UNIVERSITY OF CALIFORNIA
SANTA CRUZ

EFFICIENT POLICY-BASED ROUTING IN THE INTERNET

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Bradley R. Smith

September 2003

The Dissertation of Bradley R. Smith
is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Darrell D. E. Long

Professor Patrick E. Mantey

Frank Talamantes
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

Efficient Policy-Based Routing in the Internet

by

Bradley R. Smith

Traditional Internet routing has focused on shortest-path routing where paths are chosen which minimize an additive weight function on a single, typically delay-related metric. The evolving topology control requirements of the Internet require the generalization of this model to satisfy functions on multiple metrics. The inclusion of multiple metrics in a routing computation is called policy-based routing. Policy-based routing supports traffic engineering by the computation of routes in the context of constraints on the traffic allowed over portions of an internet. Analogously, policy-based routing supports quality-of-service (QoS) by the computation of routes in the context of constraints on the paths specific traffic flows are allowed to use. Previous work on policy-based routing has focused on virtual-circuit-based solutions, and has resulted in computationally expensive algorithms. This paper presents a number of advances in the provision of policy-based routing services in networks and internetworks. A family of routing algorithms are presented for computing routes in the context of traffic-engineering constraints, quality-of-service constraints, and a combination of the two, which achieve new levels of computational efficiency. In addition, a forwarding architecture is presented that efficiently supports hop-by-hop forwarding in the context of multiple paths to each destination, which is desirable for policy-based routing.

Acknowledgements

At the end of such a long (16 year!) trip the list of people to be acknowledged is quite long. I would like to thank Profs. Pat Mantey and Darrell Long for providing the strategic nudges needed to get me looking and moving in a new, exciting direction, and for serving on my dissertation committee. I am grateful to the Computer Science and Computer Engineering staff and faculty at large for providing a supportive and encouraging environment. I am especially grateful to the technical staff for providing an exciting, stimulating, challenging, and always entertaining environment to work in. I would also like to give special thanks to Profs. Phokion Kolaitis, Allen Van Gelder, and Tracy Larrabee for going above and beyond the call of duty in their efforts to answer my questions regarding, and open my eyes to the beauty and power of formal methods and mathematical logic which have become critical components of my problem solving toolbox. I owe thanks to all the graduate administration folks, including Carol Mullane in SoE, and the people in the Graduate Division and Registrar's office for keeping my case alive for these oh-so-many years. I would like to thank the CoCo group, including Ewerton (and Daniela and Marcela), Jyoti, Marcelo Spohn (and Luci and baby), Lichun, Soumya, Chane, Chris and Mike, Srinivas, Marcelo Carvalho, Marco, Ramesh, Saro, Sharon, Rodrigo, Brian, and Clay for a stimulating and supportive research environment in JJ's Computer Communication Research Group.

I owe my deepest thanks to Prof. J.J. Garcia-Luna-Aceves for the opportunity and mentorship he has given me in reaching this goal. The terms "teacher"and "advisor" are used all too lightly in day-to-day discourse in that they embody the

selfless giving of hard-earned knowledge, skills, and discipline that result, at the doctoral level, in the creation of a peer. There are few places in life where gifts of such magnitude are given so freely. Thank you JJ, for the opportunity, inspiration, and occasional swift kick in the pants!

Words cannot express the debt and gratitude I owe to my family. This dissertation would not have been possible without the love and support of my wife, Alicia. My parents, who somehow, somewhere instilled in me the values and fortitude I've drawn on to set and reach this goal, are due far more than thanks or gratitude. This degree is truly theirs more than mine.

Chapter 1

Motivation

The architecture of today's Internet has its origins in the original ARPANET protocol architecture [1]. In the ARPANET architecture, a communication backbone was used to interconnect host computers directly attached to the backbone. The backbone was formed by packet switches called Interface Message Processors (IMP), and a host attached to an IMP through a terminal IMP processor (TIP). The protocols needed to establish end-to-end communication in support of such applications as file transfer and remote login were IMP-to-IMP, IMP-to-host, and host-to-host protocols. The host-to-host protocol was called the Network Control Program (NCP). The IMP-to-IMP protocol supported reliable communication between neighboring IMPs and routing of packets from source to destination IMP. The IMP-host protocol supported the passage of messages between a host and an IMP in order to create a virtual communication path between hosts. NCP supported the communication between two remote hosts attached to the ARPANET.

As local area networks (LANs), the DARPA packet radio network (PRNET) and the DARPA satellite network (SATNET) evolved, the ARPANET backbone started to grow. Consequently, the end-to-end communication among hosts could no longer assume packet switches in a common backbone, running the same protocol for packet forwarding. The need to mask the details of how packets, originated and consumed by hosts, were forwarded within specific networks was imperative to the ability to interconnect any set of hosts through any set of networks. To cope with the need for interconnecting hosts on an end-to-end basis through many computer networks rather than a single backbone, Cerf et al proposed the catenet model.

The modern Internet Architecture is based on the *catenet model for internetworking* [18, 19, 20]. A catenet is defined to be a "collection of packet networks that are connected together." The two basic components of a catenet are networks and gateways, where a catenet is formed by the interconnecting of networks with gateways. A primary goal of the catenet model, and therefore the Internet Architecture, was to encourage the development and integration of new networking technologies into the developing catenets. To achieve this goal, only minimal assumptions were made of networks by the catenet model. Specifically, networks were assumed to support the attachment of a number of computers, transport datagrams, allow switched access so that attached computers could "quickly" send datagrams to different destinations, and provide best-effort delivery, where the definition of best-effort allowed datagrams to be dropped, or delivered out of order.

This best-effort model of communication has proven surprisingly powerful.

Indeed, much of the success of the Internet Architecture can be attributed to this inspired design decision. However, largely as a product of its own success, limitations of the Internet Architecture are being encountered as new application and management requirements are made of it as it is applied to ever more demanding environments [11]. Real-time applications, such as on-demand streaming, audio and video conferencing, visualization, and virtual reality require varying degrees of bandwidth, delay, and delay jitter commitments from the network infrastructure. Furthermore, to support the efficient management of network resources, traffic engineering [5] and network management services require the ability to control the allocation of these resources in an internet among network flows. This is accomplished by the assignment of traffic classes or "colors" to links, specifying the traffic routable over a given link. Similarly, to provide protection from denial-of-service between classes of traffic, and from disclosure of sensitive traffic from transmission over insecure links in a network, it must be possible to constrain the topology used for the forwarding of traffic through an internet. Fundamental to these new requirements is the ability to control the link-layer topology used to forward network-layer traffic.

Traditional Internet routing has focused on shortest-path routing where paths are chosen which minimize an additive weight function on a single, typically delay-related metric. The evolving topology control requirements of the Internet require the generalization of this model to satisfy functions on multiple metrics. The inclusion of multiple metrics in a routing computation is called *policy-based* routing [22, 87]. Policy-based routing supports *traffic engineering* via the enforcement of administra-

tive constraints on what subset of an internet is authorized to carry a given traffic class. Analogously, policy-based routing supports *quality-of-service* (QoS) by the enforcement of constraints on the performance characteristics of topologies that can be used to carry traffic for different traffic classes.

Metrics used in routing 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 routing 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 where 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 constraints (described in Section 2.2). *Additive metrics* are link metrics where 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, and reliability.

While, in general, policy-based routing is an NP-complete problem [36, 42], there are many sub-classes of this general problem that have been shown to have polynomial-time solutions. For example, any problem involving two metrics with at least one of them concave can be solved in polynomial-time by a traditional shortest path algorithm on the graph where all links that do not comply with the concave constraints have been pruned [22, 52, 87]. However, even for this case, as the number

of constraints becomes exponential in the size of the graph, this result no longer holds.

The foundational work on the problem of computing routes in the context of more than one additive metric was done by Jaffe [42] where he defined the multiply-constrained path problem (MCP) as the computation of routes in the context of two additive metrics. He presented an enhanced distributed Bellman-Ford algorithm that solved this problem with time complexity of $O(n^4 b \log nb)$, where b is the largest possible metric value. An algorithm with run time complexity of this form is called pseudopolynomial [36] in that it is not polynomial in the length of the input (which would be $n \log b$ in this instance), however it is polynomial in the length of the input and the *largest value* in the input (which is b in this case). The significance of an algorithm being pseudopolynomial is this indicates the possibility that its run time complexity can be kept polynomial by limiting the largest value in the input to some polynomial function of the length of the input.

As a result, since Jaffe's work, efforts at finding general, efficient solutions to MCP and related policy-based routing algorithms have focused on the metric values as the source of the computational complexity of policy-based routing algorithms. Jaffe's paper itself analyzed metric-focused approaches to controlling the run time costs of MCP including mapping unbounded metrics to smaller ranges, and the use of a function of the two metrics for computing approximate solutions. The drawbacks of the current policy-based routing solutions are they have poor average case performance, they implement inflexible routing models, and solutions for computing approximate solutions do not work with the administrative constraints used for traffic engineering.

This dissertation presents a number of advances in the provision of policy-based routing services in an Internet environment. New policy-based routing algorithms are presented that significantly improve the average case time complexity of computations in the context of administrative constraints for traffic engineering, and the worst case space and time complexity of routing computations in the context of performance constraints for the satisfaction of QoS requirements. A more comprehensive model of the time complexity of policy-based routing is presented which suggests new, more effective strategies for controlling these costs that work in the context of both traffic engineering and QoS requirements, and new algorithms are explored based on these insights. Lastly, a model is developed for efficiently implementing table-driven, policy-based routing in the Internet using existing label-swap mechanisms.

Section 2 presents the network model and reviews basic architectural issues. In Section 3 a family of policy-based routing algorithms are presented and analyzed that compute optimal routes in the context of policy-based link metrics. Section 4 presents a new approach to computing approximate solutions that address the run time costs of the optimal solutions in the context of general policy-based link metrics. Section 5 discusses traffic expression processing, and possible approaches to dealing with the inherent computational complexity of this problem. Lastly, Section 6 and 7 discuss applications of these technologies to intra-domain and inter-domain policy-based routing.

Chapter 2

Preliminaries

In this dissertation a network is modeled 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 $n = \mid N \mid$ and $m = \mid E \mid$. 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.

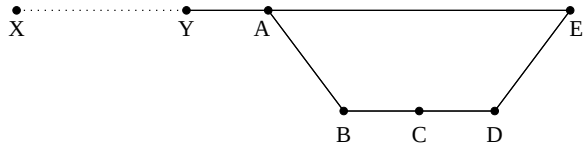

Figure 2.1: Effects of Routing Model

2.1 Routing Computation Model

The routing model used by the Internet architecture is a *table-driven, hop-by-hop* routing model. In this model routers learn about the state of connectivity in an internet by exchanging messages with each other, and run local routing computations whose output is a forwarding table. This forwarding table is used by the router's forwarding process to make per-packet forwarding decisions. There are a number of advantages to this routing model. First, changes in network state are detected earliest by those routers needing to change their forwarding tables (e.g. the damage caused by the failure of link (A, E) in Figure 2.1 is repaired after nodes A through D are notified, which are the first nodes to receive notification of the failure). Second, the autonomous control of the forwarding tables significantly simplifies the coordination of inter-router state (e.g. adaptation to the failure of link (A, Y) in Figure 2.1 does not require any change in the forwarding tables of nodes A through E). Lastly, information about a link needs only be propagated to those routers using the link in their forwarding topology (e.g. the information regarding link (D, E) in Figure 2.1 only needs to be propagated to nodes A, C, D, and E). As a result, routing is implemented in the Internet as a distributed routing computation, with fully autonomous control of forwarding state, that is very efficient, responsive, and robust.

In contrast, many recent policy-based routing proposals have endorsed an *on-demand, source-driven* routing model where routes are computed on receipt of a new connection request, and forwarding is source driven through the use of path setup or source routing techniques. In contrast to the table-driven, hop-by-hop model, there are a number of weaknesses to the on-demand, source-driven model. First, changes in network state must be propagated to the source, and topology change requests propagated back into the network to adapt to changes in an internet (e.g. the damage caused by the failure of link (A, E) in Figure 2.1 is repaired only after node X is notified, able to compute an alternate path, and communicates the necessary forwarding table changes back to nodes A through E). Second, it implements a centralized model of routing control where forwarding state for all sources at a given point in the network is maintained by the router acting for those sources (e.g. adaptation to the failure of link (A, Y) in Figure 2.1 requires the deletion of forwarding state in nodes A through E by node X). Lastly, information about a link must be propagated to all routes in an internet (e.g. information regarding link (D, E) must be propagated to all nodes in the internet to allow the computation of routes to E under any link failure scenario). As a result, in these proposals, routing is implemented as a centralized routing computation, with remote control of forwarding state, that is less efficient, responsive, and robust to current solutions. Therefore, while the algorithms developed for this thesis will work with either routing model, a primary goal of their design has been that they be compatible with a table-driven, hop-by-hop model.

2.2 Administrative Constraints for Traffic Engineering

The original motivation for traffic engineering services was the need by IP network providers to manage network bandwidth in the context of the single-path routing model of IP networks [81]. Lacking such capabilities, the tendency of single-path routing is to aggregate traffic for a given destination onto a subset of the possible paths to that destination. As a result, networks frequently experience congestion in spite of the availability of excess capacity for the offered load. To avoid this problem, network providers developed the technique of deploying IP networks over a circuit switched layer 2 technology (e.g. ATM or Frame Relay), and implementing a richly connected topology of virtual circuits among the routers composing the provider's backbone. By manipulating the paths followed by individual virtual circuits it was possible to distribute traffic more evenly over the underlying network.

There are a number of problems with this solution, however, including the cost and responsiveness of manual intervention, the fidelity of IP routing decisions based on an abstracted topology, the magnification of link events over all circuits using a given link, and the inefficiency of IP routing protocols operating in network density regimes they weren't designed for. As a result, the requirements of traffic engineering mechanisms are that: they require minimal human intervention only to establish the policies to be used in computing forwarding topologies, they efficiently and effectively forward traffic over the underlying network topology in accordance with these policies, and that traffic-engineering-enhanced IP routing operates directly on the link-layer topology. Given this basic definition, traffic engineering services can be

applied to a number of other problems.

The vulnerability of IP services to basic network disclosure and denial of service threats is another result of the lack of topology control available from current IP mechanisms. Excluding computationally expensive firewall mechanisms, IP traffic can potentially traverse any link in an internet. Therefore, the disclosure of any traffic stream and the denial-of-service attack of any destination in an internet are unavoidable threats. While unicast communication provides some protection against disclosure in that forwarding paths will naturally follow the underlying network topology, allowing networks to be structured such that sensitive information will only traverse trusted infrastructure, these protections are delicate. The paths computed in unusual failure modes are difficult to predict, and may violate security policies. Additionally, maintaining the topology invariants necessary to meet given security policies in the context of on-going network evolution is difficult, at best. And for multicast communication even these limited controls are lost as the location of sources and destinations is determined by the users joining the group, magnifying the potential for disclosure of multicast group communication. As a result, the forwarding topology for traffic in a given group is not controllable. For denial-of-service the situation is worse in that, if a destination is reachable from a portion of an internet, it is reachable by all nodes in that region via the same path. The traffic engineering capability to directly express constraints on what topologies can be used to carry given classes of traffic, and to have these constraints be honored in the routing computation, is a powerful tool for mitigating the affects of these threats.

Lastly, traffic engineering services provide important functionality for viable QoS services. QoS services come in both virtual-circuit (e.g. IntServ [11]) and hop-by-hop (e.g. DiffServ [10]) forms. Both forms provide specific mechanisms for satisfying resource requirements of traffic forwarded along QoS-enabled paths (e.g. RSVP mechanisms [12] for IntServ and per-hop-behavior code-points for DiffServ). Assuming less than universal deployment of these mechanisms, it must be possible to forward QoS service-dependent-traffic along paths providing the mechanisms that implement the service. Traffic engineering services provide this ability to compute specific service-enabled routes, and to ensure that service-dependent traffic is forwarded along appropriate paths. Additionally, as will be discussed in detail in Section 3, the cost of computing routes that satisfy QoS constraints can potentially be exponential in size of a network, and is ultimately determined by the virtual topology induced by the current link metrics. Since these metrics, reflecting existing link conditions, are not subject to control, the cost of QoS routing computation is also not controllable. Section 4 shows how traffic engineering mechanisms provide a means of controlling these costs by allowing administrative control of the virtual topology induced by the current link states of a network.

As discussed in Section 1, the current prune-and-route traffic engineering routing solutions have poor average case performance, and implement an inflexible traffic-engineering model. The cost incurred by current solutions is to perform a standard shortest path routing computation for each possible value of the concave metric. Since the range of concave metrics can easily be exponential in the size of

the graph, these solutions have high best-case computation costs. Section 3 presents solutions to routing in the context of concave metrics that grow only with the average number of paths *actually existing* to the nodes in a graph. Furthermore, Section 4 presents new approximation solutions that contain this cost even in those instances where this aver number of paths is large.

The inflexibility of current solutions derives from their use of a traffic engineering model which is *imperative* in nature, requiring in effect the programming of an internet with *how* to implement the desired forwarding policies. Specifically, the current model requires the pre-definition of a set of forwarding classes which capture the policy-significant traffic in an internet. The number of forwarding classes must be minimized to control the cost of the prune-and-route style computation described above. These forwarding class definitions must then be installed in traffic classifiers at the edge of the internet. Lastly, the appropriate links in the internet must be labeled with the correct forwarding class sets to implement the desired policies. This "program-the-network" style of traffic engineering is labor intensive and error-prone.

In the following, we propose a *declarative* traffic engineering model where network links are labeled with statements declaring *what* the desired routing policies are in the form of constraints of the traffic allowed on each link. These constraints take the form of expressions, called *link-predicates*, in a boolean *traffic algebra* which describe the traffic allowed on a link. New, efficient policy-based routing algorithms then compute a minimal set of routes, composed of a *path predicate* and a next hop, for each destination in an internet. These algorithms, in effect, *discover* the optimal

set of forwarding classes needed at a given source in the internet to implement the desired policies. These path predicates are then installed in the appropriate traffic classifiers.

