Best-Effort Quality-of-Service

Bradley R. Smith †
brad@soe.ucsc.edu
†Computer Engineering Department,
University of California, Santa Cruz
Santa Cruz, CA 95064, USA

J.J. Garcia-Luna-Aceves †*
jj@soe.ucsc.edu
*Palo Alto Research Center (PARC)
3333 Coyote Hill Road
Palo Alto, CA 94304, USA

Abstract—**We show that the fundamental problems in providing quality-of-service in the existing Internet architecture has been the assumption that a single, "best" path from source to destination is adequate for any communications requirements of the network. We present a new, best-effort architecture for providing quality-of-service in the Internet based on the use of the "best set of paths" to destinations. We show that this set of paths is well defined, can be efficiently computed, and present an approach to efficiently implement this new, Best-Effort Quality-of-Service architecture.**

I. Introduction

The two basic approaches to packet switching are virtual circuits and datagrams. Both schemes segment messages into limited-size packets, add control information to each packet to accomplish its switching, and rely on statistical multiplexing of the shared communication links. Virtual circuits emulate circuit-switching used in the early telephone network. The virtual-circuit model is connection-oriented in that communication occurs in three phases (path setup, data transfer, and path teardown), routing is done once per flow by the ingress node during path setup, and paths are implemented using label-swap forwarding such that all traffic for a given flow follows the same path through the network. In contrast, packet switching based on datagrams is a more drastic departure from the circuit-switching model. Datagram switching is connectionless in that there are no phases in the communication process, packets are transmitted when the source host is ready to transmit, routing is computed at every router in the network on an event-driven basis, and forwarding decision is made on a hop-by-hop basis as packets flow through the network with the result that different packets in a given flow may follow different paths through the network.

The datagram approach to packet switching has a number of strengths. It is robust in the sense that it co-locates the routing process with the state it computes, manifesting a design principle called *fate-sharing* first described by Clark [1]. This ensures that the failure of any single component of an internet does not invalidate state located elsewhere in the internet, effectively localizing the affects of any failures. The datagram model is efficient and responsive for a couple of reasons. First, by implementing distributed control of forwarding state it requires only *simplex* communication of topology change events. Second, by assuming a distributed, hop-by-hop routing model, the datagram model enables the use of more efficient and responsive routing algorithms that can operate with partial information regarding the topology of the network.

Virtual-circuit switching is based on a centralized routing model, in that routes are computed on-demand, and forwarding is source-specified through the use of path setup techniques [2]. Hence, virtual circuits are less robust than datagrams due to the requirement that the ingress router control remote forwarding state in routers along the paths it has set up. The virtual-circuit model is less efficient and responsive for a couple of reasons. First, by implementing centralized control of forwarding state it requires *duplex* communication of topology change events: outbound notification of a topology event, and inbound notification of forwarding state changes. Second, by assuming a centralized routing computation the virtual-circuit model requires the use of full-topology routing algorithms to ensure every router can compute optimal paths to any destination in an internet

The architecture of today's Internet is based on the *catenet model of internetworking* [3]. In the catenet model, networks are built by the concatenation of disparate networks through the use of routers. The primary goals of the catenet model, and therefore the Internet architecture, were to support packet-switched communication between computers over internets composed of networks based on diverse network technologies, and to encourage the development and integration of new networking technologies into these internets.

To achieve these goals, a simple but powerful variant of the datagram communication model was adopted. Specifically, the Internet routing architecture is based on a *best effort* communication model in which the "best" path is pre-computed by each router to all destinations (triggered by topology changes), and packets are forwarded on a best effort basis (and may be dropped or delivered out of order in the event of congestion or routing changes). Packet forwarding is implemented on a hop-by-hop basis using destination-address based packet forwarding state computed by the routing process.

