Chapter 1

A NEW APPROACH TO POLICY-BASED ROUTING IN THE INTERNET

Bradley R. Smith
University of California, Santa Cruz
Corresponding Author
brad@soe.ucsc.edu

J.J. Garcia-Luna-Aceves
University of California, Santa Cruz
jj@soe.ucsc.edu

1. Introduction

The architecture of today's Internet is based on the *catenet model of internetworking* defined in Cerf, 1978; Cerf and Cain, 1983; Cerf and Kahn, 1974. 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 routing architecture was adopted.

The Internet routing architecture is based on a *best effort* communication model in which traffic is forwarded through an internet along paths that minimize a single, typically delay-related metric (which often is simply hop-count), and such that packets may be dropped or delivered out of order. These paths are constructed by a *distributed* routing computation where destination *address-based* packet forwarding state is computed autonomously by each router for a *single forwarding class* which provides minimum delay delivery.

There are a number of strengths of the Internet routing architecture. 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, 1988. 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 Internet routing architecture is *efficient* and *responsive* for a couple of reasons. By implementing distributed control of forwarding state it requires only *simplex* communication of topology change events. Specifically, since the routing process is co-located with the forwarding state it controls, a router only requires one-way (simplex) notification of the event from a remote router local to the event that detects it. By assuming a distributed, hop-by-hop routing model, the Internet routing architecture enables the use of more efficient and responsive routing algorithms that can operate with partial information regarding the topology of the network.

This best-effort, distributed, hop-by-hop routing architecture 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 (see Braden et al., 1994). The primary limitation of this routing model is that it only supports a single path between any given source and destination. Specifically, Internet forwarding state is composed of a single entry for each destination. Each entry is composed of the next-hop router on the chosen path to the destination. As a result, the Internet routing architecture only supports one path for any given destination, and that path is computed to optimize a single metric, typically delay or hop-count. This model has been extended to multi-path, equal-cost routing, which improves robustness but retains the limitations of only supporting a single performance class. Therefore, assumptions of uniform network performance requirements and network usage policies have been "hard-coded" into the Internet architecture. Specifically, the Internet routing architecture assumes all applications using an internet require minimization of the same performance parameter, and traffic from these applications may traverse any link in the internet to reach their destination. Clearly, such a model is not adequate for many of the demanding applications to which the Internet is currently being applied.

It is easy to find examples of diverse network performance requirements in the Internet today. While the minimum delay paths used in the best-effort communications model are well suited for the data services (e.g. e-mail, telnet, http, etc.) prevalent in the early Internet, they

are inadequate for new applications of Internet technologies. For example, the on-demand delivery of isochronous streams of data (i.e. data requiring delivery within specific time constraints, such as video and audio) requires low delay variance (called jitter), while the interactive delivery of isochronous data (e.g. Internet telephony) requires both low delay and low delay variance. Similarly, the delivery of streaming video requires high bandwidth, and is relatively loss-tolerant, while streaming audio requires relatively low bandwidth, but is loss-intolerant. Due to the single-class forwarding model used in the Internet architecture, only one of such a set of diverse service models can be effectively supported in an internet today. While some service models satisfy the requirements of others (e.g. a high-bandwidth, low delay, and low delay variance model can satisfy the requirements of both video and audio conferencing) this approach does not utilize the network resources as effectively as a set of custom service models.

Similarly, it is easy to find examples of network resource usage polices. The inability to provide differentiated services has become a stumbling block to realizing the commercial potential of Internet technologies. Commercial Internet Services Providers (ISPs) would benefit from the ability to provide different levels of services (e.g., bronze, silver, or gold), and a suite of service options (e.g., on-demand video or audio, and interactive video or audio conferencing) that would allow them to extract additional revenue from existing infrastructure. Non-commercial application of similar capabilities would enable the management of network resources. For example, portions of a network could be allocated along departmental (e.g., accounting, engineering, or sales), functional (e.g., instruction vs. research), and usage (e.g., video, audio, web, or e-mail) lines. Such service differentiation and resource management capabilities are not, in general, possible in the single forwarding class communications model used in the Internet today.

As a special case of service differentiation, the issues of security and trust have become critical for many modern applications of Internet technology. While security was important in the design of the Internet architecture, its implementation and deployment took lower priority to the implementation and deployment of the basic technology for what was still a very proof-of-concept communications architecture. More recent work (e.g. SSL, Allen and Dierks, 1999, and SSH) has focused on application-level, end-to-end security. This has left network-layer security and trust largely unresolved. In general, security and trust in the network layer revolve around questions of who can see traffic as it traverses an internet, and who can generate traffic load targeted at some point in an internet. The former represents a disclosure threat even for

end-to-end protected traffic where traffic analysis may result in significant disclosures. The latter represents a critical denial-of-service threat as has been demonstrated in the many large-scale DDoS attacks perpetrated in the Internet. Given the single forwarding class communications model underlying the current Internet architecture, these vulnerabilities are fundamentally unresolvable. While end-to-end solutions like those mentioned above can help mitigate the problems, the basic vulnerabilities remain.

The fundamental challenge of policy-based routing is to enhance the Internet routing architecture to support diverse network performance requirements and usage policies, without compromising the robustness, efficiency, and responsiveness of the existing distributed, hop-by-hop routing model. The remainder of this chapter reviews previously proposed solutions to this problem and identifies their limitations. It then presents an enhanced Internet routing architecture that supports these requirements for diverse policies without sacrificing the strengths of the original architecture. Lastly, it presents a new family of path-selection algorithms required by the new architecture that efficiently compute paths in the context of network performance and usage policies.