The traffic algebra is a boolean algebra used to define traffic classes in a flexible and efficient way. Specifically, it is composed of the standard boolean operations on the set $\{0, 1\}$, where p primitive propositions (variables) are true/false statements describing characteristics of network traffic. The syntax for expressions in the algebra is specified by the BNF grammar:

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

In terms of the truth-table based semantics of a Boolean algebra, expressions in this traffic algebra specify the set of rows in the truth table composed of all the variables in a given expression that evaluate to 1 (or true). From a set theoretic perspective, these expressions can then be interpreted as this set of true rows, and each row as a description of a traffic class. Therefore, each expression can be interpreted as identifying a set of traffic classes with the Boolean operators acting as set operations (e.g. $\wedge$ has the effect of set intersection, $\vee$ has the effect of set union, etc.). When assigned to a link in a network, expressions in the traffic algebra are called *link predicates*. Link predicates identify the traffic classes allowed to traverse the link, and are denoted by ε_{ij} in the algorithms. *Path predicates*, which are denoted by ε_p in the algorithms, and defined as $\varepsilon_p = \varepsilon_{x_1x_2} \wedge \varepsilon_{x_2x_3} \wedge \ldots \wedge \varepsilon_{x_{d-1}x_d}$, specify the set of traffic classes allowed to traverse the path. There is a maximum of 2^p unique sets of administrative constraints.

The $SAT(\varphi)$ primitive of the traffic algebra is the satisfiability problem of

traditional Boolean algebra. SAT answers the question "is there an assignment of truth values to the propositional variables in φ such that φ evaluates to **true**?" The $SAT(\varphi)$ primitive evaluates to 1 (**true**) if such a truth assignment exists, and 0 (**false**) otherwise. Satisfiability must be tested in two situations by algorithms presented in Section 3 that implement traffic-engineering computations. First, when a new route to a destination is considered for comparison to an existing route for the same destination (e.g. lines 8 and 12 in Figure 3.29), they should only be compared if classes of traffic exist that can use either route. Therefore, new routes are only compared with existing routes when the conjunction of their path predicates is satisfiable. Second, given that classes of traffic exist that can use either path, the algorithms must determine whether all traffic supported by one path could use the other. This is the case if the path predicate for one path implies ("$\rightarrow$") the other or, more precisely, if the expression $(\varepsilon_x \rightarrow \varepsilon_y)$ is always true (i.e. is *valid*). Determining if an expression is valid is equivalent to determining if the negation of the expression is unsatisfiable. Therefore the expressions at lines 9 and 13, of the form $\varepsilon_1 \rightarrow \varepsilon_2$ are equivalent to $\neg SAT(\neg(\varepsilon_1 \rightarrow \varepsilon_2))$ (or $\neg SAT(\varepsilon_1 \wedge \neg\varepsilon_2)$). The satisfiability decision performed by $SAT(\varepsilon)$ is the prototypical NP-complete problem [36]. As is typical with NP-complete problems, it has many restricted versions that are computable in polynomial time. Section 5 analyzes the complexity of the satisfiability problem in the context of this algorithm, and identifies a number of implementation strategies that offer the promise of efficient solutions for this application.

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

defined in terms of any globally significant attributes of the ingress router's state that can be expressed as a true/false statement. Care must be taken in defining the set of primitive propositions. The definition of these propositions must be powerful enough to express all expected administrative policies, must be known by all traffic classifiers, and has a strong influence on the syntax required of link predicates. Since, as is analyzed in detail in Section 5, the computational complexity of traffic expression processing is largely determined by the richness of the link predicate syntax, the requirements of this syntax must be kept to a minimum. Therefore the selection of the set of supported primitive propositions is a balancing act between being rich enough to support the desired policies, while containing the computational complexity of the link predicate syntax required to support the desired expressiveness.

A natural, and very general example is a set of primitive propositions describing attributes of the traffic being forwarded. For example, a value of 6 in the protocol field of an IP header is represented by **IPProto(TCP)**; the value 53 in the port field of a UDP header is represented as **UDPPort(DOMAIN)**; and the value of all addresses in the UC Santa Cruz address range in the destination field of an IP header is represented by **IPDest(128.114.0.0/16)**. For example, given the two expressions:

$$\begin{aligned}
\varepsilon_{12} &= \textbf{IPSrc(128.114.0.0/16)} \wedge \textbf{IPProto(TCP)} \\
\varepsilon_{23} &= \textbf{IPSrc(128.114.48.0/24)} \wedge \\
&\quad (\textbf{IPProto(UDP)} \wedge \textbf{UDPPort(DOMAIN)}) \vee \\
&\quad (\textbf{IPProto(TCP)} \wedge \textbf{TCPPort(DOMAIN)})
\end{aligned}$$

the conjunction of these expressions evaluates to:

$$\begin{aligned}
\varepsilon_{13} = \varepsilon_{12} \wedge \varepsilon_{23} &= \textbf{IPSrc(128.114.48.0/24)} \wedge \textbf{IPProto(TCP)} \wedge \\
&\quad \textbf{TCPPort(DOMAIN)}
\end{aligned}$$

which succinctly describes the traffic allowed on a path composed of two links labeled with the expressions ε_{12} and ε_{23}.

2.3 Performance Constraints for QoS

As discussed in Section 1, the space and time complexity of current QoS-based routing algorithms only allow their use for exceptional, typically on-demand path computations. In addition, the approximation solutions that have been developed to address the performance problems with optimal solutions only work with performance-related, QoS metrics and not with administrative, traffic-engineering metrics. To comprehensively address the problems described above, QoS-based routing must be usable as the default Internet routing and forwarding model, and work in coordination with traffic engineering mechanisms. To address these problems, this dissertation presents a family of optimal routing algorithms that support routing in the presence of traffic engineering, QoS, and combined traffic engineering and QoS requirements of an internet.

The routing algorithms presented here are based on an enhanced version of the path algebra defined by Sobrinho [79] that 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. $\oplus$ is the original path composition operator, and $\preceq$ is

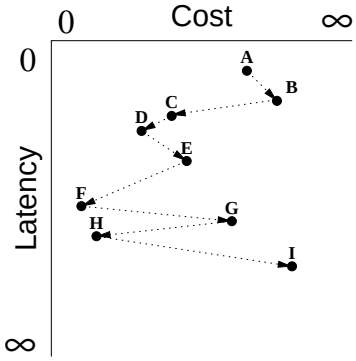

Figure 2.2: $\preceq$ relation

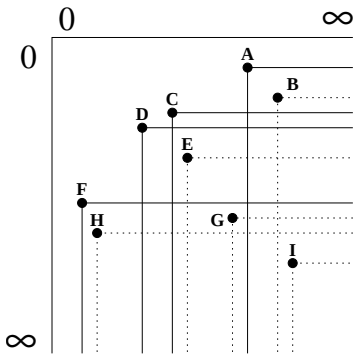

Figure 2.3: $\sqsubseteq$ relation

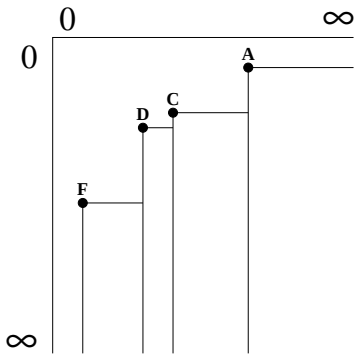

Figure 2.4: Forwarding table

the original total ordering from [79]. $\oplus$ is used to compute path weights from link weights. $\preceq$ is used by the routing algorithm 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, $\sqsubseteq$, 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. $\sqsubseteq$ 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 $s \in R - \{r\}$, $r \not\sqsubseteq s$. 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 latency and cost is

as follows:

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

Figure 2.2 is a graphical depiction of the $\preceq$ relation on a set of weights for routes (labeled A through I) to a given destination in an internet where the arrows are interpreted as $< tail\ of\ arrow > \preceq < head\ of\ arrow >$. Figure 2.3 illustrates the $\sqsubseteq$ relation where each route is represented as a subset of the plane with upper 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 $\sqsubseteq$ relation 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 2.3) contains 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 satisfy some set of constraint pairs. The $\preceq$ relation is used to select one of the set of satisfying routes for a given constraint [56]. As defined

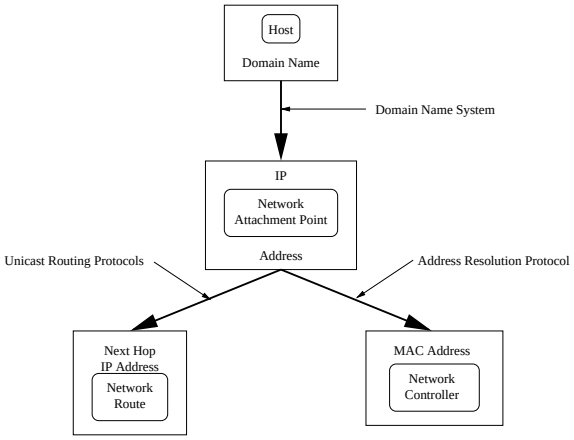

Figure 2.5: Naming Requirements of Unicast Internet Communication

in this example, the $\preceq$ relation has the affect 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 2.4). As a result, forwarding table lookups in this example involve choosing the lowest latency route with acceptable cost.

2.4 Forwarding Mechanisms

The policy-based routing algorithms presented in this dissertation compute multiple routes to the same destination to satisfy the policy requirements of an internet. Such routes are not supported by current, host-address-based packet forwarding mechanisms which only allow one route per destination. The underlying source of this limitation is the restricted naming requirements supported by the current forwarding mechanisms. Early work by Shoch [75] attempted to clarify the naming requirements of computer communication by defining the following framework:

> The name of a resource indicates what we seek;
> an address indicates where it is, and

a route tells us how to get there.

Subsequent work by Saltzer [67] proposed a more general framework in which there may be one or more forms of names for a given type of object, and the naming requirements of a communication system are defined by the possible bindings between the different types of objects in a system. Applying Saltzer's naming framework to standard Internet unicast communication, four types of objects can be identified: hosts, network attachment points, network controllers, and network routes. The names for these objects would then be Internet domain names, IP addresses, MAC addresses (e.g., Ethernet addresses), and next hop IP addresses, respectively. Following Saltzer's model, the naming requirements of this Internet unicast communication system can now be defined by the bindings that must be maintained among these four types of objects (see Figure 2.5).

Focusing on the binding between network attachment points and network routes we can see the source of this limitation of one route per destination is the use of IP addresses in the mapping of a destination network attachment point to a network route. Forcing the key to the mapping to be an IP address forces the one-to-one mapping between destinations and routes identified above. The solution to this problem is to use label-swapping technology (e.g. MPLS [24]) as a generalized forwarding mechanism which replaces IP addresses as the names for network attachment points in the route binding function with arbitrary labels which can be defined by the routing protocol to represent any policy/destination pair for which a route has been computed.

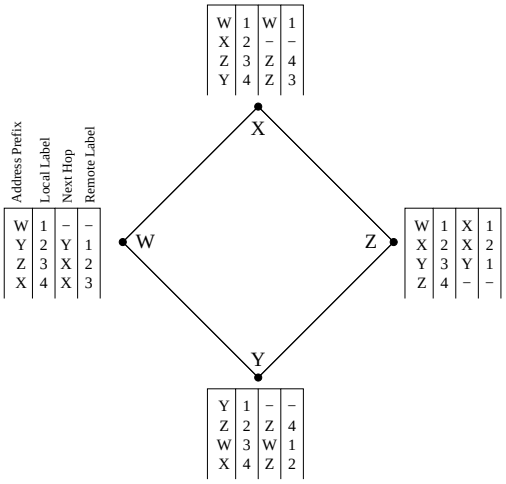

Figure 2.6: Labels with Address-Based Forwarding

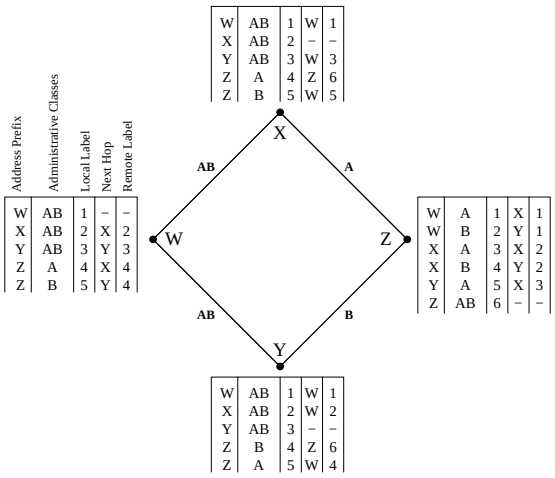

Figure 2.7: Labels with Policy-Based Forwarding

A significant innovation of the policy-based routing architecture presented here is the combination of a table-driven, hop-by-hop routing model with label-swap forwarding mechanisms. Traditionally, label-swap forwarding has only been seen as an appropriate match with an on-demand, source-driven routing model. Indeed, the virtual-circuit nature of these previous solutions has been attributed to their use of label-swap forwarding. Contrary to this view, the position taken in this work is 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 [81] within the network layer, where address-based forwarding ties the two planes together. This separation provides what might be called a *topological anonymity* of the forwarding plane that is critical to the implementation

of policy-based routes.

As illustrated in Figure 2.6, label-swap forwarding can be used in the context of traditional address-based forwarding. 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 benefit of this mechanism for traffic forwarding is it can be generalized to handle policy-based forwarding. In addition, label-swap forwarding can be used to implement traffic engineering via the assignment of traffic to administrative classes which 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 2.7 shows a small network with four nodes, two administrative classes A and B, and the given forwarding state for reaching node 4. The benefits of this architecture are that it is based on forwarding state that is agnostic to the definition of forwarding classes, allowing the data forwarding plane to remain simple yet general; and it concentrates the path computation functions in the routing protocol, which is the least time critical, and most flexible component of the network layer.

The resulting routing architecture can be seen as analogous to the Reduced Instruction Set Computer (RISC) processor architecture where, as part of the effort to simplify processor designs to allow orders-of-magnitude improvements in the performance of modern processors, researchers shifted much of the intelligence for managing the use of processor resources to the compilers which were able to bring a higher-level perspective to the task that allowed 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, hopefully, allow for significant advances in the performance and effectiveness of network infrastructure.

2.5 Traditional Dijkstra Algorithm

Figure 2.8 presents the traditional Dijkstra algorithm. Section 3 presents a number of enhancements to this algorithm to support different subsets of policy-based routing services. This section presents the traditional algorithm with an analysis of its complexity, and a proof of correctness as a baseline for the following sections to build on.

The Dijkstra algorithm works by maintaining a set T of temporarily assigned routes, and a set P of permanently assigned routes. Each routes is specified by a 3-tuple $< x, p_x, \omega_x >$. ω_x is the path weight currently assigned to node x. For nodes in P, ω_x is the final weight assignment specifying the shortest distance to x. For nodes in T, ω_x is the current best estimate of the shortest distance to x based on routes currently contained in P. p_x is the predecessor to x on the currently selected route with weight ω_x. T_j is the entry in T for node j.

The time complexity of the Dijkstra algorithm is dominated by the loop at

```
algorithm Dijkstra
  begin
1 Push(<s, s, 0>, P);
2 for each {(s, j) ∈ A(s)}
3    Insert(<j, s, ω_sj>, T);
4 while (|T| > 0)
     begin
5    <i, p_i, ω_i> ← Min(T);
6    DeleteMin(T);
7    Push(<i, p_i, ω_i>, P);
8    for each {(i, j) ∈ A(i)}
9       if (T_j = ∅)
10         then Insert(<j, i, ω_i + ω_ij>, T)
11      else if (ω_i + ω_ij < T_j.ω_j)
12         then DecreaseKey(<j, i, ω_i + ω_ij>, T);
     end
  end
```

Figure 2.8: Traditional Dijkstra Shortest-Path Algorithm.

line 4, which is executed at most once for each node in the graph for a total of n times, and the loop at line 7, which is executed at most once for each edge in the graph for a total of m times. The most time consuming operations within these processing blocks are the access to the set T at lines 5 and 9. Assuming the use of a d-heap for representing the set T, these operations require $\log_d n$ time [2]. Therefore, line 5 requires at most $n \log_d n$ time, and line 9 requires at most $m \log_d n$ time, resulting in a worst-case overall time complexity for Dijkstra of $O(m \log_d n)$.

2.5.1 Proof of Correctness

The following proof of correctness of the Dijkstra algorithm is derived from that given in [9].

Lemma 1 *At the beginning of each iteration of the loop at line 4,* $\omega_p \leq \omega_t$ *for all* $<p, p_p, \omega_p> \in P$, *and* $<t, p_t, \omega_t> \in T$.

Proof: By induction. The property is true for the base case of:

$$P = \{< s, s, 0 >\},\ T = \{< x, s, \omega_{sx} > \mid\ (s, x) \in E\}.$$

Assume it is true at the beginning of an iteration, it is also true after the iteration since the node added to P during the iteration is the node with the least weight in T, the weight of all nodes left in T after the iteration may only have been modified by $\omega_j \leftarrow \omega_i + \omega_{ij}$ (in line 10 or 11), and $\omega_{ij} > 0$. ■

Lemma 2 *At the beginning of each iteration of the loop at line 4, for each route* $J = < j, p_j, \omega_j > \in P \cup T$, ω_j *is the least weight to* j *using paths with all nodes except possibly* j *having routes in* P.

Proof: By induction. The property holds for the base case (described in Lemma 1). Assume the property holds at the start of a given iteration. Let $I = < i, p_i, \omega_i >$ be the route added to P in that iteration, and let $K = < k, p_k, \omega_k >$ be the route of each entry in $P \cup T$ at the beginning of that iteration. There are three cases to be considered for the iteration: $J = I$, $J \in P$, and $J \notin P \cup I$. For the case $J = I$ the property holds by the induction hypothesis (ω_i was the least weight from s to i using paths with all routes except I belonging to set P before the iteration, and nothing happens during the iteration to change this). For the case $J \in P$ the property holds by the induction hypothesis (the distances for routes in P were better at the start of the iteration) and Lemma 1 (all ω_k for $K \in T$ stay worse than ω_j in the iteration). For the case $J \notin P \cup I$, consider a path to j which is one of the shortest of all those to j with all nodes except j in $P \cup I$, and let ω'_j be the weight of this path (see Figure 2.9).