This best-effort, distributed, hop-by-hop, datagram routing

[1]This work was partially sponsored by the U.S. Army Research Office under grant W911NF-05-1-0246, by the National Science Foundation under grant CNS-0435522, by DARPA through Air Force Research Laboratory (AFRL) Contract FA8750-07-C-0169, and by the Baskin Chair of Computer Engineering. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

model has proven surprisingly powerful. Indeed, much of the success of the Internet architecture can be attributed to its routing model. However, largely as a product of its own success, limitations of this model are being encountered as it is applied to more demanding applications [4].

A significant limitation is the model only supports a single path to each destination. Specifically, Internet forwarding state is composed of a single entry for each destination in an internet giving the next-hop router on the path to the destination. As a result, only one path is supported to any given destination, and that path is computed to optimize a single metric.

As will be shown below, support for diverse performance, or *quality-of-service* (QoS), requirements of a network requires support of multiple paths to a given destination. Therefore the single-path limitation of the Internet translates to the inability to directly support applications with diverse QoS requirements. Clearly, such a model is not adequate for many of the demanding applications to which the Internet is currently being applied. The remainder of this paper presents a new solution for providing QoS services in the Internet that retains the Internet's datagram communication model while supporting multiple paths per destination. Section II gives a brief introduction to QoS communications requirements, and shows how they inherently require support for multiple paths per destination. Section III reviews previously proposed solutions for this problem in the Internet. Lastly, Section IV presents our solution to this problem, which is the first QoS routing solution that simultaneously supports the Internet's datagram communication model and the use of multiple paths to a destination. Section V presents a solution that efficiently supports forwarding traffic over multiple paths to a given destination. Section VI presents our conclusions.

II. Quality-of-Service Networking

A number of metrics can be used to quantify the performance of a communications network. For example *latency* is a measure of the delay traffic experiences as it traverses a network, *jitter* is a measure of the variation in that delay, *bandwidth* is a measure of the amount of data that can pass through a point in a network over time, etc.

Many applications have special requirements of the network they run on. For example interactive audio (i.e. VoIP) requires low latency and jitter of its communication channel to support natural, conversational interaction, however it has relatively minimal bandwidth requirements. In contrast, video streaming requires high bandwidth and low jitter to provide a smooth viewing experience, however it has relatively minimal latency requirements (it's OK if the video takes a number of seconds to start. as long as it runs smoothly once it starts). In further contrast, interactive video (i.e. video conferencing) is the most demanding in that it requires high bandwidth, low latency, and low jitter, combing the challenges of the previous two examples. The defining characteristic of these QoS requirements is they involve constraints on multiple performance metrics.

To satisfy constraints on multiple metrics requires, in general, the use of multiple paths between any two nodes in a network. For example, given two paths between two nodes with the following parameters: path 1 has bandwidth of 100Kbps, latency of 20ms, and low jitter; path 2 has bandwidth of 2Mbps, latency of 200ms, and low jitter, which is the better path? Path 1 would be preferred for an interactive audio application while path 2 would be preferred for video streaming. With multiple metrics, the preferred path depends on the requirements of the application.

It is true, in the example described here, that a path that provides the QoS required by interactive video would satisfy all applications. However, the availability of such premium paths can't be depended on, and the use of such paths, when they exist, for less demanding applications is, in general, a waste of valuable communication resources.

This correspondence between multiple metrics and multiple paths can be described formally by representing the set of metrics used to describe the performance of paths from a given source and destination pair as points in a multidimensional space. We'll call such a set of multiple metrics a link or path *weight*. Figure 1 plots the weights of 9 paths between a specific source and destination in an example network where the metrics composing the weights are bottleneck bandwidth and latency. "Better" values of these metrics are towards the origin of the graph (i.e. a perfect path would have infinite bandwidth and 0 latency).

These points can be interpreted as representing a region, up and to the right (away from the origin) of QoS values that each weight *satisfies* in the sense that the path represented by the weight would satisfy any QoS requirement in that region of the graph. Figure 2 depicts the regions satisfied by each path. Note that regions satisfied by some of the paths are fully contained in the regions of other paths. In the figure these *covered* regions are represented with dashed lines.