2. Previous Work

We define *policy-based routing* as the routing of traffic over paths in an internet that honor policies defining performance and resource utilization requirements of the internet. Based on this definition, *quality-of-service routing* (QoS) is the special case of routing in the context of performance policies, and *traffic engineering* is routing in the context of resource-utilization policies. This definition of traffic engineering is a generalization of that in current use. The current definition of traffic engineering can be stated as the management of network resources to *minimize or eliminate congestion* without the use of per-flow resource reservations. The generalized definition used here is the management of network resources to *implement arbitrary policies* without the use of per-flow resource reservations. Historically, QoS and traffic engineering have been addressed separately. The solution presented here is the first integrated solution to both. There are three main components to a policy-based routing solution: resource management, a routing architecture, and path-selection algorithms.

2.1 Policy-Based Resource Management

Two QoS architectures have been developed representing fundamentally different approaches to solving the problem of resource management

in the context of performance requirements. The goal of the *integrated services* (intserv) architecture (Braden et al., 1994) was to define an integrated Internet service model that supports best-effort, real-time, and controlled link sharing requirements. Intserv made 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 contrast, the *differentiated services* (diffserv) architecture 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 which classify traffic for one of these PHBs. Inside a diffserv domain, routing is performed using traditional hop-by-hop, single-forwarding class mechanisms.

Resource management for traffic engineering involves the specification of traffic classification rules to identify the policy-significant traffic in an internet, and the definition of resource utilization policies in terms of these traffic classes. The resource utilization policies are used as constraints in the path selection function to compute paths for difference traffic classes. Current proposals (Awduche et al., 1999) define resource-utilization policies by assigning network resources to resource classes, and then specifying what resource classes can be used for forwarding each traffic class.

2.2 Policy-Based Routing Architectures

A policy-based routing architecture defines path-selection mechanisms for computing paths through an internet that honor performance and resource-utilization policies, and forwarding mechanisms for forwarding traffic over these paths. While both the intserv and diffserv QoS resource management solutions support the use of single-forwarding-class routing models, the use of policy-based routing solutions results in significantly improved resource utilization. In contrast, traffic engineering resource management solutions require the use of a policy-based routing architecture.

Currently proposed policy-based routing architectures are based on an *on-demand, virtual-circuit* routing model where routes are computed on-demand (e.g. on receipt of the first packet in a flow, or on request by a network administrator), and forwarding is source-specified through

the use of source routing or path setup (Davie and Rekhter, 2000) techniques. These solutions are less robust, efficient, and responsive than the original distributed, hop-by-hop Internet routing architecture.

Specifically, these solutions are *less robust* due to their use of centralized control of state. For example, the forwarding paths in on-demand routing are brittle because the ingress router controls remote forwarding state in routers along paths it has set up. Furthermore, these solutions are *less efficient* and *responsive* due to their use of centralized control of state, and requirement of overly complex mechanisms for implementing some functions. Due to its centralized nature, on-demand routing requires the use of *duplex* communication of topology change events. Since the routing process controls remote forwarding state, a router requires two-way (duplex) communication to receive notification of an event, and then send forwarding state updates back into the internet. Additionally, on-demand routing is less efficient and responsive due to its requirement of complex mechanisms to implement their functionality. On-demand routing requires the use of full-topology routing algorithms to ensure that every router can compute optimal paths to any destination in an internet. Lastly, on-demand routing requires the use of more complex state management mechanisms, such as soft-state timers and repair mechanisms to manage forwarding state.

2.3 Policy-Based Path Selection

Policy-based path-selection supports traffic engineering by the computation of paths in the context of administrative constraints on the type of traffic allowed over links in an internet. Analogously, policy-based path-selection supports QoS by the computation of paths in the context of multi-component weights (Sobrinho, 2002) assigned to the links in an internet. The metrics used in these computations are assigned to individual links in the network. For a given routing application, a set of link metrics is identified for use in computing the path metrics used in the path-selection decision. Link metrics can be assigned to one of two classes based on how they are combined into path metrics. *Concave (or minmax) metrics* are link metrics for which the minimum (or maximum) value (called the bottleneck value) of a set of link metrics defines the path metric of a path composed of the given set of links. Examples of concave metrics include residual bandwidth, residual buffer space, and administrative traffic constraints. *Additive metrics* are link metrics for which the sum (or product, which can be converted to a sum of logarithms) of a set of link metrics defines the path metric of the path

composed of the given set of links. Examples of additive metrics include delay, delay jitter, cost, reliability, and packet loss..

The foundational work on the problem of computing paths in the context of more than one additive metric was done by Jaffe, 1984, who defined the multiply-constrained path problem (MCP) as the computation of paths in the context of two additive metrics, which is known to be NP-Complete (Garey and Johnson, 1979). He presented an enhanced distributed Bellman-Ford algorithm that solved this problem with time complexity of $O(n^4 b \log(nb))$, where n is the number of nodes in a graph, and b is the largest possible metric value. Since Jaffe, a number of solutions have been proposed for computing exact paths in the context of multiple metrics for special situations. Wang and Crowcroft, 1996 were the first to present the solution to computing paths in the context of a concave and an additive metric discussed above. Ma and Steenkiste, 1997 presented a modified Bellman-Ford algorithm that computes paths satisfying delay, delay-jitter, and buffer space constraints in the context of weighted-fair-queuing scheduling algorithms in polynomial time. Cavendish and Gerla, 1998 presented a modified Bellman-Ford algorithm with complexity of $O(n^3)$ which computes multi-constrained paths if all metrics of paths in an internet are either non-decreasing or non-increasing as a function of the hop count. Recent work by Siachalou and Georgiadis, 2003 on MCP has resulted in an algorithm with complexity $O(nW \log(n))$, where W is the maximumum link weight. This algorithm is a special case of the policy-based path-selection presented in Section 1.4 of this chapter. As described in Section 1.4, we have developed special case versions of the algorithm presented there for QoS and traffic engineering that provide significant improvements on existing solutions such as that presented in Siachalou and Georgiadis, 2003. However, comparisons with these solutions are not presented here due to space constraints.