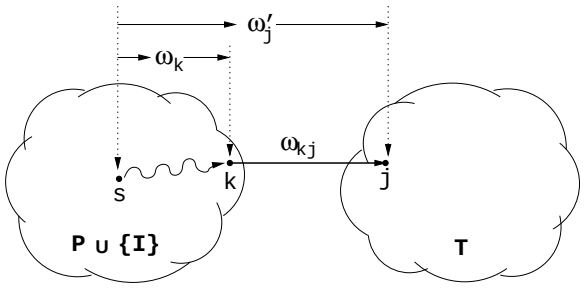

Figure 2.9: Dijkstra Relaxation Step

Such a path must be composed of a shortest path to some k composed of nodes with routes in $P \cup I$, followed by an edge $(k, j) \in E$. From this we have:

$$\begin{aligned} \omega'_j &= \min_{k \in P \cup \{i\}} (\omega_k + \omega_{kj}) \\ &= \min \left[\min_{k \in P} (\omega_k + \omega_{kj}), \omega_i + \omega_{ij} \right] \end{aligned}$$

Since the induction hypothesis implies:

$$\omega_j = \min_{k \in P} (\omega_k + \omega_{kj})$$

we get:

$$\omega'_j = \min \left[\omega_j, \omega_i + \omega_{ij} \right].$$

Thus, in lines 8 and 9, ω_j is set to the least weight ω'_j to j using paths with all nodes except j having routes in $P \cup I$. ■

Theorem 1 *Dijkstra computes the shortest route to each node in N in finite time.*

Proof: Since a route for a new node is added to P in each iteration, the algorithm terminates after n iterations. By Lemma 2 Dijkstra computes routes for the shortest paths to each destination. ■

The important observation to make of the correctness of the Dijkstra algorithm is it is based on three invariants:

1. For all routes $< p, p_p, \omega_p > \in P$ and $< t, p_t, \omega_t > \in T$, $\omega_p < \omega_t$.

2. For all routes $< j, p_j, \omega_j > \in P \cup T$, ω_j is the weight of the shortest path to j using nodes with routes in P.

3. In each iteration, the route moved from P to T is the shortest route in T, and is for a destination that doesn't currently have a route in P.

These invariants are targeted very specifically for a single-constraint routing application. The following slight generalization of the second invariant maintains the correctness of compliant algorithms, but expands the range of algorithm design and implementation options:

2a. For all routes $< j, p_j, \omega_j > \in P$, ω_j is the shortest path to j using nodes with routes in P.

2b. The set of routes $< j, p_j, \omega_j > \in P \cup T$, contains a route whose weight is that of the shortest path to j using nodes with routes in P.

Using these invariants, the algorithm in Figure 2.10 can be defined where all routes discovered in the for loop at line 9 are unconditionally added to the heap T (where P_i denotes the entry for node i in queue P). The loop at line 4 considers the minimum route in T, until a route is found for a destination with no route in P. This would not be an efficient algorithm as it would require a heap large enough to hold all

```
algorithm Generalized-Dijkstra()
    begin
1   Push(<s, s, 0>, P);
2   for each {(s, j) ∈ A(s)}
3      Insert(<j, s, ω_sj>, T);
4   while (|T| > 0)
       begin
5      <i, p_i, ω_i> ← Min(T);
6      DeleteMin(T);
7      if (P_i = ∅)
          then begin
8            Push(<i, p_i, ω_i>, P);
9            for each {(i, j) ∈ A(i)}
10              Insert(<j, i, ω_i + ω_ij>, T);
             end
       end
    end
```

Figure 2.10: Generalized (but Impractical) Dijkstra.

routes discovered for a destination before the permanent route is determined (which is limited to the number of neighbors of the destination). However, as will be shown in the remainder of this dissertation, it provides a good model for efficient solutions to challenging, policy-based routing problems.

Chapter 3

Algorithms for Exact Solutions

The challenge of multi-constrained routing is to find an optimal set of independent routes to each destination in an efficient manner. A generalized set of invariants required for correctness of Dijkstra-like single-constraint algorithms was presented in the previous section which provides a fruitful foundation for multi-constrained algorithms. The appeal of these invariants for use in multi-constrained routing is that they support the presence of multiple routes to the same destination during the computation. Due

1.	For all routes $I \in P$ and $J \in T$, $I \preceq J$.
2.	All routes to $j \in P$ are incomparable for some satisfying truth assignment.
3.	The maximal subset of routes to $j \in P \cup T$ represents the maximal subset of paths to j using nodes with routes in P.
4.	In each iteration, the route moved from T to P is the lightest route in T, and is incomparable with any route for the same destination in P with a shared satisfying truth assignment.

Table 3.1: Invariants Required for Correct Multi-Constrained Routing

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.
H_n	$\equiv$	Heap of best routes on each link for node n.
H_n^k	$\equiv$	Entry in H_n for link (k, n).
Q_n^k	$\equiv$	Queue of routes for node n over link (k, n).
$\mathcal{E}_n$	$\equiv$	Summary of traffic expression for all routes in P_n.
ε_n^k	$\equiv$	Summary of traffic expressions for all routes that have been added to Q_n^k (included those added but subsequently deleted).

Table 3.2: Notation.

to the presence of multiple metrics for each link and path, the generalized invariants for single-metric routing must be enhanced as shown in Table 3.1.

The remainder of this section presents a family of routing algorithms that provide on-demand and table-driven solutions for traffic engineering only, QoS only, and combined traffic engineering and QoS routing, simulation results from these algorithms, and a review of previous solutions for computing optimal, policy-based routes. The notation used in the algorithms presented in the following is summarized in Table 3.2. In addition, the maximum number of unique truth assignments is denoted by $A = 2^p$ (see Section 2.2), the maximum number of unique weights by $W = \min(\textit{range of weight components})$ (see Section 2.3), and the maximum number of adjacent neighbors by $a_{max} = \max\{|\, A(i) \,|\ |\ i \in N\}$. Table 3.3 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. Note that d-Heaps are assumed in the remainder of this dissertation rather than Fibonacci-Heaps

Notation	Description	Complexity
	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	$O(\log(n))$

Table 3.3: Operations on Data Structures [2].

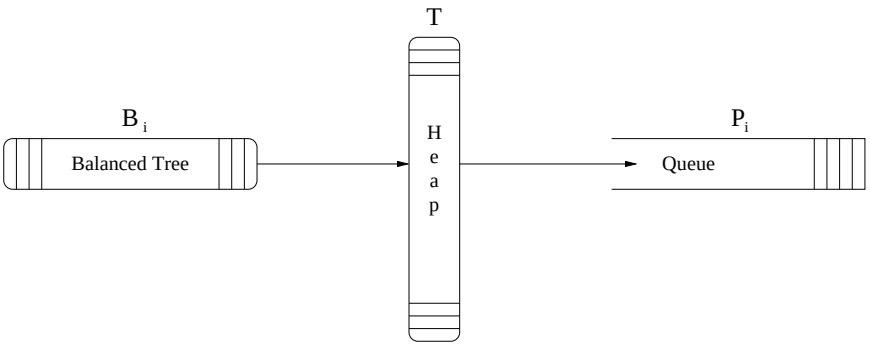

Figure 3.1: Model of Data structures for Basic Algorithms

in spite of the better asymptotic behavior of the latter. This choice was made due to the simplicity of the d-Heap algorithms (allowing for simpler implementation), and the fact, due to the large constant factors associated with Fibonacci heap's run-time, that they are actually less efficient when applied to the scale of problem expected for these algorithms.

The algorithms presented in this section apply the algorithmic model presented in Figure 2.10 to the data structure model shown in Figure 3.1. 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). 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 these algorithms will be to take the minimum entry from the heap T, compare it with existing routes in the appropriate P_i, if it is incomparable by 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.

As will be detailed below, the runtime complexity of these algorithms all have a $\log F$ factor (where F denotes the maximum number of forwarding classes, whose components depend on the algorithm). This factor reflects the cost of maintaining B_i, which may grow to contain $a_{max}F$ elements. The use of balanced trees for B_i is driven by the requirements that finding the minimum element of B_i must be inexpensive, that no assumptions can be made of the order in which routes are added to a given B_i, and that the maximum size of B_i ($a_{max}F$) is too large (as well as unlikely to occur) to justify the use of a heap (which requires a fixed maximum size). Section 3.7 presents enhanced versions of the algorithms in this section that exploit the fact that routes to a given node *with the same predecessor* are discovered in increasing (or non-decreasing, depending on the algorithm) order. This ordering is used to reduce the binary tree data structures to more intricate, but more efficient queue-based data structures with the result that the maintenance of these structures becomes a lower order term in the complexity of the algorithms.

3.1 On-Demand Traffic Engineering

As a demonstration of the traffic algebra, a modified version of the Dijkstra algorithm that supports the on-demand computation of routes subject to link-predicate constraints is given in Figure 3.2. In this algorithm a request R is being processed, typically triggered by the receipt of the first packet of a flow, with the set of truth assignments to the primitive propositions given by R_τ. This algorithm works by evaluating the link predicate of every candidate link on the given truth assignment,

```
algorithm OD-TE-Dijkstra
  begin
1 Push(<s, s, 0>, P);
2 for each {(s, j) ∈ A(s)}
3    Insert(<j, s, ω_sj>, T);
4 while (|T| > 0)
     begin
5    <i_p, i, ω_i> ← Min(T);
6    DeleteMin(T);
7    Push(<i, p_i, ω_i>, P);
8    for each {(i, j) ∈ A(i) | ε̂_ij(R_τ)}
9       if (T_j = ∅)
10         then Insert(<j, i, ω_i + ω_ij)
11      else if (T_j.ω_j > ω_i + ω_ij)
12         then DecreaseKey(<j, i, ω_i + ω_ij >, T);
     end
  end
```

Figure 3.2: On-Demand, Traffic Engineering Dijkstra.

specified by $\widehat{\varepsilon}_{ij}(R_\tau)$, and processing only those links whose link predicate evaluates to 1 (or true).

This algorithm works by running the traditional Dijkstra algorithm on the subset of the graph containing links whose predicates are satisfied by the request's truth assignment (R_τ). This is, in effect, an in-lined version of the prune preprocessing solution discussed in Section 1. The satisfying sub-graph is "discovered" as the algorithm progresses.

Implementing, in effect, the Dijkstra algorithm on a subset of the graph, ignoring the cost for traffic algebra expression evaluations (which will be analyzed in depth in Section 5), the time complexity of this algorithm is the same as the traditional Dijkstra algorithm (i.e. $m \log_d n$). Similarly, the correctness derives from the correctness of the Dijkstra algorithm which was proven in Section 2.5.1.

```
algorithm TD-TE-Dijkstra()
    begin
1   Push(<s, s, 0, 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    if (¬(ε_i → E_i))
        then begin
11        Push(<i, p_i, ω_i, ε_i>, P_i);
12        E_i ← E_i ∨ ε_i;
13        for each {(i, j) ∈ A(i) | SAT(ε_i ∧ ε_ij) ∧ ¬((ε_i ∧ ε_ij) → E_j)}
            begin
14          ω_j ← ω_i + ω_ij; ε_j ← ε_i ∧ ε_ij;
15          if (T_j = ∅)
16            then Insert(<j, i, ω_j, ε_j>, T)
17          else if (ω_j < T_j.ω)
18            then DecreaseKey(<j, i, ω_j, ε_j>, T);
19          Insert(<j, i, ω_j, ε_j>, B_j);
            end
        end
    end
```

Figure 3.3: Basic, Table-Driven, Traffic-Engineering Dijkstra.

3.2 Table-Driven Traffic Engineering

Figure 3.3 presents a further enhancement to Dijkstra that precomputes all optimal routes to all destinations in an internet in the presence of administrative constraints on the links in the network. 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. Similar to the on-demand version, the virtual graph is "discovered" as needed by the algorithm as the computation progresses.

Similar to Dijkstra, TD-TE-Dijkstra works by maintaining a temporarily labeled set of routes, T, and a queue of permanently labeled set of routes per destination, P_d. Each route is specified by a four-tuple $< d, p_d, \omega_d, \varepsilon_d >$. For routes in P_x, ω_x is the final weight assignment specifying the shortest distance to x for traffic satisfying the path expression ε_x. For routes in T, ω_x is the current best estimate of this distance based on routes currently contained in P. p_x is the predecessor to x for the route. Similar to the Generalized-Dijkstra algorithm (Figure 2.10), TD-TE-Dijkstra proceeds using the typical Dijkstra iteration over the n^{th} closest node with the difference that, as new routes are discovered, they are inserted in the heap T if there is some class of traffic that can satisfy the path expression (the $SAT()$ test at line 13), and at the top of each iteration routes are pulled from T and discarded until one is found that is unique in P (in the sense that it includes a traffic class for which no route currently exists in P_i), which is then added to P.

Ignoring the cost for the satisfiability problem (which will be analyzed in detail in Section 5), the time complexity of the TD-TE-Dijkstra algorithm is dominated by the loops at lines 4 and 13. The loop at line 4 is executed at most once for each incomparable path (in terms of path predicates) to each node in the graph for a total of nA times. The loop at line 13 is executed at most once for each distinct instance of an edge in the graph, for a total of mA 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 $\log a_{max} A$, and an aggregate cost of $nA \log a_{max} A$. 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 $mA \log n$. For the loop at line 13, the most time consuming operation is the addition to the balanced tree B_i at line 19 with a per-operation cost of $\log a_{max} A$, and an aggregate cost of $mA \log a_{max} A$. Therefore, the worst case time complexity of TD-TE-Dijkstra, dominated by the operation at line 19, is $O(mA \log A)$.

3.2.1 Proof of Correctness

The correctness of this algorithm is based on the generalized invariants (Table 3.1). An optimization is included at line 13 where traffic classes supported by new routes are tested against those supported by routes existing in P_i before being included in the set of temporary labeled routes. In the following, Lemma 3 shows that routes in P have a smaller weight than those in T; Lemma 4 shows that each route in P is uniquely best for some truth assignments using routes in P; and Lemma 5 shows that there is a route in $P \cup T$ for every shortest path with satisfying truth assignments using nodes with routes in P.

Lemma 3 *At the beginning of each iteration of the loop at line 4,* $\omega_p \leq \omega_t$ *for all* $< p, p_p, \omega_p, \varepsilon_p > \in P$ *and* $< t, p_t, \omega_t, \varepsilon_t > \in T$.

Proof: By induction. The property holds for the base case (at the start of the first iteration) of:

$$P = \{< s, s, 0, 1 >\},\ T = \{< x, s, \omega_{sx}, \varepsilon_{sx} > \mid\ (s, x) \in E\}.$$

Assume it holds at the beginning of an iteration, it is also true after the iteration since the route $I =< i, p_i, \omega_i, \varepsilon_i >$ added to P during the iteration is the route with the

smallest distance in T, the distance of all routes $N =< n, p_n, \omega_n, \varepsilon_n >$ left in T after the iteration may only have been modified by $\omega_j \leftarrow \omega_i + \omega_{ij}$ (line 14), and $\omega_{ij} > 0$. ■

Lemma 4 *Each route $J = < j, p_j, \omega_j, \varepsilon_j > \in P$ is the only route in P with the least weight to j for some set of satisfying truth assignments to ε_j using paths with nodes having routes in P satisfied by the set of truth assignments.*

Proof: By induction. The property holds for the base case. Assume it holds at the start of some iteration. Let $I = < i, p_i, \omega_i, \varepsilon_i >$ be the route added to P in that iteration. The property holds following the iteration because: by Lemma 3, all routes in P are less than I; by the induction hypothesis, all routes in P are best for some truth assignments; and, since I is the smallest entry in T (line 5), and I is only added to P (line 11) if there are some satisfying truth assignments for ε_i that are not satisfied by routes in P (line 10), I is uniquely best among the route in $P \cup I$ for those satisfying truth assignments to $\varepsilon_i \wedge \neg \mathcal{E}_i$. ■

Lemma 5 *For every path using nodes with routes in $P \cup T$ that is shortest for a truth assignment τ, there exists a route $J = < j, p_j, \omega_j, \varepsilon_j > \in P \cup T$ such that τ satisfies ε_j and ω_j is the weight of the path.*

Proof: By induction. The property holds for the base case. Assume the property holds at the start of a given iteration. Let $I =< i, p_i, \omega_i, \varepsilon_i >$ be the route added to P in that iteration, $K =< k, p_k, \omega_k, \varepsilon_k >$ for each route in $P \cup T$ at the beginning of that iteration, and $N =< j, p_j, \omega_j, \varepsilon_j >$ be the new route possibly added to T in that iteration (lines 14–19). There are three cases to be considered at the start of the

iteration: $J = I, J \in P$, and $J \notin P \cup \{I\}$. For the case $J = I$ the property holds after the iteration by the induction hypothesis and the fact, since $\omega_{in} > 0$, that $\omega_i < \omega_n$ (line 14). For the case $J \in P$ the property holds after the iteration by the induction hypothesis and Lemma 3. For the case $J \notin P \cup \{I\}$, consider a path to j which is one of the shortest of all those to j composed of routes in $P \cup \{I\}$ where τ satisfies the path predicate; let ω'_j be the weight of this path, and ε'_j be its path predicate (see Figure 2.9). Such a path must be composed of a shortest path to some k with a route in $P \cup \{I\}$, followed by an edge $(k, j) \in E$ such that $\widehat{\varepsilon_k}(\tau) \wedge \widehat{\varepsilon_{kj}}(\tau)$. From this we have:

$$\begin{aligned}
\omega'_j &= \min_{k \in P \cup \{I\} | \widehat{\varepsilon_k}(\tau) \wedge \widehat{\varepsilon_{kj}}(\tau)} (\omega_k + \omega_{kj}) \\
&= \min \left[\min_{k \in P | \widehat{\varepsilon_k}(\tau) \wedge \widehat{\varepsilon_{kj}}(\tau)} (\omega_k + \omega_{kj}), \begin{cases} \omega_i + \omega_{ij} & \text{if } \widehat{\varepsilon_i}(\tau) \wedge \widehat{\varepsilon_{ij}}(\tau) \\ \infty & \text{otherwise} \end{cases} \right]
\end{aligned}$$

Since Lemma 4 implies

$$\omega_j = \min_{k \in P | \widehat{\varepsilon_k}(\tau) \wedge \widehat{\varepsilon_{kj}}(\tau)} (\omega_k + \omega_{kj})$$

we get

$$\omega'_j = \min \left[\omega_j, \begin{cases} \omega_i + \omega_{ij} & \text{if } \widehat{\varepsilon_i}(\tau) \wedge \widehat{\varepsilon_{ij}}(\tau) \\ \infty & \text{otherwise} \end{cases} \right].$$

Thus, lines 14–19 ensure that N is only added to T if ω_n is the least weight ω'_j to j using paths with all nodes x except possibly j having routes in P where $\widehat{\varepsilon_n}(\tau) \wedge \widehat{\varepsilon_x}(\tau)$ is true. Furthermore, since no routes for a given destination are deleted from T (lines 6–9) until it is the least weight route for the destination, the property holds at the end of the iteration. ■

```
algorithm OD-TE-CM-Dijkstra
  begin
1 Push(<s, s, 0̄>, P);
2 for each {(s, j) ∈ A(s)}
3    Insert(<j, s, ω_sj>, T);
4 while (|T| > 0)
    begin
5    <i_p, i, ω_i> ← Min(T);
6    DeleteMin(T);
7    Push(<i, p_i, ω_i>, P);
8    for each {(i, j) ∈ A(i) | ε̂_ij(R_τ) ∧ (R_c ⪯ c_ij)}
9       if (T_j = ∅)
10         then Insert(<j, i, ω_i ⊕ ω_ij)
11      else if (ω_i ⊕ ω_ij ≺ T_j.ω_j)
12         then DecreaseKey(<j, i, ω_i ⊕ ω_ij>, T);
    end
  end
```

Figure 3.4: On-Demand, Traffic Engineering with Concave QoS.