A *best set* of paths to the destination can be identified as the set of paths that are not covered by another path. This set of paths is *best* in the sense that any QoS requirement that is satisfiable by an existing path between the given source and destination, is satisfiable by a path in this set. We call these regions the *performance classes* available from the network for the destination. Figure 3 shows the performance classes for the example network.

The goal of QoS routing is to compute paths in a network that satisfy the performance requirements, expressed in terms of constraints on multiple metrics, of applications communicating across the network. The formalism presented above shows that, by definition, QoS routing must support the use of multiple paths between a source and destination.

III. Quality-of-Service in the Internet

As described in Sections I and II, the Internet architecture only supports a single path between a given source and destination, and support of QoS requires the use of multiple paths per destination. Therefore, the Internet architecture is inherently not able to directly support QoS. Two enhancements to the Internet architecture to support QoS have been proposed representing fundamentally different approaches to solving the

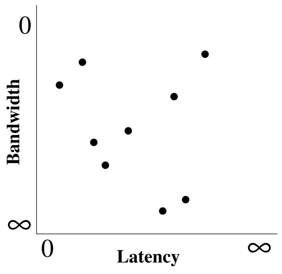


Fig. 1. Path Weights

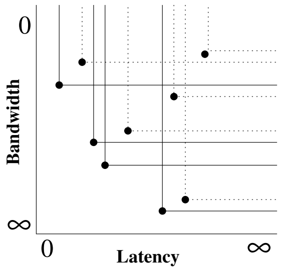


Fig. 2. QoS Regions

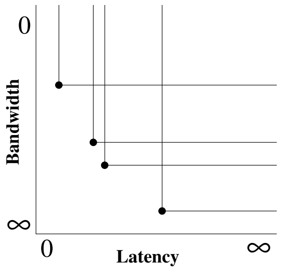


Fig. 3. QoS Classes

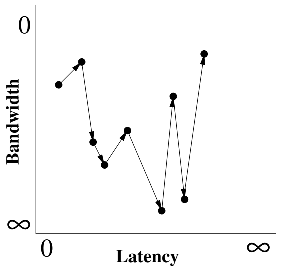


Fig. 4. Total Ordering

problem of resource management in the context of performance requirements, the Intserv and Diffserv architectures.

The goal of the *integrated services* (Intserv) architecture [4] is to define an integrated Internet service model that supports best-effort, real-time, and controlled link sharing requirements. Intserv makes the assumption that network resources must be explicitly controlled, and defines an architecture where applications reserve the network resources required to implement their functionality, and an infrastructure of admission control, traffic classification, and traffic scheduling mechanisms which implement the reservations. In the Intserv architecture resource reservations are sent along paths computed by the existing routing infrastructure. As a result requests may be denied when resources do not exist along the current route when in fact paths exist that could satisfy the request. Intserv is based on a virtual-circuit communications model and, as such, has all the limitations of that model relating to robustness, efficiency, and responsiveness discussed in Section I.

In contrast, the *differentiated services* (Diffserv) architecture [5] provides resource management without the use of explicit reservations. In Diffserv, a small set of *per-hop forwarding behaviors* (PHBs) is defined within a Diffserv domain which provide resource management services appropriate to a class of application resource requirements. Traffic classifiers are deployed at the edge of a Diffserv domain that classify traffic for one of these PHBs. Inside a Diffserv domain, routing is performed using traditional hop-by-hop, address-based forwarding mechanisms.

Diffserv retains the best-effort, distributed, hop-by-hop, datagram routing model of the Internet, and therefore retains the robustness, efficiency, and responsiveness of the Internet discussed in Section I. However, similar to the Intserv model, communications resources to a given flow in a Diffserv environment are limited to those available along the paths computed by the existing routing infrastructure. As a result QoS requirements may not be satisfied when adequate resources are not available along the current route when in fact paths exist that could satisfy the requirements.