While significant improvements in the performance of solutions to the general MCP problem have been obtained (as described above), these solutions still suffer from worst-case runtime that is exponential in the size of the graphs (specifically, worst-case runtimes are pseudopolynomial, Garey and Johnson, 1979). To address this problem several algorithms have been proposed that compute approximate solutions to the MCP problem. Both Jaffe, 1984 and Chen and Nahrstedt, 1998 propose algorithms which map a subset of the metrics comprising a link weight to a reduced range, and show that using such solutions, the cost of a policy-based path computation can be controlled at the expense of the accuracy of the selected paths. Similarly, a number of researchers (Jaffe, 1984; Mieghem et al., 2001) have presented algorithms which com-

pute paths based on a function of the multiple metrics comprising a link weight. In summary, the drawbacks of the current policy-based path selection solutions are that they have poor average case performance, they implement inflexible path selection models, and those based on algorithms that compute approximate solutions result in significant loss in fidelity of the path costs.

In summary, while existing proposals for QoS resource management will work with the traditional single-forwarding-class routing architecture, their effectiveness is severely limited. Furthermore, traffic engineering resource management cannot work with such routing architectures at all. To realize the potential of policy-based resource management, policy-based path-selection mechanisms must be used. Existing policy-based path-selection architectures are based on on-demand, virtual-circuit routing models which are inherently less robust, efficient, and responsive than the distributed, hop-by-hop model adopted by the Internet architecture. Furthermore, the performance of proposed exact solutions will not support the requirements of a distributed architecture.

3. Distributed Label-Swap Routing

Policy-based routing requires the ability to compute and forward traffic over multiple paths for a given destination. This is clearly the case for traffic engineering where multiple paths may exist that satisfy disjoint network usage policies. It is also true for QoS due to the fact that there may not exist a universally "best" route to a given node in a graph. For example, which of two paths is best when one has delay of $5ms$ and jitter of $4ms$, and the other has delay of $10ms$ and jitter of $1ms$ depends on which metric is more critical for a given application. For FTP traffic, where delay is important and jitter is not, the former would be more desirable. Conversely, for video streaming, where jitter is very important and delay is relatively un-important, the latter would be preferred. Such weights are said to be *incomparable.* In contrast, it is possible for one route to be clearly "better" than another in the context of multi-dimensional link weights. For example, a route with delay of $5ms$ and jitter of $1ms$ is clearly better than a route with delay of $10ms$ and jitter of $5ms$ for all possible application requirements. Such weights are said to be *comparable.*

The goal of routing in the context of multi-dimensional link weights is to find the largest set of paths to each destination with weights that are "better-than" all other routes in the graph with comparable weights (this definition will be made precise in Section 1.4.2). The weights in such a set are called the *performance classes* of a destination. (Formally, the

"better-than" relation is a partial ordering on the set of path weights, and the goal of a routing computation is to find the paths with the maximal set of path weights). Such a set of routes is not supported by the current Internet routing architecture because, as described above, the Internet only supports a single path between any given source and destination.

The solution proposed here is to use label-swap forwarding technology as a generalized forwarding mechanism to implement multiple paths per destination computed by policy-based path-selection algorithms. The path-selection algorithms compute a set of paths per destination that provide all combinations of performance and use policies existing in an internet. By replacing IP addresses with semantically neutral labels, routes can be assigned a local label, and label-swap forwarding can then be used to forward traffic for each class along an appropriate path. This combination of distributed, hop-by-hop routing with label-swap forwarding is called *distributed label-swap routing* (DLSR).

Traditionally, label-swap forwarding has only been seen as an appropriate match with an on-demand, source-driven routing model. Indeed, to a large extent, the virtual-circuit nature of these previous solutions has been attributed to their use of label-swap forwarding. Contrary to this view, we submit that host addresses and labels are largely equivalent alternatives for representing forwarding state, and that the virtual-circuit nature of prior architectures derives from their use of a source-driven forwarding model. The primary conceptual difference between address and label-swap forwarding is that label-swap forwarding provides a clean separation of the control and forwarding planes (Swallow, 1999) within the network layer, where address-based forwarding semantically ties the two planes together unnecessarily. The distinguishing characteristic of DLSR forwarding, as compared with MPLS (Davie and Rekhter, 2000) is that label-swap forwarding state is pre-computed in a distributed manner for all destinations, as compared with on-demand route computation in MPLS. This separation provides what might be called a *topological anonymity* of the forwarding plane that is critical to the implementation of policy-based routes.

Chandranmenon and Varghese, 1995 present a similar notion, which they call *threaded indices*, where neighboring routers share the indexes into their routing tables for specific routes which are then included in forwarded packets to allow rapid forwarding table lookups. In addition, they present a modified Bellman-Ford algorithm that exchanges these indices among neighbors. Distributed label-swap forwarding generalizes the threaded index concept to use generic labels (with no direct forwarding-table semantics), then uses these labels to represent rout-

ing policies computed by the routing protocols, and defines a family of routing protocols to exchange local labels among neighbors.