Theorem 2 *TD-TE-Dijkstra computes a set of shortest routes to each reachable node in N for every truth assignment with a path to the node in G.*

Proof: Since the number of routes in the set of routes induced by G is finite, no duplicate routes are added to T, and a new route is added to P in each iteration, the algorithm terminates after a finite number of iterations. By Lemma 4, every route in P is a uniquely best route for some set of truth assignments. By Lemma 5, the set of routes in $P \cup T$ contains routes for all shortest paths using nods with routes in P. Therefore, the algorithm terminates with $T = \emptyset$, and P containing a route for the shortest path to every reachable node in N for every truth assignment for which a path exists. ■

3.3 On-Demand Traffic Engineering and Concave QoS

Figure 3.4 presents a modified version of the OD-TE-Dijkstra algorithm (Figure 3.2), enhanced to support concave QoS link metrics. The OD-TE-CM-Dijkstra algorithm performs such computations where the request R also includes a set of concave link metric constraints (R_c), and each link is labeled with a QoS subset, c_{ij}, of the concave link metrics, ω_{ij} (i.e. $c_{ij} \subseteq \omega_{ij}$). The algorithm works by testing the constraints for every candidate link, and only processing those links whose metrics satisfy the constraint. This is, in effect, an in-lined version of the prune preprocessing solution mentioned discussed in Section 1. The satisfying sub-graph is "discovered" as the algorithm progresses.

Implementing, in effect, the Dijkstra algorithm on a subset of the graph, ignoring the cost for traffic algebra expression evaluations (which will be analyzed in depth in Section 5), the time complexity of this algorithm is the same as the traditional Dijkstra algorithm ($m \log_d n$). Similarly, the correctness derives from the correctness of the Dijkstra algorithm which was proven in Section 2.5.1.

3.4 Table-Driven General QoS

Figure 3.5 presents a modified Dijkstra algorithm that computes an optimal set of routes to each destination subject to multiple general (additive or concave) path metrics. An *optimal set of routes* for a graph is a subset R_o of the routes in the graph such that for any requested set of metric constraints ω_r to a destination d, the shortest

```
algorithm TD-QoS-Dijkstra
    begin
1   Push(<s, s, 0̄>, P_s);
2   for each {(s, j) ∈ A(s)}
3      Insert(<j, s, ω_sj>, T);
4   while (|T| > 0)
       begin
5      <i, p_i, ω_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     if (ω_i ⋤ Tail(P_i).ω)
          then begin
11           Push(<i, p_i, ω_i>, P_i);
12           for each {(i, j) ∈ A(i) | ω_i ⊕ ω_ij ⋤ Tail(P_i).ω}
                begin
13              ω_j ← ω_i ⊕ ω_ij;
14              if (T_j = ∅)
15                 then Insert(<j, i, ω_j>, T)
16              else if (ω_j ≺ T_j.ω)
17                 then DecreaseKey(<j, i, ω_j>, T);
18              Insert(<j, i, ω_j>, B_j);
                end
             end
       end
    end
```

Figure 3.5: Table-Driven, QoS Dijkstra.

route $< d, p_s, \omega_s >$ to d in R_o such that $\omega_r \sqsubseteq \omega_s$ is a shortest path in the graph with path metrics satisfying the request constraints. The following theorem defines the goal of the TD-QoS-Dijkstra in terms of the path algebra presented above.

Theorem 3 *Any maximal subset of the set of routes R induced by a given G, is an optimal set of routes for the network modeled by G.*

Proof: By contradiction. Given a route request ω_r for destination d, let $R_{best} = < d, p_b, \omega_b >$ be the smallest route from a maximal set of routes for the graph where $\omega_r \sqsubseteq \omega_b$. Also assume the route is not the shortest path to d satisfying ω_r. There must exist a route $R_{shorter} = < d, p_s, \omega_s >$ that is not a maximal route for the graph

where $\omega_r \sqsubseteq \omega_s$. This implies that $\omega_r \succeq \omega_b \succeq \omega_s$. This either implies that $\omega_b \sqsubseteq \omega_s$, which contradicts the fact that R_{best} is a maximal route for the graph, or that there exists another maximal route $R'_{best} = < d, p'_b, \omega'_b >$ where $\omega_s \sqsubseteq \omega'_b$ (since $R_{shorter}$ is not a maximal route), which implies that $\omega'_b \preceq \omega_s \preceq \omega_b \preceq \omega_r$, which contradicts the fact that R_{best} is the smallest maximal route $\preceq \omega_r$. ■

Therefore, the route computation involves the computation of a maximal set of routes for the input graph, and the route lookup function involves finding the smallest route from this set such that the request is $\sqsubseteq$ the maximal route.

The complexity of TD-QoS-Dijkstra is of the same form as that for TD-TE-Dijkstra (Section 3.2), except for the source of the limit on the maximum number of incomparable paths to a destination. For TD-QoS-Dijkstra the maximum number of incomparable paths (in terms of incomparable path weight values) paths to a given node in the graph is limited by the number of unique path weight values, W. Similar to TD-TE-Dijkstra, ignoring the cost for the satisfiability problem, the time complexity of the TD-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 TD-TE-Dijkstra, dominated by the operation at line 18, is $O(mW \log W)$.

TD-QoS-Dijkstra performs the same computation as Jaffe's pseudopolynomial time algorithm for the MCP problem [42] using only worst case $O(nW)$ space complexity compared with guaranteed space complexity of $O(n^2b)$ (where b is the largest metric value) of Jaffe's algorithm. The space savings comes from directly computing the maximal set of routes. From this set of routes, all n^2b routes computed by Jaffe's algorithm can be inferred with minimal overhead.

3.4.1 Proof of Correctness

Lemma 6 *At the beginning of each iteration of the loop at line 3, for all pairs of routes to to a given destination j, $<j, p_p, \omega_p> \in P$ and $<j, p_t, \omega_t> \in T$, $\omega_p \prec \omega_t$.*

Proof: By induction. The property holds for the base case (at the start of the first iteration) of:

$$P = \{<s, s, \overline{0}>\},\ T = \{<x, s, \omega_{sx}> \mid\ (s, x)\ \in E\}.$$

Assume it holds at the beginning of an iteration. Let i be the destination for which a route is moved from T to P in the iteration. There are two cases to be considered for each iteration: $j = i$ and $j \neq i$. For the case $j = i$ the property holds by the

induction hypothesis, the fact that the lightest weight route to j in T is moved to P in the iteration, and the fact, since no self-loops (i.e. edges of the form (j, j)) exist in the graph), that no routes to j are modified in T during the iteration. For the case $j \neq i$ the property holds by the induction hypothesis, and the fact, since all routes to i left in T after the iteration may only have been modified by $\omega_i \leftarrow \omega_j \oplus \omega_{ji}$ (at line 13), that all routes to i in T must still be greater than any route in P. ■

Lemma 7 *All routes* $J = < j, p_j, \omega_j > \in P_j$ *are incomparable.*

Proof: By induction. The property holds for the base case. Assume the property holds at the start of some iteration. Let $I = < i, p_i, \omega_i >$ be the route considered for addition to P_j in that iteration. By Lemma 6, $\omega_j \prec \omega_i$, and therefore, by Property 1, $\omega_j \not\sqsupseteq \omega_i$. Furthermore, $\omega_i \not\sqsupseteq \omega_j$ by the test at line 10. Therefore, if I is added to P_j at line 11, I is incomparable with all routes in P_j, and the property still holds following the iteration. ■

Lemma 8 *For any* $j \in N$, *the maximal set of routes to* j *using paths composed of nodes with routes in* P *is a subset of the set of routes* $\{< j, p_j, \omega_j > \in P \cup T\}$.

Proof: By induction. The property holds for the base case. Assume the property holds at the start of some iteration. Let $I = < i, p_i, \omega_i >$ be the route considered for addition to P at line 11. I is not added to P only if it is comparable to a route currently in P (line 10). Similarly, relaxed routes are not added to T (lines 14–18) only if they are comparable to a route in P (line 12). Therefore, all potentially incomparable routes are kept in P or T in each iteration, and the property holds after the iteration.

■

Theorem 4 *TD-QoS-Dijkstra computes the maximal set of routes to each node in N in finite time.*

Proof: Since the number of routes in the set of routes induced by G is finite, and a new route is added to P in each iteration, the algorithm terminates after a finite number of iterations. Therefore, when the algorithm terminates T is empty. By Lemma 7 all routes in P are incomparable. Furthermore, by Lemma 8, the maximal set of routes for G is a subset of P. ■

3.5 Table-Driven Traffic Engineering and General QoS

Figure 3.6 presents a modified Dijkstra algorithm that computes an optimal set of routes to each destination subject to multiple general (additive or concave) path metrics, in the presence of administrative constraints on the links. Ignoring the costs for the satisfiability problem (which will be analyzed in Chapter 5, the time complexity of Policy-Based-Dijkstra is dominated by the loops at lines 4, 11, and 16. Similar to the complexity of the previous algorithms, the loop at line 4 is executed nWA times, and the loop at line 16 mWA times. The loop at line 11, which is new in this algorithm, 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

```
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      if (ω_i ⊑ ptr.ω)
13      ε_tmp ← ε_tmp ∧ ¬ptr.ε; ptr ← ptr.next;
14    if (ε_tmp ≠ 0)
        then begin
15        Push(<i, p_i, ω_i, ε_i>, P_i);
16        for each {(i, j) ∈ A(i) | SAT(ε_tmp ∧ ε_ij)}
            begin
17          ω_j ← ω_i ⊕ ω_ij; ε_j ← ε_tmp ∧ ε_ij;
18          if (T_j = ∅)
19            then Insert(<j, i, ω_j, ε_j>, T)
20          else if (ω_j ≺ T_j.ω)
21            then DecreaseKey(<j, i, ω_j, ε_j>, T);
22          Insert(<j, i, ω_j, ε_j>, B_j);
            end
        end
    end
```

Figure 3.6: General-Policy-Based Dijkstra.

to a given destination) for a total of

$$\sum_{i=1}^{WA-1} i = \frac{(WA-1)WA}{2}$$

times. For completeness, the statements at lines 6 and 22 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 18–20 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 at line 11, is $O(nW^2A^2)$.

3.5.1 Proof of Correctness

Lemma 9 *At the beginning of each iteration of the loop at line 3, for all pairs of routes to to a given destination j, $< j, \varepsilon_p, \omega_p, p_p > \in P$ and $< j, \varepsilon_t, \omega_t, p_t > \in T$ where $SAT(\varepsilon_p \wedge \varepsilon_t)$, $\omega_p \prec \omega_t$.*

Proof: By induction. The property is true for the base case (following the first iteration) of:

$$P = \{< s, 1, \overline{0}, s >\},\ T = \{< x, \varepsilon_{sx}, \omega_{sx}, s > \mid\ (s, x)\ \in E\}.$$

Assume it is true at the beginning of an iteration. Let i be the destination for which a route is moved from T to P in the iteration. There are two cases to be considered for each iteration: $j = i$ and $j \neq i$. For the case $j = i$ the property holds by the induction hypothesis, the fact that the lightest weight route to j in T is moved to P in the iteration, and the fact, since no self-loops (i.e. edges of the form (j, j)) exist in the graph), that no routes to j are modified in T during the iteration. For the case $j \neq i$ the property holds by the induction hypothesis, and the fact, since all routes to i left in T after the iteration may only have been modified by $\omega_i \leftarrow \omega_j \oplus \omega_{ji}$ (at line 16), that all routes to i in T must still be greater than any route in P. ■

Lemma 10 *For all routes $J\ = < j, p_j, \omega_j > \in\ P_j$, the set of routes $\{< j, p_x, \omega_x, \varepsilon_x > \in\ P_j\ \mid\ SAT(\varepsilon_j \wedge \varepsilon_x)\}$ are incomparable.*

Proof: By induction. The property holds for the base case. Assume the property holds at the start of some iteration. Let $I\ = < i, p_i, \omega_i, \varepsilon_i >$ be the route considered for addition to P_j in that iteration. By Lemma 9, $\omega_j\ \prec\ \omega_i$, and therefore, by Property 1,

$\omega_j \not\sqsubseteq \omega_i$. Furthermore, from the loop at line 11, $\neg(\varepsilon_i \rightarrow \varepsilon_{sum})$ where ε_{sum} is the disjunction of path expressions for all J where $SAT(\varepsilon_i \wedge \varepsilon_j)$. Therefore, if I is added to P_j at line 15, I is incomparable with all routes in P_j where $SAT(\varepsilon_i \wedge \varepsilon_j)$, and the property still holds following the iteration. ■

Lemma 11 *For each route $J = <j, p_j, \omega_j, \varepsilon_j> \in P \cup T$ the maximal set of routes to j using paths composed of routes in the set $\{<s, p_s, \omega_s, \varepsilon_s> \in P \mid SAT(\varepsilon_j \wedge \varepsilon_s)\}$ is a subset of $\{<j, p_x, \omega_x, \varepsilon_x> \in P \cup T \mid SAT(\varepsilon_j \wedge \varepsilon_x)\}$.*

Proof: By induction. The property holds for the base case. Assume the property holds at the start of some iteration. Let $I = <i, p_i, \omega_i, \varepsilon_i>$ be the route considered for addition to P at line 15. I is not added to P only if it is comparable to routes currently in P_j for all satisfying truth assignments for ε_i (lines 11–12). Furthermore, all relaxed routes are added to T (lines 18–22). Therefore, all potentially incomparable routes are kept in P or T in each iteration, and the property holds after the iteration.

■

Theorem 5 *Policy-Based-Dijkstra computes the maximal set of routes to each node in N for each satisfying truth assignment in finite time.*

Proof: Since the number of routes in the set of routes induced by G is finite, and a route is delete from T in each iteration, the algorithm terminates after a finite number of iterations. Therefore, when the algorithm terminates T is empty. By Lemma 10 all routes in P are incomparable. Furthermore, by Lemma 11, the maximal set of routes for G is a subset of P. ■

3.6 Performance Results of Basic Algorithms

Figures 3.7 through 3.12 (starting on page 54) show the performance of the TD-QoS-Dijkstra algorithm on problems with a range of values for the parameters graph size, average degree, and maximum metric. Each data point in these graphs represents the worst-case performance of the algorithm over 100 runs for a given set of values for these parameters. For each set of these values 10 graphs were randomly generated, and 10 weight assignments were randomly generated for each graph. The graph size range from 100 to 1600 nodes, the average degree from 8 to 32, and the maximum metric from 100 to 65535. Note that the ranges of metrics used are much larger than those typically encountered in deployed Internets. These values were chosen to try and bring out the exponential behavior of the algorithms, which isn't evident at smaller scales, and to match other published results ([76]). Space was measured in terms of the maximum number of entries in the B_i balanced tree structures. The main results to note are that performance as a function of graph size and degree exhibit fairly dramatic growth, hinting at exponential behavior on larger scales, while the maximum metric value has no significant affect on performance. These results agree with those reported in [76].

Figures 3.13 through 3.15 (starting on page 57) graph the number of *candidate routes*, defined as a route considered by the **for** loop at line 12 of the TD-QoS-Dijkstra algorithm, as a function of the same parameters. The similarity of these graphs with the performance graphs illustrates the fact that the performance of these algorithms is fundamentally driven by the number of paths in a graph. Based on these results, the

graphs in Figures 3.16 through 3.21 (starting on page 58) showing the performance results normalized per candidate route were generated to identify the performance characteristics of the underlying algorithms and data structures. Of these graphs, the most significant results are shown in Figures 3.16 and 3.17 where the normalized runtime and space performance of the algorithms are shown to have a significant correlation with the size of the graph. Figure 3.16 illustrates the strongly log-based increase in the performance of the balanced tree and heap data structures, used for B_i and T in the algorithms, as a function of the size of the graphs. Figure 3.17 illustrates that the candidate discard strategies get more effective as the networks grow. The small variation on the y-axis, and the relative flat graphs of Figures 3.19 through 3.21 illustrate the lack of correlation of space with average degree, and space and runtime with maximum metric.

Figures 3.22 through 3.24 (starting on page 61) show the performance of the TD-TE-Dijkstra algorithm on a similar range of parameters. Each data point represents the worst performance of the algorithm out of 9 runs (3 randomly generated graphs with 3 random link weight assignments each). To control the number of forwarding classes in a graph, each graph was generated as two connected subgraphs with the specified average degree and half the specified number of nodes. *Bridge* links were then added between 32 randomly selected pairs of vertices from each subgraph to form a single graph with at most 32 paths between any two nodes in different original subgraphs. 32 tests of the algorithm are then run with all traffic classes initially allocated to one bridge link (resulting in one forwarding class for all 32 traffic classes),

and successive runs are performed with traffic classes distributed over one additional link for each test, with the final run allowing one traffic class over each bridge link (resulting in a one-to-one mapping of traffic classes to forwarding classes). In each test the link predicate of all non-bridge links is set to allow all traffic classes (i.e. it is set to *true*). Each plot shows the results for 1, 8, 16, 24, and 32 forwarding classes in terms of the runtime of the algorithm normalized as a fraction of the "Brute Force" runtime required to run the traditional Dijkstra algorithm once for each traffic class. The plots show that the algorithm performed very well, providing significant savings when the number of forwarding classes is small, and gracefully degrading as the number of forwarding classes grows. Figures 3.25 through 3.27 (starting on page 63) plot the results of these tests normalized per candidate route, illustrating again the log-based performance of the balanced tree and heap data structures.