IV. Best-Effort QoS

So far we have made the case that no effective solutions are currently known for providing QoS services in the Internet. Section II showed that QoS routing, by definition, must support multiple paths to each destination. Intserv moves the Internet back to a less robust, efficient, and responsive virtual-circuit communication model, and is limited to the use of only one path to a given destination. Diffserv retains the best-effort, distributed, hop-by-hop, datagram communication model of the Internet; however, it is still restricted to the use of only one path to a given destination. The remainder of this paper presents a new solution for providing QoS services in the Internet that retains the Internet's datagram communication model while supporting multiple paths per destination. The primary insight motivating this new QoS model is that the limiting assumption of the Internet architecture is the sufficiency of *the best path* to each destination. Given this assumption, the Internet protocols implement a routing computation that finds a single, best route to each destination in an internet, and use address-based forwarding that only supports a single path between any source and destination. Our new QoS model presented here is based on the need to support *the best set of paths* to each destination. This new model requires the precise definition of this set, a path selection algorithm that efficiently computes this set, and a forwarding plane that efficiently supports assignment and forwarding of traffic over multiple paths per destination. We call this new QoS model *Best-Effort Quality-of-Service* routing.

A. Best Set of Paths

As described in Section II, *path weights* are composed of multi-component metrics that capture all important performance measures of a link such as delay, delay variance ("jitter"), available bandwidth, etc. The best set of paths to a destination is defined using an enhanced version of the path algebra defined by Sobrinho [6].

Formally, the path algebra $P = < \mathcal{W}, \oplus, \preceq, \sqsubseteq, \overline{0}, \overline{\infty} >$ is defined as a set of weights $\mathcal{W}$, with a binary operator $\oplus$, and two order relations, $\preceq$ and $\sqsubseteq$, defined on $\mathcal{W}$. There are two distinguished weights in $\mathcal{W}$, $\overline{0}$ and $\overline{\infty}$, representing the least and absorptive elements of $\mathcal{W}$, respectively. Operator $\oplus$ is the original path composition operator, and relation $\preceq$ is the original total ordering from [6], which is used to order the paths for traversal by the path selection algorithm. Operator $\oplus$ is used to compute path weights from link weights. The routing algorithm uses relation $\preceq$ to build the forwarding set, starting with the minimal element, and by the forwarding process to select the minimal element of the forwarding set whose parameters satisfy a given QoS request.

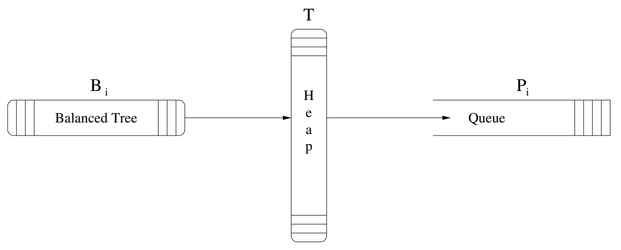


Fig. 5. Data structures for the QoS-Dijkstra Algorithm

P	$\equiv$	Queue of permanent routes to all nodes.
P_n	$\equiv$	Queue of permanent routes to node n.
T	$\equiv$	Heap of temporary routes.
T_n	$\equiv$	Entry in T for node n.
B_n	$\equiv$	Balanced tree of routes for node n.
$\mathcal{E}_n$	$\equiv$	Summary of traffic expression for all routes in P_n.

TABLE I
NOTATION.

Notation	Description
	Queue
$Push(r, Q)$	Insert record r at tail of queue Q ($O(1)$)
$Tail(Q)$	Return record at tail of queue Q ($O(1)$)
	d-Heap
$Insert(r, H)$	Insert record r in heap H ($O(\log_d(n))$)
$IncreaseKey(r, r_h)$	Replace record r_h in heap with record r having greater key value ($O(d\log_d(n))$)
$DecreaseKey(r, r_h)$	Replace record r_h in heap with record r having lesser key value ($O(\log_d(n))$)
$Min(H)$	Return record in heap H with smallest key value ($O(1)$)
$DeleteMin(H)$	Delete record in heap H with smallest key value ($O(d\log_d(n))$)
$Delete(r_h)$	Delete record r_h from heap ($O(d\log_d(n))$)
	Balanced Tree
$Insert(r, B)$	Insert record r in tree B ($O(\log(n))$)
$Min(B)$	Return record in tree B with smallest key value ($O(\log(n))$)
$DeleteMin(B)$	Delete record in tree B with smallest key value ($O(\log(n))$)

TABLE II
OPERATIONS ON DATA STRUCTURES [7].