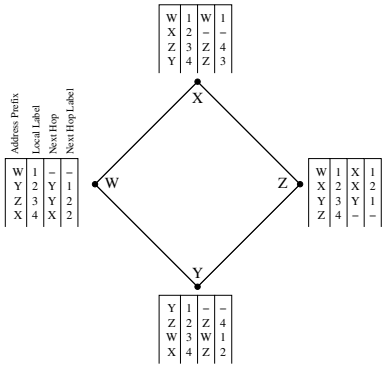

Figure 1.1. Labels with Address-Based Forwarding

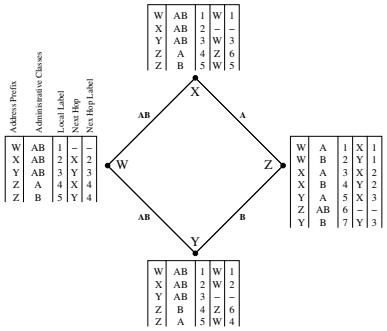

Figure 1.2. Labels with Policy-Based Forwarding

As illustrated in Figure 1.1 for traffic engineering, distributed label-swap forwarding can be used in the context of traditional address-based forwarding. This figure shows a four node network, with the forwarding tables at each node. In this example the forwarding table is referenced for both traffic classification (through the "address prefix" field), and for label-swap forwarding (through the "local label" field). The difference between DLSR forwarding and virtual-circuit mechanism (e.g. ATM and MPLS) is the use of topology-driven vs. on-demand routing computation. Specifically, in DLSR forwarding routes are pre-computed for all destination and traffic classes.

The benefit of this mechanism for traffic forwarding is that it can be generalized to handle policy-based forwarding. Specifically, distributed label-swap forwarding can be used to implement traffic engineering via

the assignment of traffic to administrative classes that are used to select different paths for traffic to the same destination depending on the labeling of links in the network with administrative class sets. For example, Figure 1.2 shows a small network with four nodes, two administrative classes A and B, and the given forwarding state for reaching the other nodes. The benefits of this architecture are that it is based on forwarding state that is agnostic to the definition of forwarding classes, which allows the data forwarding plane to remain simple yet general. Second, it concentrates the path computation functions in the routing protocol, which is the least time critical, and most flexible component of the network layer. This concept can be generalized to handle QoS in a straightforward manner.

The resulting routing architecture can be seen as analogous to the Reduced Instruction Set Computer (RISC) processor architecture in which researchers shifted much of the intelligence for managing the use of processor resources to the compilers that were able to bring a higher-level perspective to the task, thus allowing much more efficient use of the physical resources, as well as freeing the hardware designers to focus on performance issues of much simpler processor architectures. Similarly, the communications architecture proposed here requires a shift in intelligence for customized (i.e. policy-based) path composition to the routing protocols and frees the network layer to focus solely on hop-by-hop forwarding issues, adding degrees of freedom to the network hardware engineering problem that allow for significant advances in the performance and effectiveness of network infrastructure.

The enhancement of traditional unicast routing systems with the policy-based routing technology presented above is straight forward. The routing protocol must be enhanced to carry the additional link metrics required to implement the desired policies. This requires the use of either a link-state or link-vector routing protocol (Garcia-Luna-Aceves and Behrens, 1995) that exchanges information describing link state. However, for a system depending on on-demand routing computations, a topology broadcast protocol is required to ensure an ingress router has the information it needs to compute an optimal route. In contrast, hop-by-hop based routing systems can work with partial-topology protocols as each routing process is ensured of learning a topology from its neighbors containing optimal routes for reaching all destinations in an internet.

The forwarding state 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 be placed at the edge of an internet, where "edge" is defined to be any point

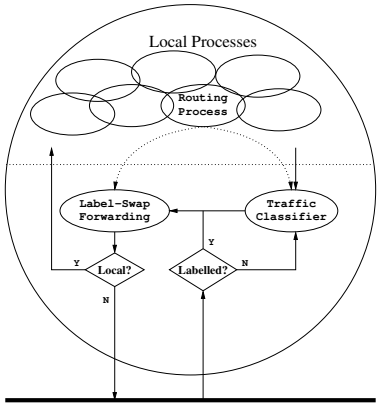

Figure 1.3. Traffic Flow in Policy-Enabled Router

from which traffic can be injected into the internet. Since each router represents a potential traffic source (for CLI and network management traffic), this effectively means a traffic classification component must be present in each router. As illustrated in Figure 1.3, the resulting traffic flow requirements are that all non-labeled traffic (sourced either from a router itself, or from a directly connected host or non-labeling router) must be passed through the traffic classifier first, and all labeled traffic (sourced either from the traffic classifier or a directly connected labeling router) must be passed to the label-swap forwarding process.

4. Policy-Based Path-Selection Algorithm

We model a network as a weighted undirected graph $G = (N, E)$, where N and E are the node and edge sets, respectively. By convention, the size of these sets are given by $n = |\, N \,|$ and $m = |\, E \,|$. Elements of E are unordered pairs of distinct nodes in N. $A(i)$ is the set of edges adjacent to i in the graph. Each link $(i, j) \in E$ is assigned a weight, denoted by ω_{ij}. A *path* is a sequence of nodes $< x_1, x_2, \ldots, x_d >$ such that $(x_i, x_{i+1}) \in E$ for every $i = 1, 2, \ldots, d-1$, and all nodes in the path are distinct. The weight of a path is given by $\omega_p = \sum_{i=1}^{d-1} \omega_{x_i x_{i+1}}$. The nature of these weights, and the functions used to combine these link weights into path weights are specified for each algorithm.