There are two lessons to be learned from this analysis. First, to minimize the underlying performance driver, candidate routes should be discarded or deleted as soon as it can be determined that they will not be eligible for inclusion in the set of permanent routes. And second, to reduce the performance multiplier incurred by each candidate route considered by the algorithm, the data structures used for the temporary routes should be made as efficient as possible. The next section presents enhanced versions of these algorithms that use these strategies to achieve significant performance improvements.

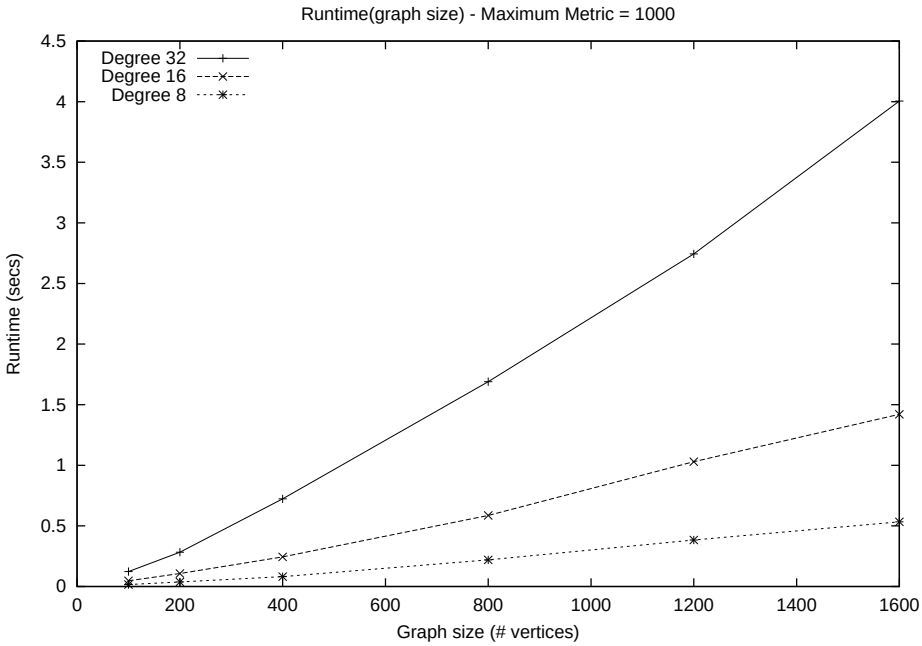

Figure 3.7: Runtime(Size)

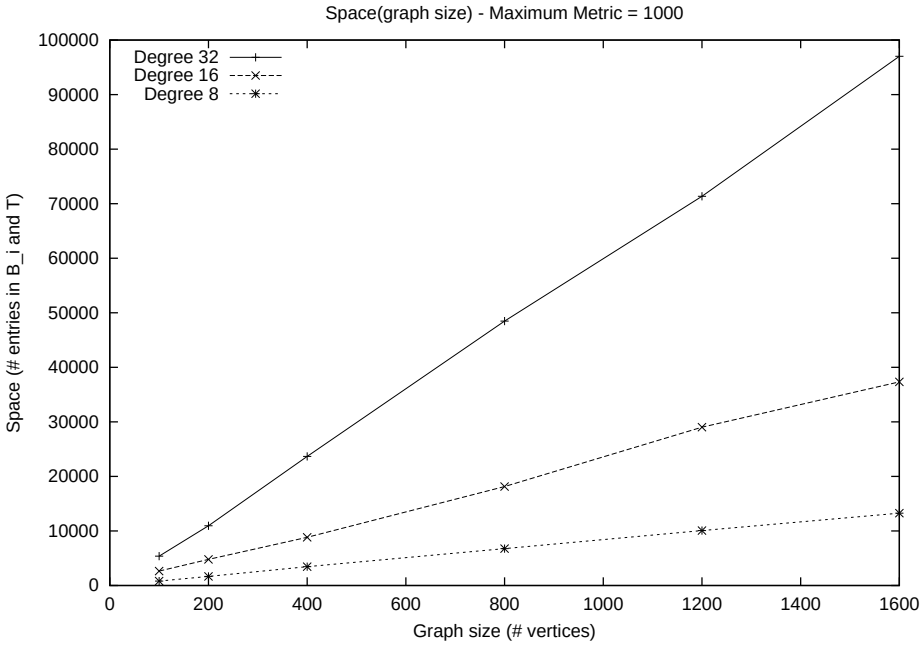

Figure 3.8: Space(Size)

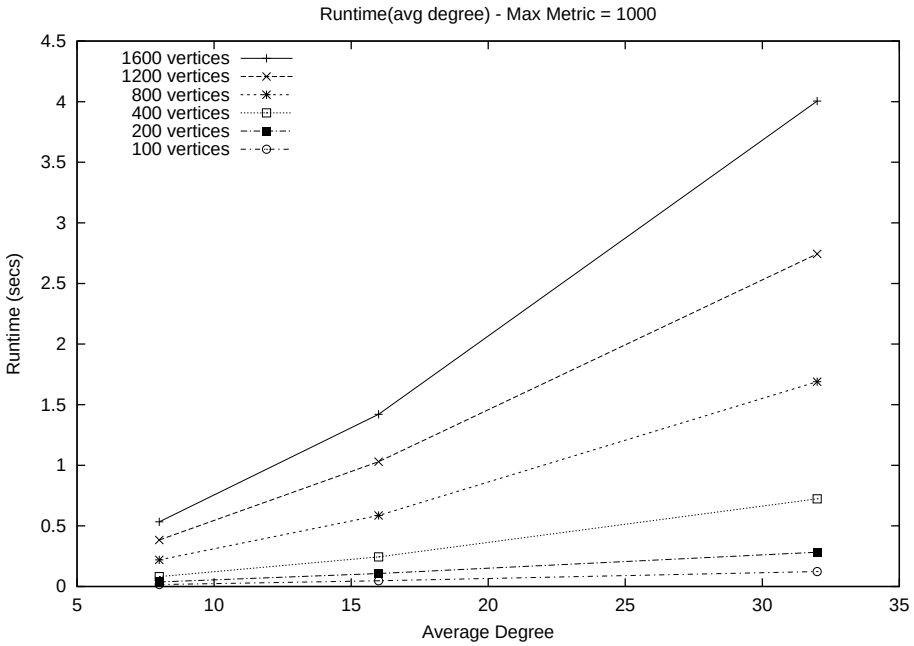

Figure 3.9: Runtime(Avg Degree)

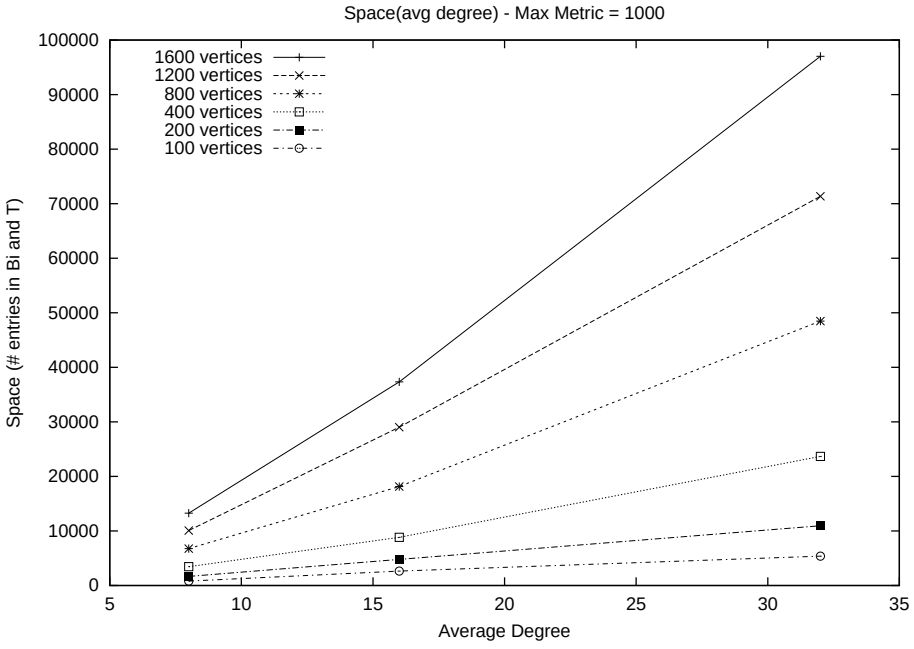

Figure 3.10: Space(Avg Degree)

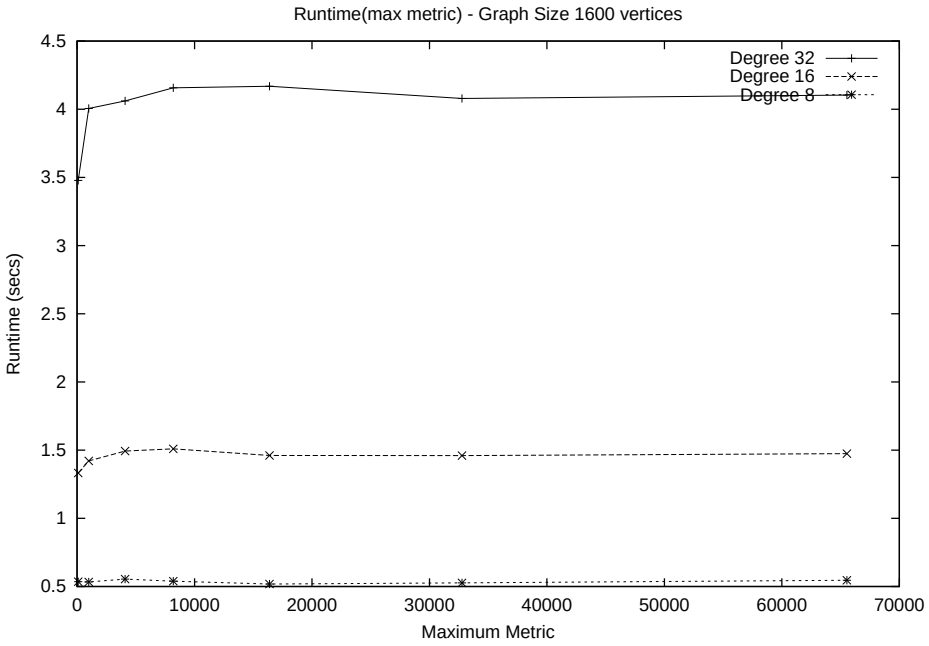

Figure 3.11: Runtime(Maximum Metric)

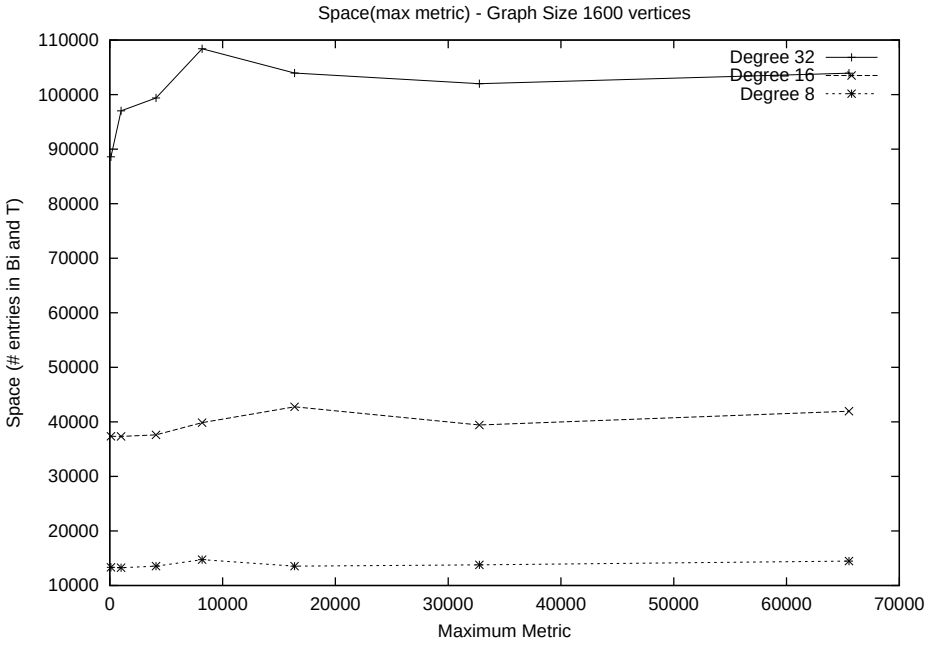

Figure 3.12: Space(Maximum Metric)

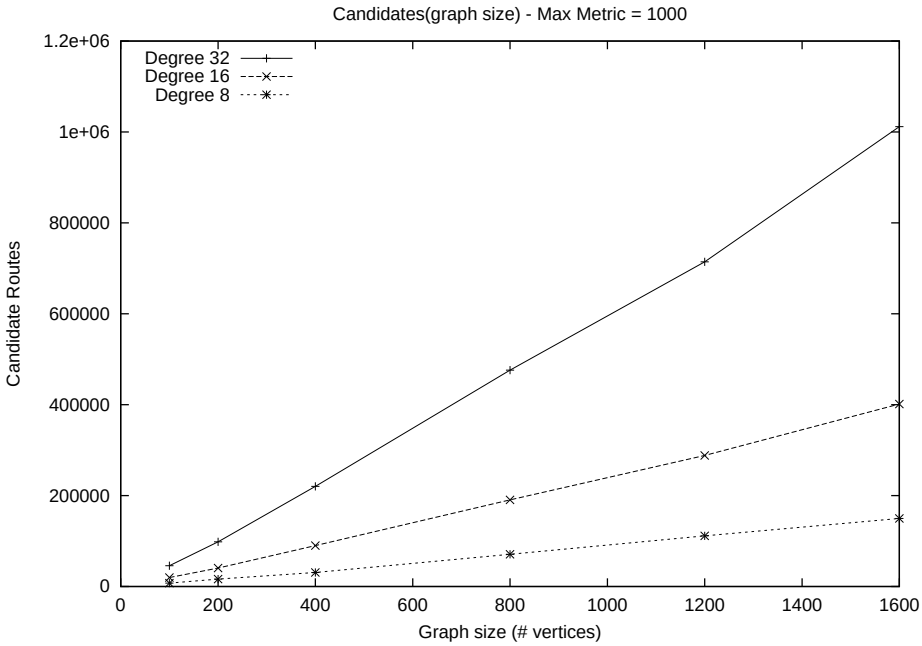

Figure 3.13: Candidate(Size)

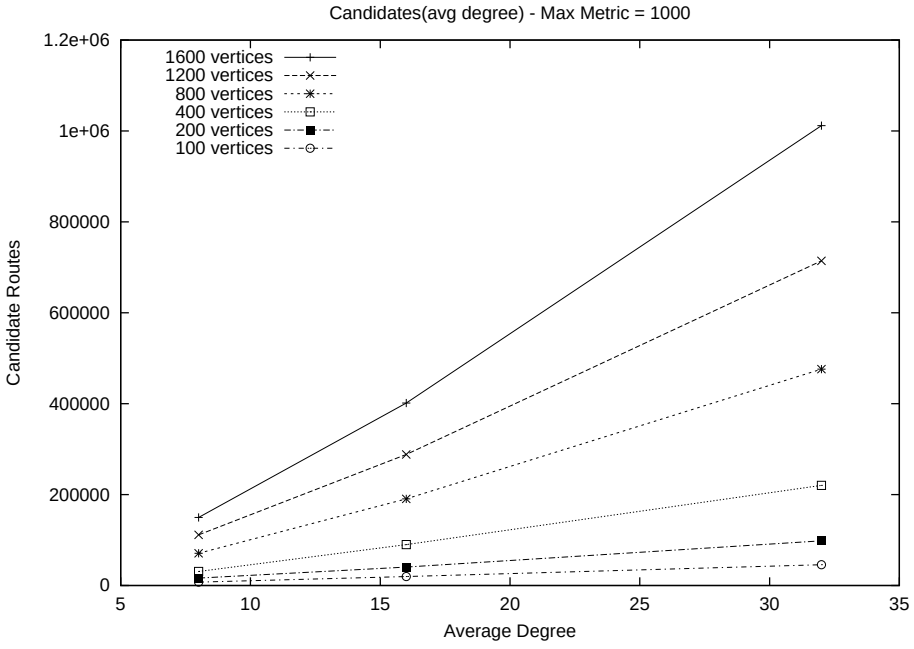

Figure 3.14: Candidate(Avg Degree)

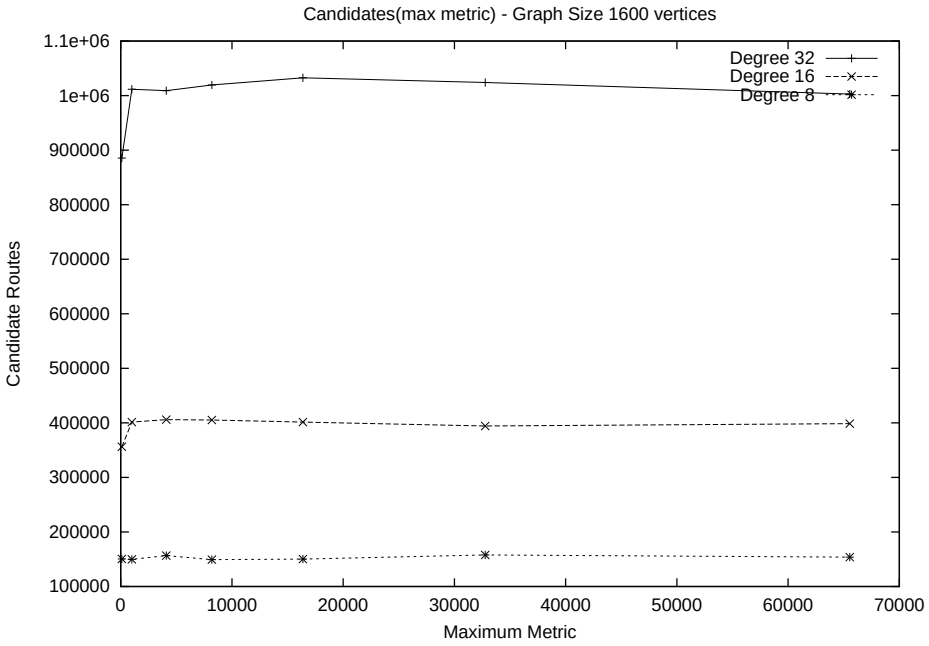

Figure 3.15: Candidate(Maximum Metric)