A new relation on routes, $\sqsubseteq$, is added to the algebra and used to define classes of comparable routes and select maximal elements of these classes for inclusion in the set of forwarding entries for a given destination. Relation $\sqsubseteq$ is a partial ordering (reflexive, anti-symmetric, and transitive) with the following, additional property:

Property 1: $(\omega_x \sqsubseteq \omega_y) \Rightarrow (\omega_x \succeq \omega_y)$.

A route r_m is a *maximal element* of a set R of routes in a graph if the only element $r \in R$ where $r_m \sqsubseteq r$ is r_m itself. A set R_m of routes is a *maximal subset* of R if, for all $r \in R$ either $r \notin R_m$, or $r \in R_m$ and for all $s \in R - \{r\}$, $\neg(r \sqsubseteq s)$. The maximum size of a maximal subset of routes is the smallest range of the components of the weights (for the two component weights considered here). An example path algebra based on weights composed of delay and bottleneck bandwidth is as follows:

$$\begin{aligned}
\omega_i &\equiv (d_i, b_i)\\
\overline{0} &\equiv (0, \infty)\\
\overline{\infty} &\equiv (\infty, 0)\\
\omega_i \oplus \omega_j &\equiv (d_i + d_j, Min(b_i, b_j))\\
\omega_i \preceq \omega_j &\equiv (d_i < d_j) \vee ((d_i = d_j) \wedge (b_i \geq b_j))\\
\omega_i \sqsubseteq \omega_j &\equiv (d_j \leq d_i) \wedge (b_j \geq b_i)
\end{aligned}$$

Figure 4 is a graphical depiction of the relation $\preceq$ on the set of weights used as a example in Section II where $x \preceq y$ is depicted as $x \rightarrow y$. The $\sqsubseteq$ relation, illustrated by Figure 2, formalizes the *covers* notion presented above. And, lastly, R_m formalizes the notions of performance classes in a graph, and is the best set of routes we are looking for. R_m is illustrated in Figure 3.

B. Multi-constrained Path Selection

The notation used in the algorithms presented below is summarized in Table I. In addition, the maximum number of distinct performance classes is denoted by W, and the maximum number of adjacent neighbors by $a_{max} = \max\{|A(i)| \mid i \in N\}$. Table II defines the primitive operations for queues, heaps, and balanced trees used in the algorithm, and gives their time complexity used in the complexity analysis.

```
**algorithm** QoS-Dijkstra
     **begin**
1    $Push(<s, s, \overline{0}>,\ P_s)$;
2    **for each** $\{(s, j)\ \in\ A(s)\}$
3       $Insert(<j, s, \omega_{sj}>,\ T)$;
4    **while** $(|T| >\ 0)$
        **begin**
5       $<i, p_i, \omega_i> \leftarrow\ Min(T)$;
6       $DeleteMin(B_i)$;
7       **if** $(|B_i| =\ 0)$
8          **then** $DeleteMin(T)$
9          **else** $IncreaseKey(Min(B_i),\ T_i)$;
10      **if** $(\omega_i\ \not\sqsubseteq\ Tail(P_i).\omega)$
           **then begin**
11            $Push(<i, p_i, \omega_i>,\ P_i)$;
12            **for each** $\{(i, j)\ \in\ A(i)\ \mid\ \omega_i \oplus \omega_{ij}\ \not\sqsubseteq\ Tail(P_i).\omega\}$
                 **begin**
13               $\omega_j\ \leftarrow\ \omega_i\ \oplus\ \omega_{ij}$;
14               **if** $(T_j\ =\ \emptyset)$
15                  **then** $Insert(<j, i, \omega_j>,\ T)$
16               **else if** $(\omega_j\ \prec\ T_j.\omega)$
17                  **then** $DecreaseKey(<j, i, \omega_j>,\ T)$;
18               $Insert(<j, i, \omega_j>,\ B_j)$;
                 **end**
              **end**
        **end**
     **end**
```

Fig. 6. QoS Dijkstra.