4.1 Administrative Policies

We use a *declarative* model of administrative policies in which constraints on the traffic allowed in an internet are specified by expressions in a boolean traffic algebra. The *traffic algebra* is composed of the stan-

dard boolean operations on the set $\{0, 1\}$, where a set of p primitive propositions (variables) represent statements describing characteristics of network traffic or global state that are either true or false. The syntax for expressions in the algebra is specified by the BNF grammar:

$$\varphi \ ::= \ 0 \mid 1 \mid v_1 \ldots v_p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi)$$

The set of primitive propositions, indicated by v_i in the grammar, can

Figure 1.4. Traffic Engineering Example

be defined in terms of network traffic characteristics or global state. For example (referring to Figure 1.4) *gold* could be a variable that is true (has the value 1) if the source or destination IP address of the current packet is that of a "gold" customer who has subscribed to a premium level of service from their Internet service provider (ISP), with *silver* and *bronze* defined similarly. In this context, Figure 1.4 illustrates the portion of the ISP's network connecting customers and the ISP's server farm that provides the services offered by the ISP. By assigning the link predicates as shown, the service provider has defined a network resource usage policy that grants gold customers access to all client-server interconnections, silver customers access to two-thirds of this connectivity, and bronze customers to one-third. Consequently, premium customers get quantifiably improved service relative to lower-priority customers, both in terms of reliability and performance.

In addition, a $SAT(\varphi)$ primitive is required for expressions in the traffic algebra which is the SATISFIABILITY problem of traditional propositional logic. Satisfiability must be tested in two situations by the algorithms presented below for the implementation of traffic-engineering computations. First, an extension to a known route should only be considered if classes of traffic exist that are authorized to use both the path represented by the known route and the link used to extend the path. This is true *iff* the conjunction of these expressions is satisfiable (i.e., $SAT(\varepsilon_i \wedge \varepsilon_{ij})$ where ε_i is the predicate for the path to i, and ε_{ij} is the predicate for the link from i to j). Second, given that classes of traffic exist that are authorized to use a path represented by a new route, the algorithms must determine whether all traffic supported by that route

has also been satisfied by other previously discovered shorter routes. This is true *iff* the new route's traffic expression implies the disjunction of the traffic expressions for all known better routes (i.e., $(\varepsilon_i \rightarrow \varepsilon_{i_1}, \varepsilon_{i_2}, ..)$ is *valid*, which is denoted by $(\varepsilon_i \rightarrow \mathcal{E}_i)$ in the algorithms). Determining if an expression is valid is equivalent to determining if the negation of the expression is unsatisfiable. Therefore, expressions of the form $\varepsilon_1 \rightarrow \varepsilon_2$ are equivalent to $\neg SAT(\neg(\varepsilon_1 \rightarrow \varepsilon_2))$ (or $\neg SAT(\varepsilon_1 \wedge \neg\varepsilon_2)$).

SATISFIABILITY is the prototypical NP-complete problem (Garey and Johnson, 1979). As is typical with NP-complete problems, it has many restricted versions that are computable in polynomial time. An analysis of strategies for defining computationally tractable traffic algebras is beyond the scope of this paper; however, we have implemented an efficient, restricted solution to the SAT problem by implementing the traffic algebra as a set algebra with the set operations of intersection, union, and complement on the set of all possible forwarding classes.

In summary, administrative policies are specified for an internet by a set of link and global predicates. These predicates define a set of *forwarding classes*, and constrain the topology that traffic for each forwarding class is authorized to traverse, as required by the administrative policies.

4.2 Performance Characteristics

As described in Section 1.2.3, *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. As discussed in Section 1.3, there is not a universally "best" route between two nodes in a graph in the context of multi-component weights. To address this fact, the routing algorithm presented here is based on an enhanced version of the path algebra defined by Sobrinho, 2002, which supports the computation of a *set* of routes for a given destination containing the "best" set of routes for each destination.

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 $\preceq$, called "lighter-than," is the original total ordering from Sobrinho, 2002. 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.

A new relation on routes, $\sqsubseteq$, called "better-than," 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, antisymmetric, and transitive) with the following, additional property:

PROPERTY 1.1 $(\omega_x \sqsubseteq \omega_y) \Rightarrow (\omega_x \succeq \omega_y)$.

This relation is equivalent to the concept of *dominated* paths (Henig, 1985). 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 cost is as follows:

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

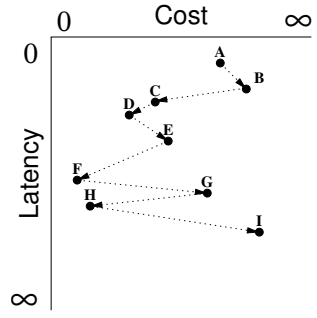

Figure 1.5. $\preceq$ relation

Figure 1.5 is a graphical depiction of the relation $\preceq$ on a set of weights for routes (labeled A through I) to a given destination in an internet where $x \preceq y$ is depicted as $x \rightarrow y$. Figure 1.6 illustrates the relation $\sqsubseteq$ where each route is represented as a subset of the plane with upper