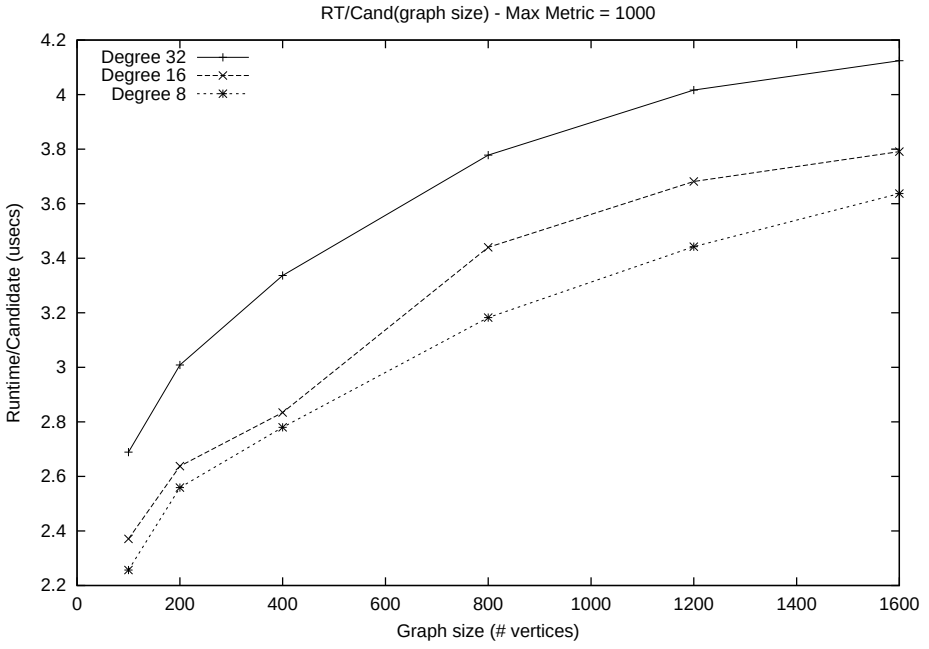

Figure 3.16: Normalized Runtime(Size)

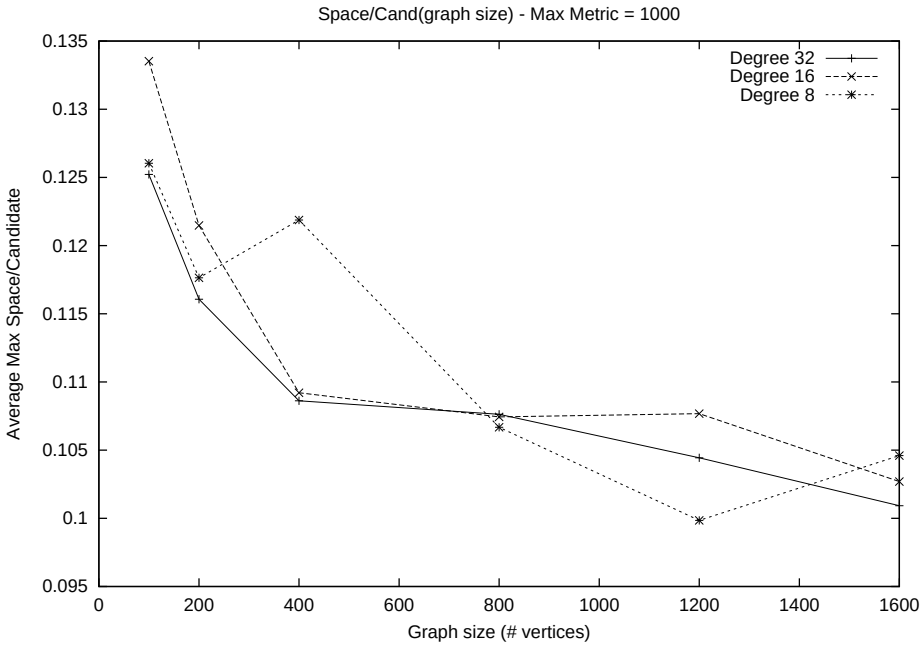

Figure 3.17: Normalized Space(Size)

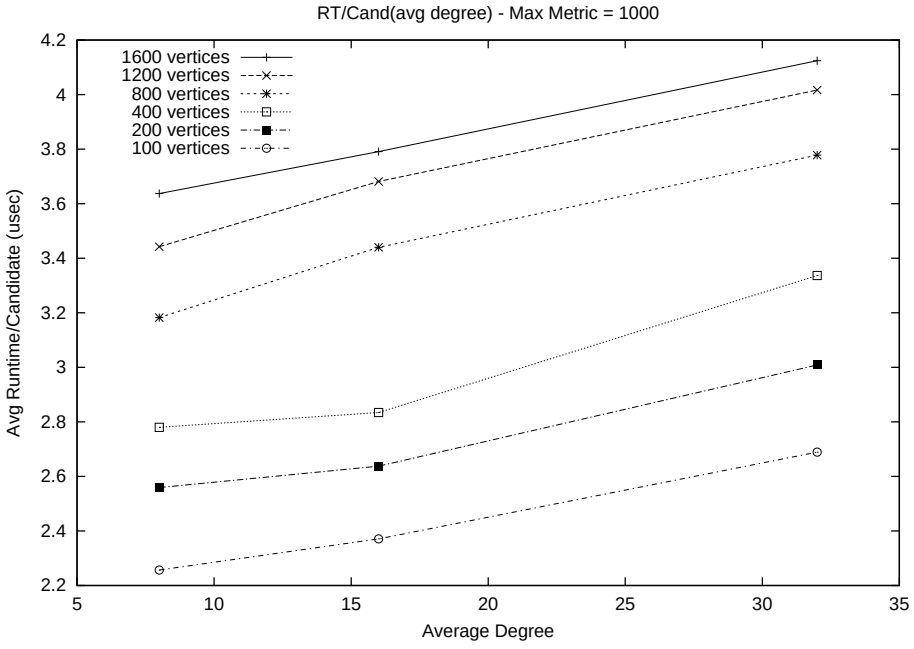

Figure 3.18: Normalized Runtime(Avg Degree)

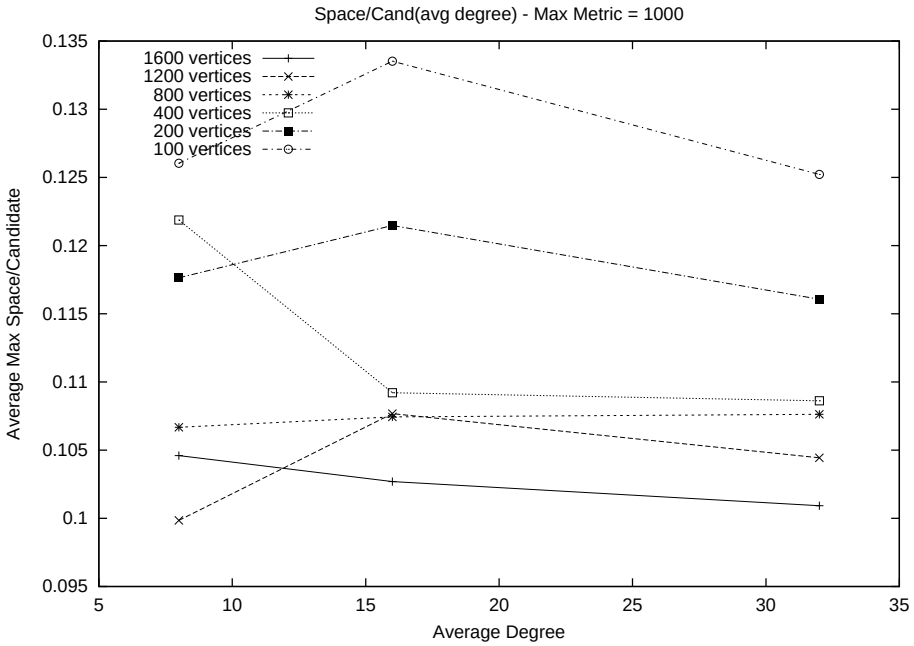

Figure 3.19: Normalized Space(Avg Degree)

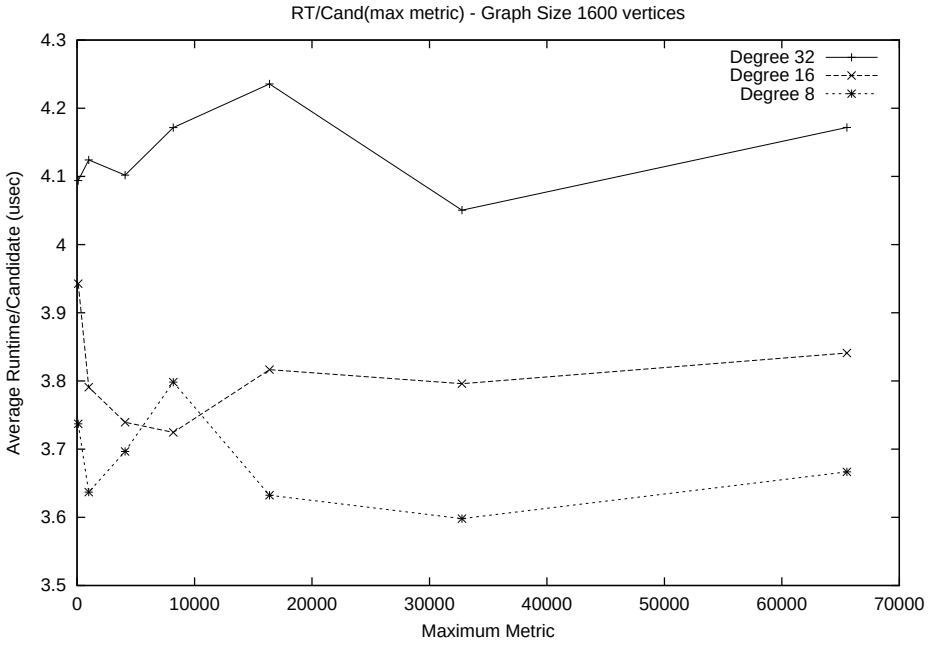

Figure 3.20: Normalized Runtime(Maximum Metric)

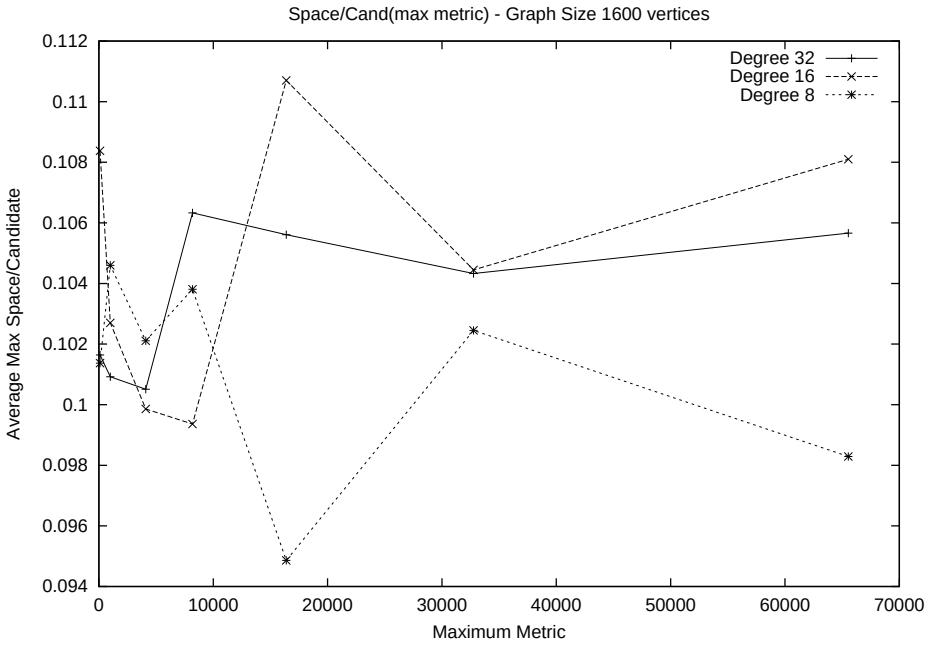

Figure 3.21: Normalized Space(Maximum Metric)

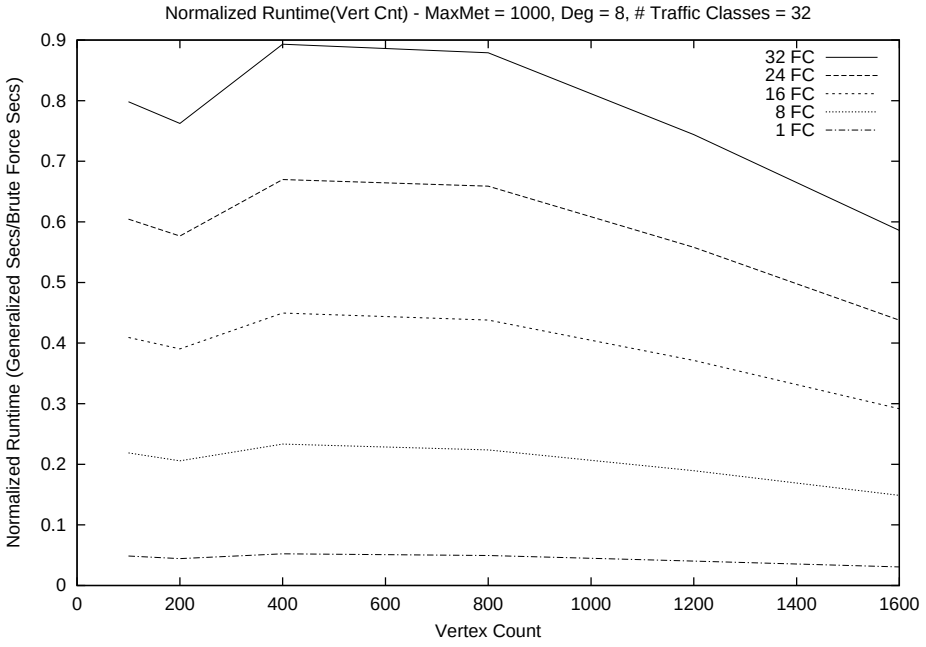

Figure 3.22: Traffic Engineering - Generalized/Brute Force Runtime(Size), Deg = 8

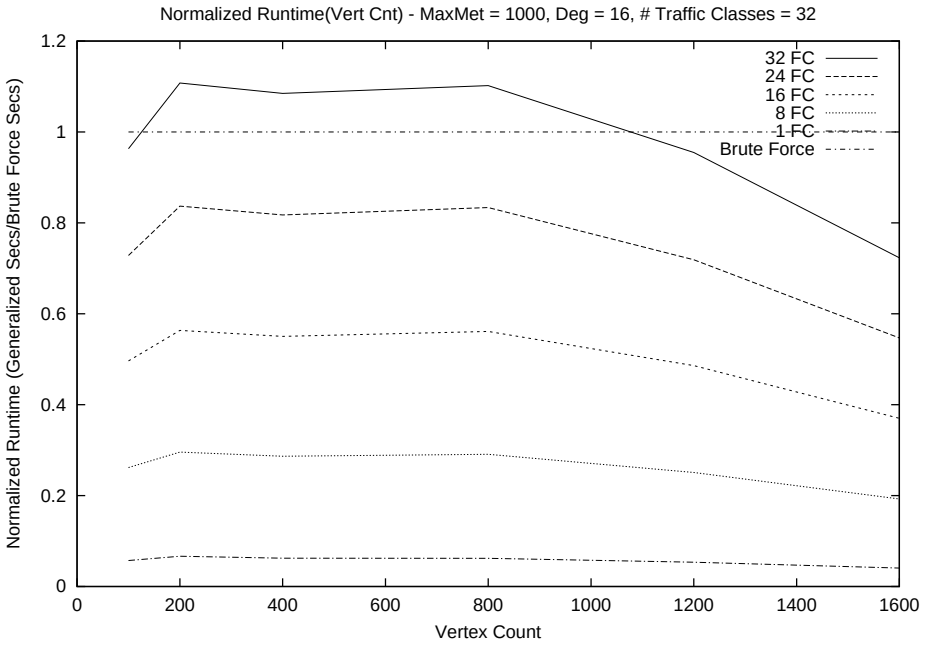

Figure 3.23: Traffic Engineering - Generalized/Brute Force Runtime(Size), Deg = 16