The algorithm presented in this section is based on the data structure model shown in Figure 5. In this structure, a balanced tree (B_i) is maintained for each node in the graph to hold newly discovered, temporary labeled routes for that node. The heap T contains the lightest weight entry from each non-empty B_i (for a maximum of n entries). A queue, P_i, is maintained for each node which contains the set of permanently labeled routes discovered by the algorithm, in the order in which they are discovered (which will be in increasing weight).

The general flow of the algorithm is to take the minimum entry from the heap T, compare it with existing routes in the appropriate P_i, if it is incomparable with existing routes in P_i it is pushed onto P_i, and "relaxed" routes for its neighbors are added to the appropriate B_x's.

Figure 6 presents a modified Dijkstra SPF algorithm that computes the maximal set of routes to each destination subject to multiple metrics. The correctness of this algorithm is based on the maintenance of the following three invariants: for all routes $I \in P_*$ and $J \in B_*$, $I \preceq J$, all routes in a given P_i are incomparable, and the maximal subset of routes to a given destination i in $P_i \cup B_i$ represents the maximal subset of all paths to i using nodes with routes in P. Furthermore, these invariants are maintained by the following two constraints on actions performed in each iteration of these algorithms: (1) only known-non-maximal routes are deleted or discarded, and (2) only the smallest known-maximal route is moved to P. See [8] for a full proof of correctness.

The time complexity of the QoS-Dijkstra algorithm is dominated by the loops at lines 4 and 12. The loop at line 4 is executed at most once for each incomparable path (in terms of path weights) to each node in the graph for a total of nW times. The loop at line 12 is executed at most once for each distinct instance of an edge in the graph, for a total of mW times. The most time consuming operation performed as part of the loop at line 4 is the deletion from the balanced tree B_i at line 6 of the best temporarily labeled route with per-operation cost of $\log a_{max}W$, and an aggregate cost of $nW \log a_{max}W$. The accesses in lines 7–9 to the best route in heap T have a per-operation cost $\log_d n$, for an aggregate cost of $mW \log n$. For the loop at line 12, the most time consuming operation is the addition to the balanced tree B_i at line 18 with a per-operation cost of $\log a_{max}W$, and an aggregate cost of $mW \log a_{max}W$. Therefore, the worst case time complexity of QoS Dijkstra, dominated by the operation at line 18, is $O(mW \log W)$. Algorithms using enhanced data structures achieve time complexity of $O(mW \log(n))$ (see [8] for details).

C. Performance of QoS Dijkstra

Figures 7 and 8 present the results of experiments run using the enhanced version of the QoS-Dijkstra algorithm (not shown here). The experiments were run on a 2.2GHz Intel Core 2 Duo based system (Apple MacBook Pro) with 2GB of RAM. The algorithms were implemented using the C++ Standard Template Library (STL) and the Boost Graph Library. Each test involved running the algorithm on ten random weight assignments to ten randomly generated graphs (generated using the GT-ITM package [9]). For each test the worst case measurements are graphed. The metrics were generated using the "Cost 2" scheme from [10] where the delay component is randomly selected in the range $1..MaxMetric$, and the cost component is computed as $cost = \sigma(MaxMetric - delay)$, where σ is a random integer in the range $1..5$; this scheme