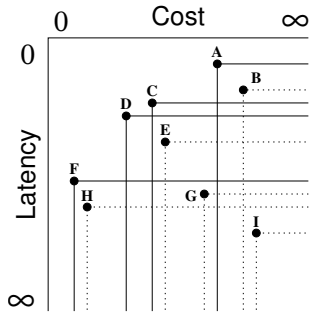

Figure 1.6. $\sqsubseteq$ relation

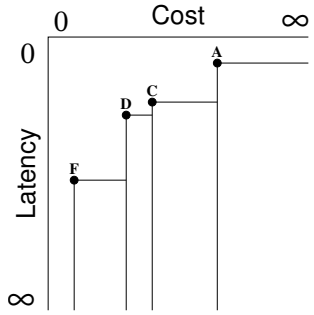

Figure 1.7. Forwarding table

left-hand corner at the coordinates for the route. The intuition communicated here is that a route satisfies any constraint pair contained in its sub-region of the plane. Building on this intuition, the relation $\sqsubseteq$ defines an ordering on routes in terms of the containment (subset) of one route's region within another's, i.e. if $\omega_i \sqsubseteq \omega_j$, then the set of constraint pairs that route i can satisfy is a subset of those satisfiable by route j. The maximal subset of a set of such routes (the set of routes shown with solid lines in Figure 1.6) contain routes that satisfy all constraint pairs satisfiable by any route in the internet, and is the goal of the routing computation. Clearly, any pair of routes in the maximal subset of routes overlap, and can both satisfy some set of constraint pairs. The relation $\preceq$ is used to select one of the set of satisfying routes for a given constraint. As defined in this example, the relation $\preceq$ has the effect of truncating the extent of a route's region at the first overlapping route to the right in the maximal subset of routes (as shown in Figure 1.7). As a result, forwarding table lookups in this example involve choosing the lowest delay route with acceptable cost.

Table 1.1. Notation.

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.

The goal of routing in the context of multi-component link weights is to find the largest set of paths to each destination with weights that are "better-than" all other routes in the graph with comparable weights. (Formally, the "better-than" relation is a partial ordering on the set of path weights, and the goal of a routing computation is to find the paths with the maximal set of path weights).

In summary, the goal of policy-based routing is to compute the maximal set of routes to each destination in an internet for each traffic class for which a path to the destination exists. In terms of the DLSR architecture, this translates to computing the set of performance classes for each forwarding class in an internet. Realizing this goal requires the ability to efficiently compute paths in the context of link predicates and multi-component link weights, and to efficiently forward traffic over the multiple paths per destination resulting from such a computation.

4.3 Path Selection

The notation used in the algorithms presented in the following is summarized in Table 1.1. In addition, the maximum number of unique truth assignments is denoted by $A = 2^p$, the maximum number of unique weights by $W = \min(range\ of\ weight\ components)$, and the maximum number of adjacent neighbors by $a_{max} = \max\{\mid A(i) \mid \mid i \in N\}$. Table 1.2 defines the primitive operations for queues, heaps, and balanced trees used in the algorithms, and gives their time complexity used in the complexity analysis of the algorithms (where d is the degree of the tree implementing the heap structure).

The algorithm presented in Figure 1.9 implements an SPF-style search through the paths in a graph in order of increasing weight (in terms of $\preceq$), adding the paths with maximal weights (in terms of $\sqsubseteq$) for each traffic class (defined by the traffic algebra). This algorithm is based on the data structure model shown in Figure 1.8. 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

Table 1.2. Operations on Data Structures (Ahuja et al., 1993).

Notation	*Description*
	Queue
$Push(r, Q)$	Insert record r at tail of queue Q ($O(1)$)
$Head(Q)$	Return record at head of queue Q ($O(1)$)
$Pop(Q)$	Delete record at head of queue Q ($O(1)$)
$PopTail(Q)$	Delete 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 ($O(\log(n))$)

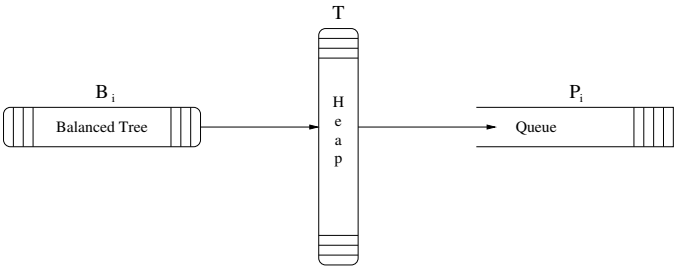

Figure 1.8. Model of Data structures for Basic Algorithms

```
algorithm Policy-Based-Dijkstra
    begin
1   Push(< s, s, 0̄, 1 >, P_s);
2   for each {(s, j) ∈ A(s)}
3      Insert(< j, s, ω_sj, ε_sj >, T);
4   while (| T | = 0)
       begin
5      < i, p_i, ω_i, ε_i > ← Min(T);
6      DeleteMin(B_i);
7      if (| B_i | = 0)
8         then DeleteMin(T)
9         else IncreaseKey(Min(B_i), T_i);
10     ε_tmp ← ε_i; ptr ← Tail(P_i);
11     while ((ε_tmp ≠ 0) ∧ (ptr ≠ ∅))
12        ε_tmp ← ε_tmp ∧ ¬ptr.ε; ptr ← ptr.next;
13     if (ε_tmp ≠ 0)
          then begin
14           Push(< i, p_i, ω_i, ε_i >, P_i);
15           for each {(i, j) ∈ A(i) | SAT(ε_i ∧ ε_ij}
                begin
16              ω_j ← ω_i ⊕ ω_ij; ε_j ← ε_ij;
17              if (T_j = ∅)
18                 then Insert(< j, i, ω_j, ε_j >, T)
19              else if (ω_j ≺ T_j.ω)
20                 then DecreaseKey(< j, i, ω_j, ε_j >, T);
21              Insert(< j, i, ω_j, ε_j >, B_j);
                end
             end
       end
    end
```

Figure 1.9. General-Policy-Based Dijkstra.

the lightest weight entry from each non-empty B_i (for a maximum of n entries). Lastly, 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 this 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 add "relaxed" routes for its neighbors to the appropriate B_x's.