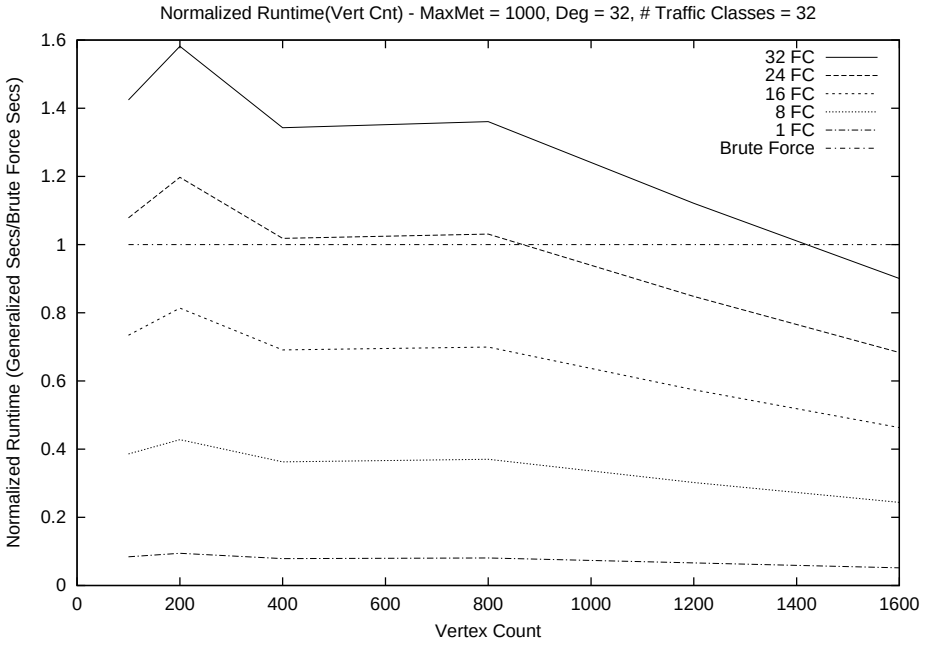

Figure 3.24: Traffic Engineering - Generalized/Brute Force Runtime(Size), Deg = 32

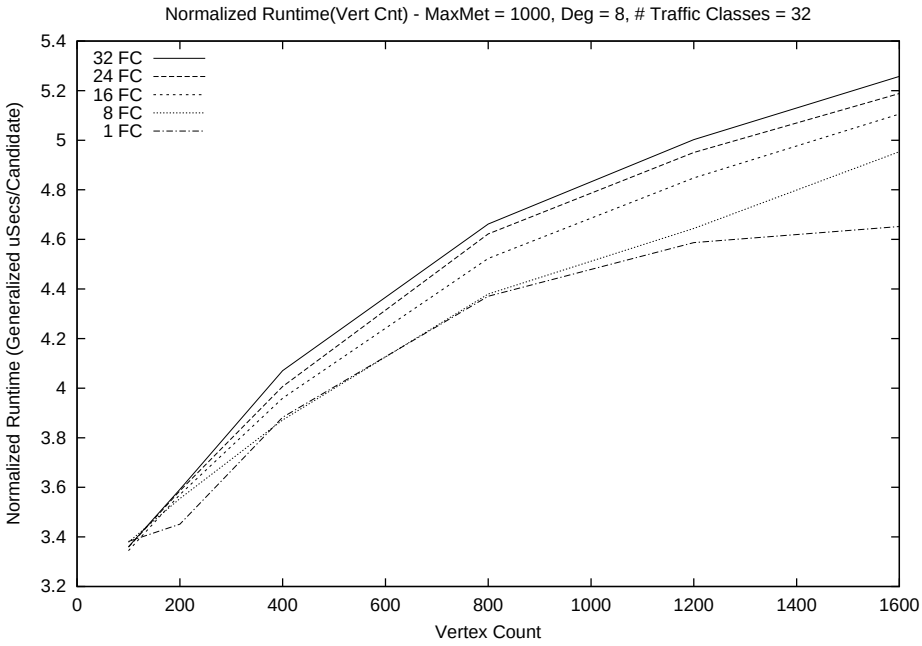

Figure 3.25: Traffic Engineering - Per Candidate Runtime(Size), Deg = 8

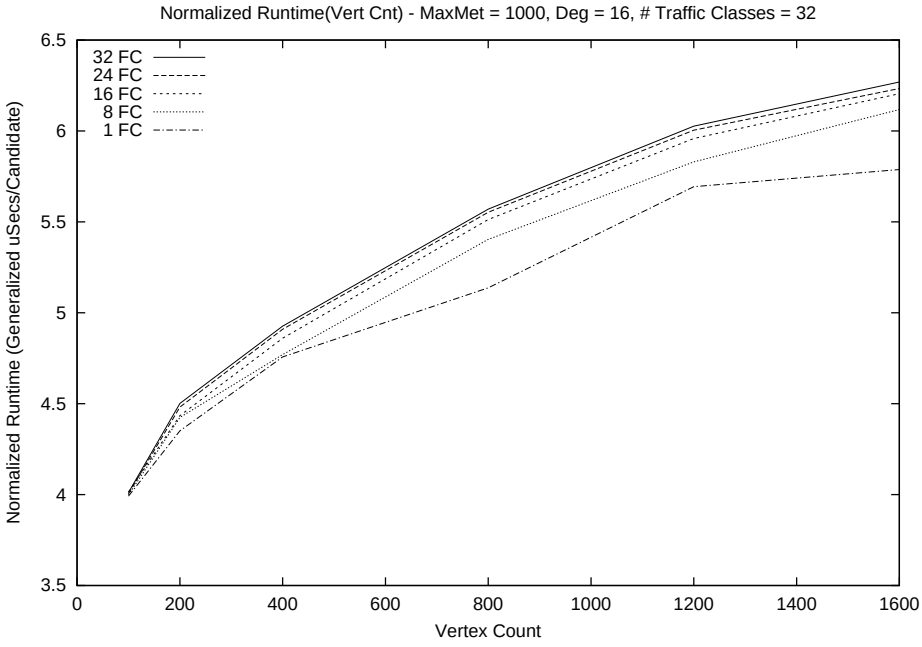

Figure 3.26: Traffic Engineering - Per Candidate Runtime(Size), Deg = 16

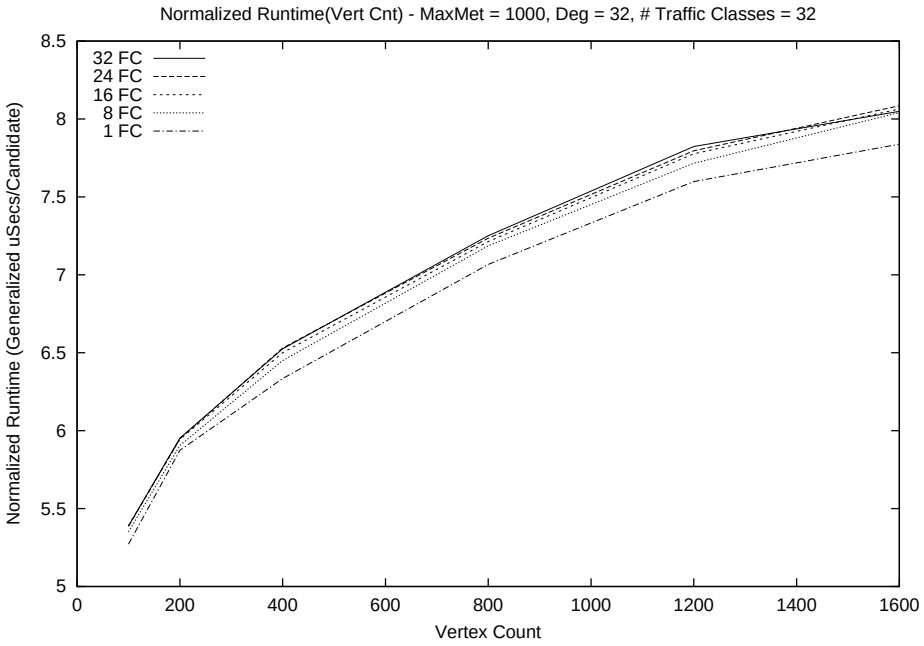

Figure 3.27: Traffic Engineering - Per Candidate Runtime(Size), Deg = 32

3.7 Enhanced Exact Algorithms

As discussed in the introduction to this chapter, the $\log(A)$ and $\log(W)$ factors in the complexity of the TD-TE-Dijkstra and TD-QoS-Dijkstra algorithms (respectively) presented in Sections 3.2 and 3.4 is a result of the use of a balanced tree for storing the temporarily labeled nodes for a given destination. This section presents enhanced versions of those algorithms which use a queue-based data structure for this purpose, reducing the cost of managing these structures to a lower order term in the time complexity. As a result the runtime cost of the enhanced algorithms becomes dominated by $\log_d(n)$ factors from the manipulation of the T heap.

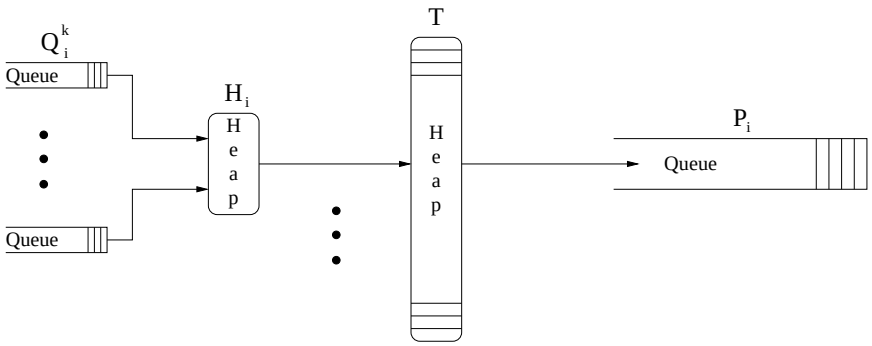

Figure 3.28: Model of Data Structures for Enhanced Algorithms

This enhancement is based on the property that routes to a given node *with the same predecessor* are discovered in strictly increasing (or non-decreasing, depending on the algorithm) order. This property is a direct result of Lemmas 3 and 6 which imply that routes to a given predecessor will be discovered in strictly increasing (non-decreasing) order, and therefore the order of discovery of routes from a given predecessor to one of its neighbors will have the same property.

Based on this insight, the data structure shown in Figure 3.28 can be used to improve the performance of the algorithms presented in Chapter 3. In this data structure the balanced trees for each node are replaced with a set of queues for each neighbor of the node, and a summary heap containing the head of each neighbor queue. Exploiting the ordering property of these queues, the algorithms ensure that each node head H_i, and therefore T_i, contain the lightest route in the link queues that is not subsumed by the routes in P_i.

Figure 3.29 presents the enhanced version of the TD-TE-Dijkstra algorithm. As described in Table 3.2, the new notation $\mathcal{E}_n^k$ represents a traffic expression representing all routes that have been added to Q_n^k. The proof of correctness of this algorithm

```
algorithm TD-TE-Dijkstra
    begin
1   for each {n ∈ N} E_n ← 0;
2   for each {(n,m) ∈ E} ε_n^m ← ε_m^n ← 0;
3   Push(<s, s, 0, 1>, P_s);
4   for each {(s,j) ∈ A(s)}
      begin
5     Push(<j, s, ω_sj, ε_sj>, Q_j^s);
6     Insert(<j, s, ω_sj, ε_sj>, H_j);
7     Insert(<j, s, ω_sj, ε_sj>, T);
      end
8   while (|T| > 0)
      begin
9     <n, p, ω, ε> ← Min(T);
10    E_n ← E_n ∨ ε;
11    Push(<n, p, ω, ε>, P_n);
12    DeleteTMin();
13    for each {(n,j) ∈ A(n)}
        begin
14      ω_n ← ω ⊕ ω_nj; ε_n ← ε ∧ ε_nj;
15      AddCandidate(<j, n, ω_n, ε_n>);
        end
      end
    end


function TE-DeleteTMin()
    // Delete minimum entry from T and restore invariants.
    //   Invariant 3 – only deletes routes (line 9) where
    //   every STA for the route also satisfies a known
    //   better route (a route in P_n).
    //   Invariant 4 – loop at line 7 ensures there are some
    //   STAs for new T_n that do not satisfy any known
    //   better route (current T_n or any route in P_n).
    // Assumes E_n has been updated with Min(T).ε.
    begin
1   <n, p, ω, ε> ← Min(T);
2   Pop(Q_n^p);
3   if (|Q_n^p| > 0)
4     then IncreaseKey(Head(Q_n^p), H_n^p)
5     else DeleteMin(H_n);
6   if (|H_n| > 0)
      then begin
        // Find smallest route in link queues
        // where ¬(ε → E_n).
7       for each {(n,k) ∈ A(n) | (|Q_n^k| > 0) ∧
                   (Head(Q_n^k).ε → E_n)}
          begin
8           while ((|Q_n^k| > 0) ∧ (Head(Q_n^k).ε → E_n))
9             Pop(Q_n^k);
10          if (|Q_n^k| > 0)
11            then IncreaseKey(Head(Q_n^k), H_n^k)
12            else Delete(H_n^k);
            end
13      if (|H_n| > 0)
          then IncreaseKey(Min(H_n), T_n); return;
        end
14  DeleteMin(T);
    end


function TE-AddCandidate(<n, p, ω_a, ε_a>)
    // Add new route to appropriate Q and restore invariants.
    //   Invariant 3 – only drops routes (lines 1, 10, and 21)
    //   where every STA for the route also satisfies a known
    //   better route.
    //   Invariant 4 – ensures there is an STA for Min(H_n)
    //   that is not satisfied by any known better route.
    begin
1   if (ε_a → E_n) then return;
2   if (|H_n| = 0)
      then begin
3       Push(<n, p, ω_a, ε_a>, Q_n^p);
4       Insert(<n, p, ω_a, ε_a>, H_n);
5       Insert(<n, p, ω_a, ε_a>, T);
6       return;
        end
7   <n, k, ω_m, ε_m> ← Min(H_n);
8   if (ω_m ≤ ω_a)
      then // Know ¬(ε_a → E_n) from line 1.
9       if (ε_a → (ε_m ∨ ε_n^p)))
10        then return; // All STAs for ε_a satisfy a better route.
11        else begin // ¬(ε_a → (ε_m ∨ ε_n^p)))
            // There is an STA for ε_a that does not satisfy
            // any known better routes.
12          if (|Q_n^p| = 0)
13            then Insert(<n, p, ω_a, ε_a>, H_n)
14          Push(<n, p, ω_a, ε_a>, Q_n^p);
            end
      else // ω_m > ω_a; since ω_a > Min(H_n^p), it must be
           // true that |Q_n^p| = 0.
        if (¬(ε_m → (ε_a ∨ E_n)))
          then begin
            // There is an STA for ε_m that doesn't satisfy any
            // known better route.
15          Push(<n, p, ω_a, ε_a>, Q_n^p);
16          Insert(<n, p, ω_a, ε_a>, H_n);
            // Following replaces <n, k, ω_m, ε_m>.
17          DecreaseKey(<n, p, ω_a, ε_a>, T_n);
            end
          else begin // (ε_m → (ε_a ∨ E_n))
            // All STAs for ε_m also satisfy a better route.
18          Push(<n, p, ω_a, ε_a>, Q_n^p);
            // Following replaces <n, k, ω_m, ε_m>.
19          DecreaseKey(<n, p, ω_a, ε_a>, H_n^k);
20          DecreaseKey(<n, p, ω_a, ε_a>, T_n);
21          Pop(Q_n^k);
22          if (|Q_n^k| > 0)
23            then Insert(Head(Q_n^k), H_n);
            end
    end
```

Figure 3.29: Enhanced Traffic Engineering Dijkstra.

```
algorithm TD-QoS-Dijkstra
    begin
1   Push(<s, s, 0̄>, P_s);
2   for each {(s, j) ∈ A(s)}
      begin
3     Push(<j, s, ω_sj>, Q^s_j);
4     Insert(<j, s, ω_sj>, H_j);
5     Insert(<j, s, ω_sj>, T);
      end;
6   while (|T| > 0)
      begin
7     <n, p, ω> ← Min(T);
8     Push(<n, p, ω>, P_n);
9     DeleteTMin();
10    for each {(n, j) ∈ A(n)}
        begin
11      ω_n ← ω ⊕ ω_nj;
12      AddCandidate(<j, n, ω_n>);
        end
      end
    end


function QoS-DeleteTMin()
    // Delete minimum entry from T and restore invariants:
    //   Invariant 3 – only deletes routes (line 9) that are
    //   ⊑ another route.
    //   Invariant 4 – loop at line 7 ensures new T_n ⋤
    //   new Tail(P_n).
    begin
1   <n, p, ω> ← Min(T);
2   Pop(Q^p_n);
3   if (|Q^p_n| > 0)
4     then IncreaseKey(Head(Q^p_n), H^p_n)
5     else DeleteMin(H_n);
6   if (|H_n| > 0)
      then begin
        // Find smallest route in link queues that is not
        // ⊑ the deleted route.
7       for each {(n, k) ∈ A(n) | (|Q^k_n| > 0) ∧
                (Head(Q^k_n).ω ⊑ ω)}
          begin
8           while ((|Q^k_n| > 0) ∧ (Head(Q^k_n).ω ⊑ ω))
9             Pop(Q^k_n);
10          if (|Q^k_n| > 0)
11            then IncreaseKey(Head(Q^k_n), H^k_n)
12            else Delete(H^k_n);
            end
13      if (|H_n| > 0)
          then IncreaseKey(Min(H_n), T_n); return;
        end
14  DeleteMin(T);
    end
```

```
function QoS-AddCandidate(<n, p, ω_a>)
    // Add new route to appropriate Q and restore invariants:
    //   Invariant 3 – only drops known comparable routes
    //   (lines 1, 10, 15, and 24).
    //   Invariant 4 – ensures Min(H_n) ⪯ (and therefore ⋦)
    //   all routes in Q*_n queues.
    begin
1   if (ω_a ⊑ Tail(P_n).ω) then return;
2   if (|H_n| = 0)
      then begin
3       Push(<n, p, ω_a>, Q^p_n);
4       Insert(<n, p, ω_a>, H_n);
5       Insert(<n, p, ω_a>, T);
6       return;
        end
7   <n, k, ω_m> ← Min(H_n);
8   if (ω_m ⪯ ω_a)
      then
9       if (ω_a ⊑ ω_m)
10        then return; // ω_a down and right of ω_m
11        else begin // ((ω_a ⋢ ω_m) ∧ (ω_m ⪯ ω_a))
            // ω_a down and left of ω_m
12          if (|Q^p_n| = 0)
13            then Insert(<n, p, ω_a>, H_n)
14            else if (ω_a ⊑ Tail(Q^p_n).ω)
15              then return;
16          Push(<n, p, ω_a>, Q^p_n);
            end
      else // ω_a ≺ ω_m; since ω_a ≻ Min(H^p_n), it must be
           // true that |Q^p_n| = 0.
17      if (ω_a ⋣ ω_m)
          then begin // ω_a up and right of ω_m.
18          Push(<n, p, ω_a>, Q^p_n);
19          Insert(<n, p, ω_a>, H_n);
            // Following replaces <n, k, ω_m>.
20          DecreaseKey(<n, p, ω_a>, T_n);
            end
          else begin // (ω_a ⊒ ω_m)
            // ω_a up and left of ω_m.
21          Push(<n, p, ω_a>, Q^p_n);
            // Following replaces <n, k, ω_m>.
22          DecreaseKey(<n, p, ω_a>, H^k_n);
23          DecreaseKey(<n, p, ω_a>, T_n);
24          Pop(Q^k_n);
25          if (|Q^k_n| > 0)
26            then Insert(Head(Q^k_n), H_n);
            end
    end
```

Figure 3.30: Enhanced QoS Dijkstra.

mirrors that presented in Section 3.2, and is based on the fact that invariants listed in Table 3.1 are maintained by the enhanced algorithm. Specifically, as detailed in the comments to these algorithms, invariants 3 and 4 are maintained by the DeleteTMin() and AddCandidate() functions, and, based on this, the Dijkstra iteration over the n^{th} best route in the main body of the algorithm ensures invariants 1 and 2.