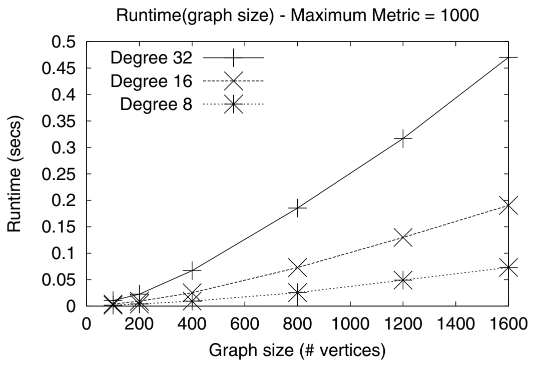


Fig. 7. Runtime vs. Vertex Count (Enhanced Algorithm)

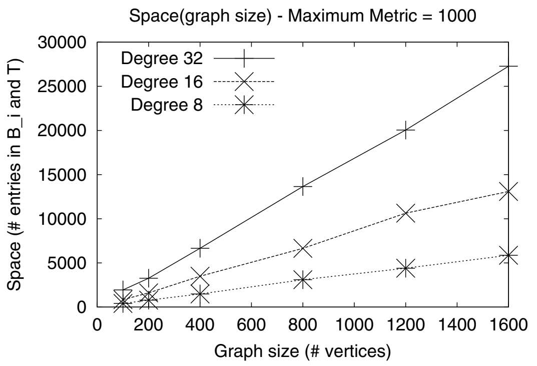


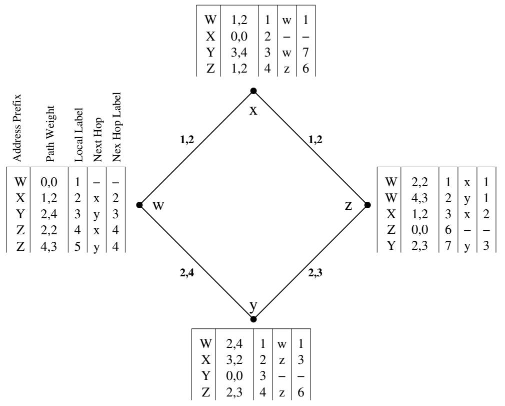
Fig. 8. Space vs. Vertex Count (Enhanced Algorithm)

was chosen as it proved to result in the most challenging computations from a number of different schemes considered. Space overhead was measured in terms of the maximum number of entries stored in the B_* structures.

Tests were run for performance (both runtime and space) as a function of graph size, average degree of the graph, and the maximum link metric value. Only the graphs for size are shown here. Also, since the maximum metric was shown to have little impact on performance, only results for tests with a maximum metric of 1000 are presented here. The results show very reasonable performance for graphs of up to 1600 nodes, well beyond the state-of-the-art in routing protocol scalability. These results indicate that these routing computations meet the performance requirements of modern routing domains with performance headroom to spare.

V. Distributed Label-Swap Forwarding

As illustrated in Figure 9, the forwarding table computed by QoS-Dijkstra contains an entry for each performance class. A performance class is defined by the weight of the path providing that performance class. Conceptually, forwarding involves determining the performance requirements for a packet based on traffic classification rules specified in terms of the contents of the packet, and selecting the path appropriate to these requirements.



Address Prefix	Path Weight	Local Label	Next Hop	Nex Hop Label
W	0,0	1	–	–
X	1,2	2	x	2
Y	2,4	3	y	3
Z	2,2	4	x	4
Z	4,3	5	y	4