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 to a given destination i in P are incomparable for some set of satisfying truth assignments, and the maximal subset of routes to a given destination j in $P_j \cup B_j$ represents the maximal subset of all paths to j 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 to a destination i is moved to P_i.

In effect, this algorithm computes routes in the *virtual graph* induced by the link predicates existing in the internet. This virtual graph is

composed of all nodes reachable by some path with a satisfiable path predicate, and all links composing these paths. The virtual graph is "discovered" as needed by the algorithm as the computation progresses. The time complexity of Policy-Based-Dijkstra is dominated by the loops at lines 4, 11, and 15. The loop at line 4 is executed nWA times, and the loop at line 15 mWA times. The loop at line 11 scans the entries in P_i to verify a new route is best for some truth assignment. For a given destination, this loop is executed at most an incrementally increasing number of times, starting at 0 and growing to $WA - 1$ (the maximum number of unique routes to a given destination) for a total of $\sum_{i=1}^{WA-1} i = \frac{(WA-1)WA}{2}$ times. For completeness, the statements in lines 6 and 21 take time proportional to $\log(a_{max}WA)$ for a total of $nWA\log(a_{max}WA)$ and $mWA\log(a_{max}WA)$, respectively; and those in lines 7–9 and 17–20 take time proportional to $\log_d(n)$ for a total of $nWA\log_d(n)$ and $mWA\log_d(n)$, respectively. Therefore, the worst-case time complexity of Policy-Based-Dijkstra, dominated by the loop in line 11, is $O(nW^2A^2)$. In practice, the cost of this algorithm is limited by the *actual number of distinct weight paths* in the graph. The loop at line 11, which dominates the cost of Policy-Based-Dijkstra, is required because there is no way to summarize the permanent routes for a destination. However, for special-case traffic engineering and QoS variants of this algorithm, the permanent routes can be summarized by a summary traffic expression (formed by the disjunction of permanent route path predicates) and the weight of the last route, respectively. Using these shortcuts, the complexity of the traffic engineering and QoS algorithms are $O(mA\log(A))$ and $O(mW\log(W))$, respectively. Lastly, for these variants, refinements in the data structures result in $O(mA\log(n))$and $O(mW\log(n))$ complexity. The details of these variants and their analysis can be found in Smith, 2003.

4.3.1 Routing Protocol Changes. Lastly, the routing protocol must be enhanced to exchange information needed to compute the label swap components of its forwarding tables. The output of the routing algorithm is forwarding information described in terms of a destination, traffic expression, and path weight for each computed route. To be used for forwarding, this information must be augmented with local and next hop labels. To determine the next hop label for a given route the routing process requires the forwarding tables of its neighbors. Therefore, the final enhancement required of routing protocols is that they exchange local forwarding tables and use this information to compute the next hop label for their routes. One challenge presented by this requirement is that the routes computed by the routing algorithm must

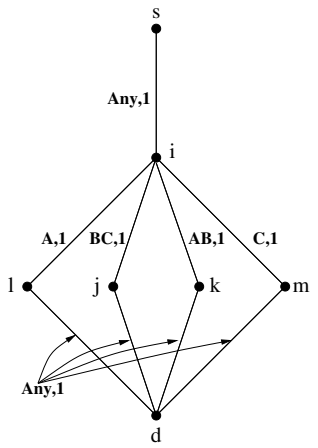

Figure 1.10. Next Hop Problem with Policy-Based Routing

be assured of matching an active route in the selected next hop neighbor. As illustrated in Figure 1.10, this is not guaranteed by the algorithms presented above. Specifically, in this internet there are a number of equally "good" routes from nodes s and i to node d. For example, it is possible that the routing process at node i selects the paths through its neighbors l and j to provide two hop paths for traffic classes A, B, and C, while node s selects the paths that go through nodes k and m. In such a case there is no next hop label that can be chosen at s for routes to d that will satisfy the traffic policies.

To address this problem, Figure 1.11 presents an enhanced version of the algorithm for use in the context of hop-by-hop forwarding. In this algorithm, routes are augmented with two additional fields; n_d is the next hop neighbor for a route to destination d, and l_d is the next hop label for d. As described above, a partial forwarding table is maintained for each neighbor, specified by $F_n[d]$, containing an array of routes for each destination in the internet. Each entry in this array, denoted by $< d, \omega_d, \varepsilon_d, l_d >$, gives the weight, traffic expression, and next hop label for each route in the neighbor's forwarding table. In this algorithm, new paths are only considered if they are extensions of paths chosen by the neighbor which is the next hop to the predecessor to the path's destination. For example, from Figure 1.10, node s will only consider paths to destination d that are extensions of node i's paths to d through nodes l and j. A fringe benefit of this enhancement is the next hop label computation can now be integrated with the routing computation (as shown by the inclusion of the next hop label in the routes computed by the algorithm).

```
algorithm HbyH-Policy-Based-Dijkstra
    begin
1   Push(< s, s, 0̄, 1, s, ∅ >, P_s);
2   for each {(s, j) ∈ A(s)}
3      Insert(< j, s, ω_sj, ε_sj, j, ∅ >, T);
4   while (| T | = 0)
       begin
5      < i, p_i, ω_i, ε_i, n_i, l_i > ← Min(T);
6      DeleteMin(B_i);
7      if (| B_i | = 0)
8         then DeleteMin(T)
9         else IncreaseKey(Min(B_i), T_i);
10     ε_tmp ← ε_i; ptr ← Tail(P_i);
11     while ((ε_tmp ≠ 0) ∧ (ptr ≠ ∅))
12        ε_tmp ← ε_tmp ∧ ¬ptr.ε; ptr ← ptr.next;
13     if (ε_tmp ≠ 0)
          then begin
14           Push(< i, p_i, ω_i, ε_i, n_i, l_i >, P_i);
15           for each {(i, j) ∈ A(i) |
                    (∃ < j, ω'_j, ε'_j, l'_j > ∈ F_{n_i}[j] |
                       (ε_{sn_i} ∧ ε'_j = ε_i ∧ ε_ij) ∧
                       (ω_{sn_i} + ω'_j = ω_i + ω_ij)) ∧
                    SAT(ε_i ∧ ε_ij)}
                 begin
16               ω_j ← ω_i ⊕ ω_ij; ε_j ← ε_ij;
17               if (T_j = ∅)
18                  then Insert(< j, i, ω_j, ε_j, n_i, l'_j >, T)
19               else if (ω_j ≺ T_j.ω)
20                  then DecreaseKey(< j, i, ω_j, ε_j, n_i, l'_j >, T);
21               Insert(< j, i, ω_j, ε_j, n_i, l'_j >, B_j);
                 end
              end
       end
    end
```

