it makes sense to limit the allowed range of values that can be advertised to only some predetermined and exponentially growing sequence of "thresholds", say $1, 2, 4, \ldots$. For example, the rule might be that an LSA is issued only when a new threshold is crossed, whereupon the LSA would advertise the threshold that is closest to the current bandwidth value. This way, the ratio between the actual and advertised bandwidth values is indeed kept within the range $0.5 \ldots 2$. However, the number of different advertised values, K, becomes now $O(\log(b_{\max}))$, rather than $O(M)$ as in Approach (3). Accordingly, the time and

[6] To keep the discussion simple, we do not bother here about potential oscillations when the values become very small, an issue that can be addressed by switching to an *additive* rule for such values.

space complexity are reduced to $O(M \cdot H \cdot \log(b_{\max}))$, and $O(N \cdot H \cdot \log(b_{\max}))$.

IV. Conclusions

In this paper, we have presented a number of path selection algorithms aimed at allowing the introduction of QoS routing capabilities in IP networks. The algorithms provide different trade-offs between accuracy, complexity, and synergy with existing implementations. In particular, the work presented here has been carried out in the context of extensions to the OSPF protocol, therefore minimizing the impact to existing OSPF implementations has been an important goal. In order to contain the complexity of computing QoS paths, we chose to focus on a single QoS requirement, namely bandwidth; we did show, however, how our approach could be used to also account for delay requirements, assuming they could be expressed in a form consistent with recent Internet service specifications [11].

Details on how the algorithms are to be incorporated in a specific proposal to extend OSPF can be found in [5], as well as [6], which covers the issue of path management when RSVP [4] is used as the signaling protocol. There are several aspects of the proposal for which additional work is needed. Two of the more important are the mechanisms used to determine when to update link information, and the relationship to call admission. The discussion of Section III provides some initial alternatives for link advertisement, but more experience is needed to validate them in practice. Similarly, the use of hop count as an optimization criteria can assist in the call admission process, still more specific rules are needed. We are currently experimenting with the proposed protocol and measuring its performance. Initial results prove to be encouraging. For example, preliminary simulations indicate that a modest communication overhead suffices to typically keep the performance of precomputed QoS routes within 10 to 15% of the "ideal" performance, where routes are computed on-demand and with exact knowledge of all the link metrics.

Acknowledgments

We would like to thank the many people who have helped shape various aspects of this paper and the approaches it describes, either through discussions or explicit suggestions. In particular, we would like to acknowledge the help and inputs of John Moy, Sanjay Kamat, Dilip Kandlur, and Dimitrios Pendarakis.

References

[1] J. Moy, "OSPF Version 2 - RFC No. 1583," INTERNET-RFC, March 1994.

[2] E. Crawley, R. Nair, B. Rajagopalan, and Hal Sandick, "A Framework for QoS-based Routing in the Internet," INTERNET-DRAFT, March 1997, work in progress.

[3] R. Guérin and A. Orda, "QoS-Based Routing in Networks with Inaccurate Information: Theory and Algorithms," in *IEEE INFOCOM'97*, Kobe, Japan, April 1997, pp. 75–83.

[4] R. Braden (Ed.), L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reSerVation Protocol (RSVP) version 1, functional specification," INTERNET-DRAFT, June 1997, work in progress.

[5] R. Guérin, A. Orda, and D. Williams, "QoS Routing Mechanisms and OSPF Extensions," INTERNET-DRAFT, November 1996, work in progress (Extended version, co-authored with S. Kamat and T. Przygienda: March 1997.).

[6] R. Guérin, S. Kamat, and S. Herzog, "QoS Path Management with RSVP," INTERNET-DRAFT, March 1997, work in progress.

[7] M.R. Garey and D.S. Johnson, *Computers and Intractability*, Freeman, San Francisco, 1979.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[9] D. Bertsekas and R. G. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1992.

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

[11] S. Shenker, C. Partridge, and R. Guérin, "Specification of Guaranteed Quality of Service," INTERNET-DRAFT, February 1997, work in progress.