W	1,2	1	w	1
X	0,0	2	–	–
Y	3,4	3	w	7
Z	1,2	4	z	6

W	2,2	1	x	1
W	4,3	2	y	1
X	1,2	3	x	2
Z	0,0	6	–	–
Y	2,3	7	y	3

W	2,4	1	w	1
X	3,2	2	z	3
Y	0,0	3	–	–
Z	2,3	4	z	6

Fig. 9. Forwarding Labels

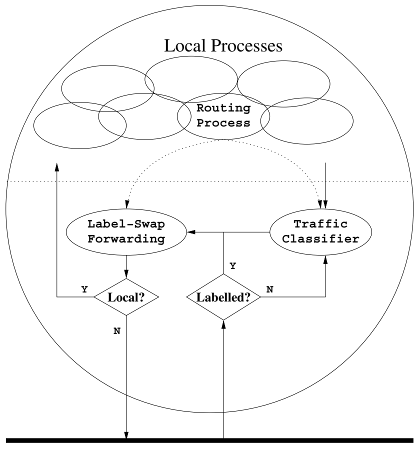


Fig. 10. Traffic flow in routers

Performing this traffic classification step at each hop in the network would be prohibitively expensive. To avoid this, we propose using label-swap forwarding to require only the first router that handles a packet to classify it before forwarding it. Accordingly, the forwarding state of a router must be enhanced to include local and next hop label information, in addition to the destination and next hop information existing in traditional forwarding tables. Traffic classifiers must then be placed at the edge of an internet, where "edge" is defined to be any point from which traffic can be injected into the internet. Figure 10 illustrates the resulting traffic flow requirements of a best-effort QoS router.

To date, label-swapping has been used in the context of connection-oriented (virtual circuit) packet forwarding architectures. A connection setup phase establishes the labels that routers should use to forward packets carrying such labels, and a label refers to an active source-destination connection [2]. Chandranmenon and Varghese [11] present *threaded indices*, in which neighboring routers share labels corresponding to indexes into their routing tables for routing-table entries for destinations, and such labels are included in packet headers to allow rapid forwarding-table lookups.

The forwarding labels in a best-effort QoS environment are similar to threaded indices. A label is assigned to each routing-table entry, and each routing-table entry corresponds to a policy-based route maintained for a given destination. Consequently, for each destination, a router exchanges one or multiple labels with its neighbors. Each label assigned to a destination corresponds to the set of service classes satisfied by the route identified by the label.

VI. Conclusion

In this paper we show that support of QoS requirements in general depends on the use of multiple paths per destination, and that the Internet architecture is limited to a single path service due to its use of shortest-path-first routing, and address-based forwarding. We then show that previous proposed solutions for the support of QoS in the Internet have significant limitations. Finally, we present a new, Best-Effort QoS routing solution, based on the use of the best set of paths to a destination, that implements the Internet's best effort, hop-by-hop, datagram communication model. This solution includes a formal definition of the "best set of paths" to a destination, an algorithm that efficiently computes this set of paths, and a forwarding architecture that efficiently implements forwarding over multiple paths to a destination. Specifically, routers independently compute the best set of routes available in an internet and forward packets using label-swap forwarding over paths appropriate to the application's performance requirements. The best set of paths is continually recomputed in response to topology events (link cost changes, or link or node failures). Performance parameters and application requirements are defined by network administration. This is the first QoS routing model that supports multiple paths per destination and retains the Internet's best-effort, hop-by-hop, datagram communication model, with the many benefits that come with it.

References

[1] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *Computer Communications Review*, vol. 18, no. 4, pp. 106–114, Aug. 1988.

[2] B. Davie and Y. Rekhter, *MPLS: Technology and Applications*. Morgan Kaufmann, 2000.

[3] V. G. Cerf and R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications*, vol. COM-22, no. 5, pp. 637–648, May 1974.

[4] B. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: an Overview," RFC1633, Jul. 1994.

[5] S. Blake, D. L. Black, M. A. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, December 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2475.txt

[6] J. L. Sobrinho, "Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 541–550, Aug. 2002.

[7] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows – Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[8] B. R. Smith, "Efficient policy-based routing in the internet," Ph.D. dissertation, University of California, Santa Cruz, 2003.

[9] E. W. Zegura, K. Calvert, and S. Bhattacharjee, "How to Model an Internetwork," in *Proceedings INFOCOM '96*. IEEE, 1996.

[10] S. Siachalou and L. Georgiadis, "Efficient QoS Routing," in *Proceedigns of INFOCOM'03*. IEEE, Apr. 2003.

[11] G. P. Chandranmenon and G. Varghese, "Trading Packet Headers for Packet Processing," *IEEE ACM Transactions on Networking*, vol. 4, no. 2, pp. 141–152, Oct. 1995, 1995.