Figure 1.11. General-Policy-Based Dijkstra.

5. Conclusions

In this chapter we have defined policy-based routing as the computation of paths, and the establishment of forwarding state to implement paths in the context of diverse performance requirements and network usage policies. We showed that a fundamental requirement of policy-based routing is support for multiple paths to a given destination, and that the address-based, single-forwarding-class Internet routing model can't support such a requirement. Furthermore, while proposed QoS resource management solutions are defined to work in a single-forwarding-class environment, their effectiveness is significantly limited by such a constraint. We then showed that previously proposed policy-based routing solutions, which are based on an on-demand, virtual-circuit model, result in significant compromises in robustness, efficiency, and responsiveness in comparison to the Internet's distributed, hop-by-hop routing model. We then presented the DLSR routing architecture in which the Internet's distributed, hop-by-hop routing model is combined with a

label-swap forwarding plane. DLSR is the first distributed, hop-by-hop policy routing architecture, and the first policy routing solution that provides integrated support of QoS and traffic engineering. Lastly, we presented a new family of efficient path-selection algorithms for use in a DLSR-based routing architecture that compute paths in the context of diverse performance requirements and network resource usage policies.

References

Ahuja, Ravindra K., Magnanti, Thomas L., and Orlin, James B. (1993). *Network Flows – Theory, Algorithms, and Applications*. Prentice Hall.

Allen, Christopher and Dierks, Tim (1999). The TLS Protocol Version 1.0. RFC 2246.

Awduche, Daniel O., Malcolm, Joe, Agogbua, Johnson, O'Dell, Mike, and McManus, Jim (1999). Requirements for Traffic Engineering Over MPLS. RFC 2702.

Braden, Bob, Clark, David, and Shenker, Scott (1994). Integrated Services in the Internet Architecture: an Overview. RFC 1633.

Cavendish, D. and Gerla, M. (1998). Internet QoS Routing using the Bellman-Ford Algorithm. In *Proceedings IFIP Conference on High Performance Networking*. IFIP.

Cerf, Vinton G. (1978). The Catenet Model for Internetworking. IEN 48.

Cerf, Vinton G. and Cain, Edward (1983). The DoD Internet Architecture Model. *Computer Networks*, 7:307–318.

Cerf, Vinton G. and Kahn, Robert E. (1974). A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, COM-22(5):637–648.

Chandranmenon, Girish P. and Varghese, George (1995). Trading Packet Headers for Packet Processing. *IEEE ACM Transactions on Networking*, 4(2):141–152. 1995.

Chen, Shigang and Nahrstedt, Klara (1998). An Overview of Quality of Service Routing for Next-Generation High-Speed Networks: Problems and Solutions. *IEEE Network*, pages 64–79.

Clark, David D. (1988). The Design Philosophy of the DARPA Internet Protocols. *Computer Communications Review*, 18(4):106–114.

Davie, Bruce and Rekhter, Yakov (2000). *MPLS: Technology and Applications*. Morgan Kaufmann.

Garcia-Luna-Aceves, J.J. and Behrens, Jochen (1995). Distributed, Scalable Routing Based on Vectors of Link States. *IEEE Journal on Selected Areas in Communications*.

Garey, Michael R. and Johnson, David S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co.

Henig, Mordechai I. (1985). The shortest path problem with two objective functions. *European Journal of Operational Research*, 25:281–291.

Jaffe, Jeffrey M. (1984). Algorithms for Finding Paths with Multiple Constraints. *Networks*, 14(1):95–116.

Ma, Qingming and Steenkiste, Peter (1997). Quality-of-Service Routing for Traffic with Performance Guarantees. In *Proceedings 4th International IFIP Workshop on QoS*. IFIP.

Mieghem, Piet Van, Neve, Hans De, and Kuipers, Fernando (2001). Hop-by-hop quality of service routing. *Computer Networks*, 37:407–423.

Siachalou, Stavroula and Georgiadis, Leonidas (2003). Efficient QoS Routing. In *Proceedigns of INFOCOM'03*. IEEE.

Smith, Bradley R. (2003). *Efficient Policy-Based Routing in the Internet*. PhD thesis, University of California at Santa Cruz.

Sobrinho, João Luís (2002). Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet. *IEEE/ACM Transactions on Networking*, 10(4):541–550.

SSH. SSH Communications Security. http://www.ssh.com/.

Swallow, George (1999). MPLS Advantages for Traffic Engineering. *IEEE Communications Magazine*, 37(12):54–57.

Wang, Zheng and Crowcroft, Jon (1996). Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, pages 1228–1234.