Similarly, Figure 3.30 presents the enhanced version of the TD-QoS-Dijkstra

algorithm. The proof of correctness of this algorithm mirrors that presented in Section 3.4, and is based on the fact that invariants listed in Table 3.1 are maintained by the enhanced algorithm. Specifically, as detailed in the comments to these algorithms, invariants 3 and 4 are maintained by the DeleteTMin() and AddCandidate() functions, and, based on this, the Dijkstra iteration over the n^{th} best route in the main body of the algorithm ensures invariants 1 and 2.

The runtime complexity of the TD-TE-Dijkstra (TD-QoS-Dijkstra) algorithm (again, ignoring the cost for determining satisfiability) is dominated by the loops at lines 8 (6) and 13 (10). The loop at line 8 (6) is executed at most once for each incomparable path to each node in the graph for a total of nA (nW) times. The loop at line 13 (10) is executed at most once for each distinct instance of an edge in the graph, for a total of mA (mW) times. The most costly operation in the loop at line 8 (6) is the DeleteTMin() call at line 11 (9). In the DeleteTMin() routine, the loop at line 7 (7) will be executed, in total, at most once per neighbor for each forwarding class for a total of $a_{max}A$ ($a_{max}W$), and the cost per call of the heap operations at lines 13 (13) and 14 (14) is $d \log_D(n)$. Therefore, the total worst-case cost of the call at line 11 (8) of the main algorithm is $nA \log_d(n) + a_{max}A$ ($nW \log_d(n) + a_{max}W$). In the AddCandidate() routine, the runtime complexity is dominated by the heap operations at lines 5 (5), 17 (20), and 20 (23), which cost $\log_d(n)$ each, for a total cost of the call to AddCandidate() at line 15 (12) of the main algorithm of $mA \log_d(n)$ ($mW \log_d(n)$). Therefore, the worst-case time complexity of the enhanced TD-TE-Dijkstra (TD-QoS-Dijkstra) algorithm is $O(mA \log(n))$ ($O(mW \log(n))$).

Figures 3.31 through 3.36 (starting on page 72) show the results of running the same tests described above with the enhanced version of the TD-QoS-Dijkstra algorithm. These graphs show a significant improvement in the performance of the enhanced algorithms in comparison to the basic algorithms. Furthermore, the normalized graphs shown in Figures 3.37 through 3.42 (starting on page 75) identify the sources of these savings. The flattening of the graph of space requirements as a function of the graph size (Figure 3.38) is a result of the far more aggressive candidate dropping performed in the enhanced algorithms. With the much smaller resulting data structures there is a corresponding decrease in performance costs across all the graphs. Specifically, the conditional return at line 15 of the enhanced QoS AddCandidate() routine ensures that each Q_i^p contains only incomparable routes at all times, and the loop at line 8 of the QoS DeleteTMin() routine ensures that the front of all Q_i^p for a give i contain only incomparable routes following the transfer of a route from i to P. The only remaining source of "excess" routes held in the data structures, and processed by the algorithms are the comparable routes carried in different Q_i^p for destination i until they are cleaned up the the DeleteTMin() loop. One inexpensive opportunity to identify and cleanup such routes occurs at line 24 of the QoS AddCandidate() routine where a loop similar to that in DeleteTMin() could be executed to clean up a queue with known deletable routes at its front. However, experiments showed that such a loop would only rarely (if ever) delete more than one route, and therefore such effort is wasted.

Similarly, the fact that the graph of runtime as a function of the graph size

(Figure 3.31 has flattened noticeably is the result of replacing the log-time balanced tree data structure for B_i with a constant-time queue data structure for Q_i^p. In addition, the very low cost of the simple queue data structure contributes to the decrease in cost across all graphs. In summary, the more aggressive candidate drop strategy and cheaper queue-based data structure used in the enhanced algorithms result in a significant general reduction in the cost of the routing computations. Furthermore, the replacement of the log-time balanced tree structures with constant-time queue data structures by these algorithms has damped the growth rate of their performance. Figures 3.43 through 3.48 (starting on page 78) summarize these results by comparing the run-time and space performance of the basic, enhanced, and traditional algorithms.

In practical terms these results are very promising. Focusing on the 200-node network, which represents a typical high-end network size supportable with current routing protocols, with an average degree of eight, which is more richly connected than is typical for current networks but reasonable to expect with the availablilty of this technology, the average worst case times are 0.0256 seconds per TD-TE-Dijkstra routing computation and 0.01844 seconds per TD-QoS-Dijkstra computation. For TD-TE-Dijkstra this translates to an ability to process approximately one administrative policy change per link every 5 seconds (assuming a separate route processor common in commercial routers). This capacity is much greater than needed for the expected use of administrative policies. For TD-QoS-Dijkstra this translates to an ability to process approximately one cost change per link every 3 seconds. Given a critical lack of results on routing update rates in the literature, it is hard to evaluate this number; however, a

few observations can be made. The need for QoS routing typically arises from the use of applications that involve relatively long-lived connections to justify the overhead of QoS signalling mechanisms. The stability of connections established for QoS flows will tend to dampen the oscillations in load in a network. Furthermore, it is possible to control the update rates of such applications using various mechanisms [4]. More research is needed in this area to evaluate the performance of the TD-QoS-Dijkstra algorithm.

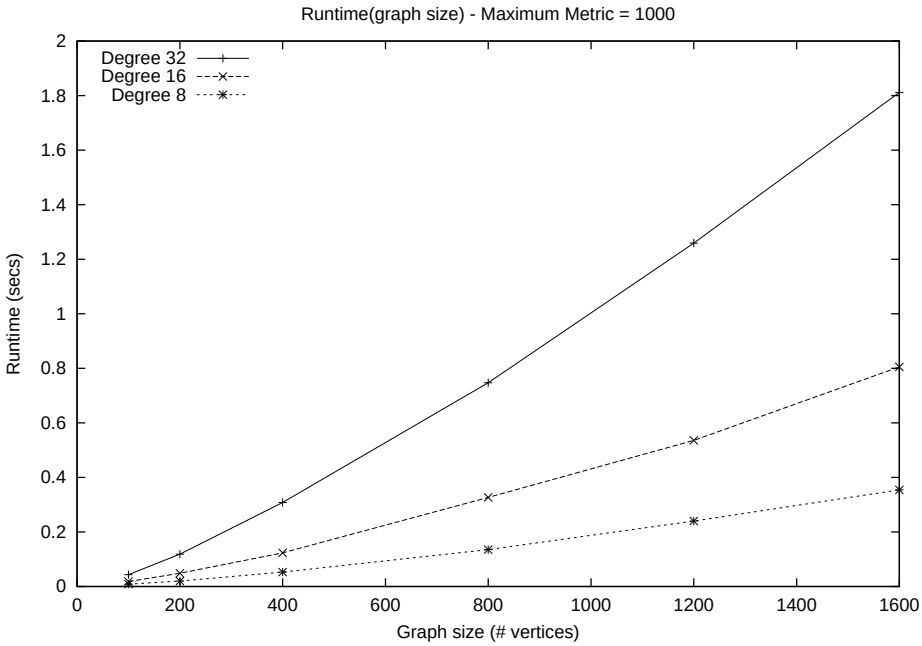

Figure 3.31: Enhanced Runtime(Size)

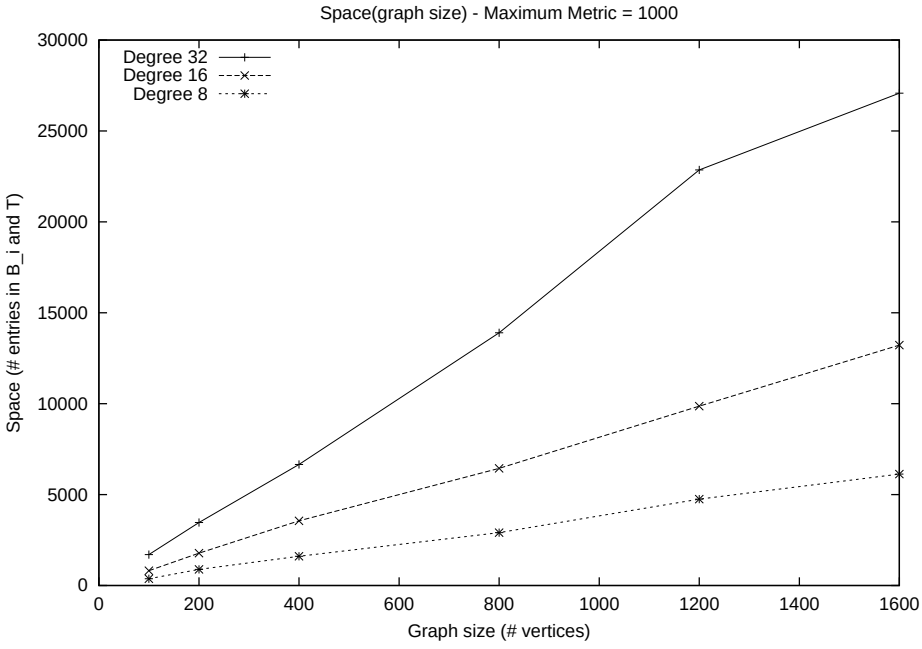

Figure 3.32: Enhanced Space(Size)

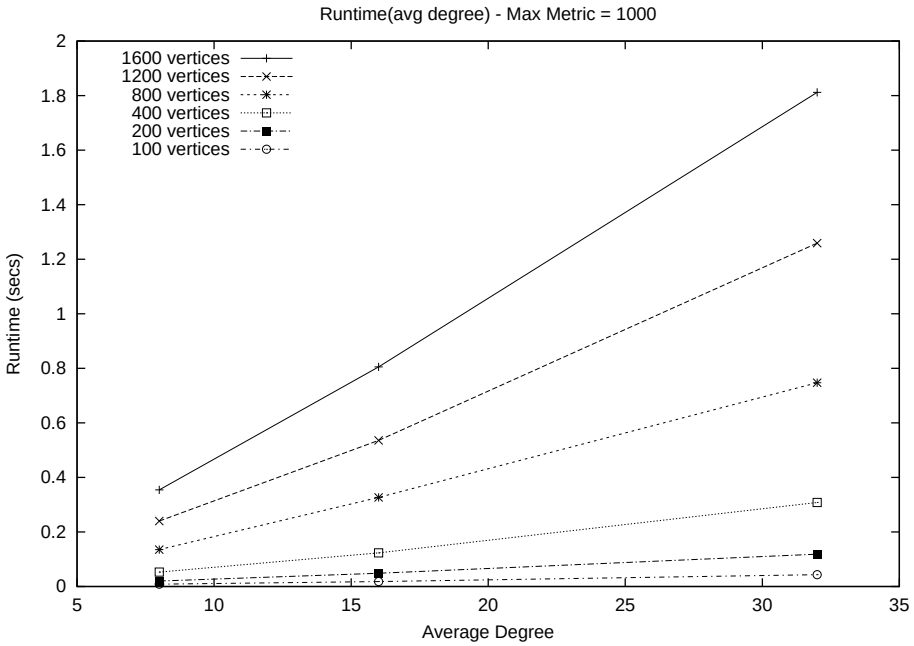

Figure 3.33: Enhanced Runtime(Avg Degree)

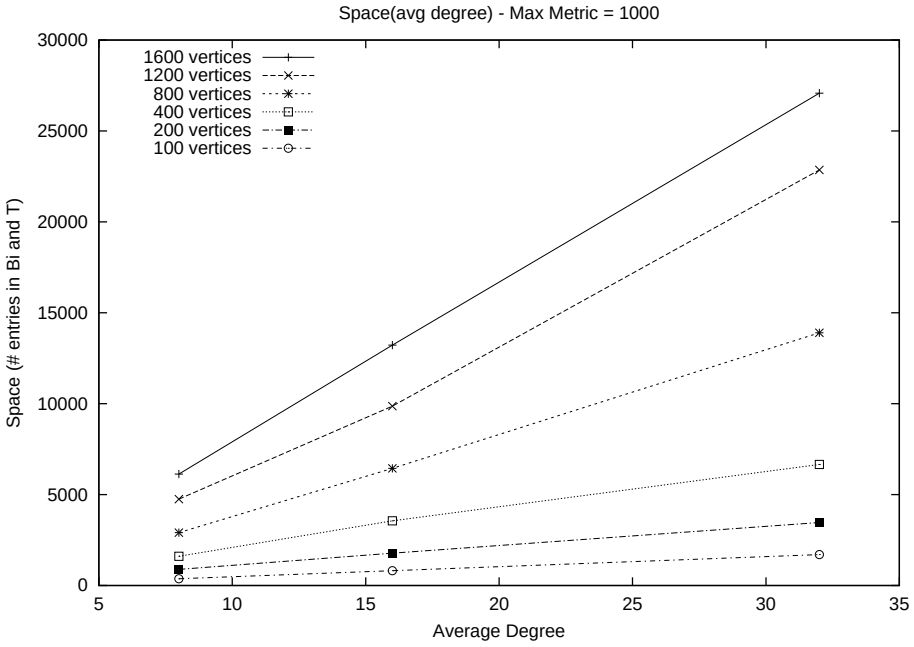

Figure 3.34: Enhanced Space(Avg Degree)

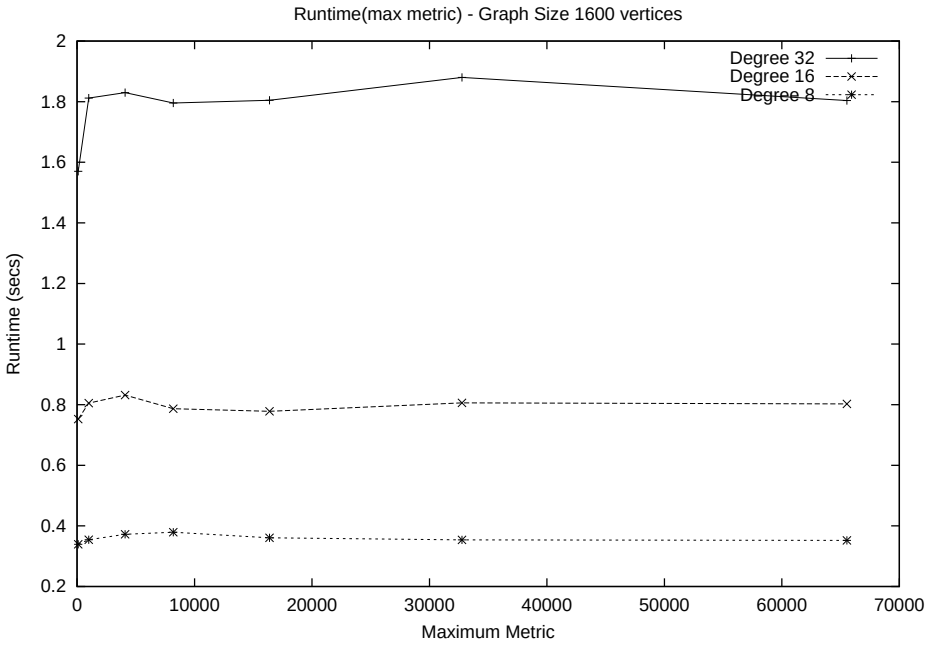

Figure 3.35: Enhanced Runtime(Maximum Metric)

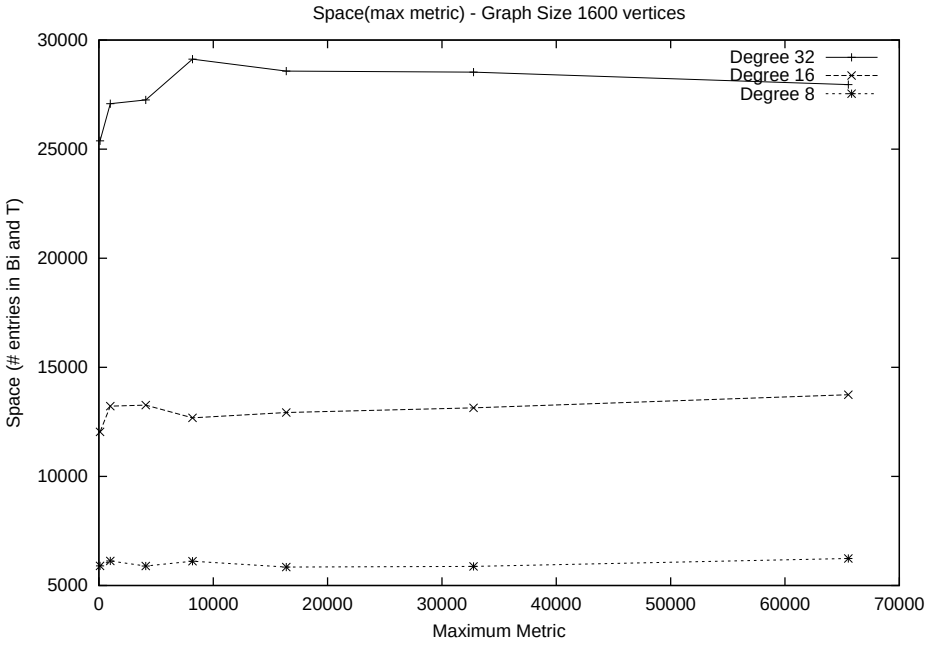

Figure 3.36: Enhanced Space(Maximum Metric)

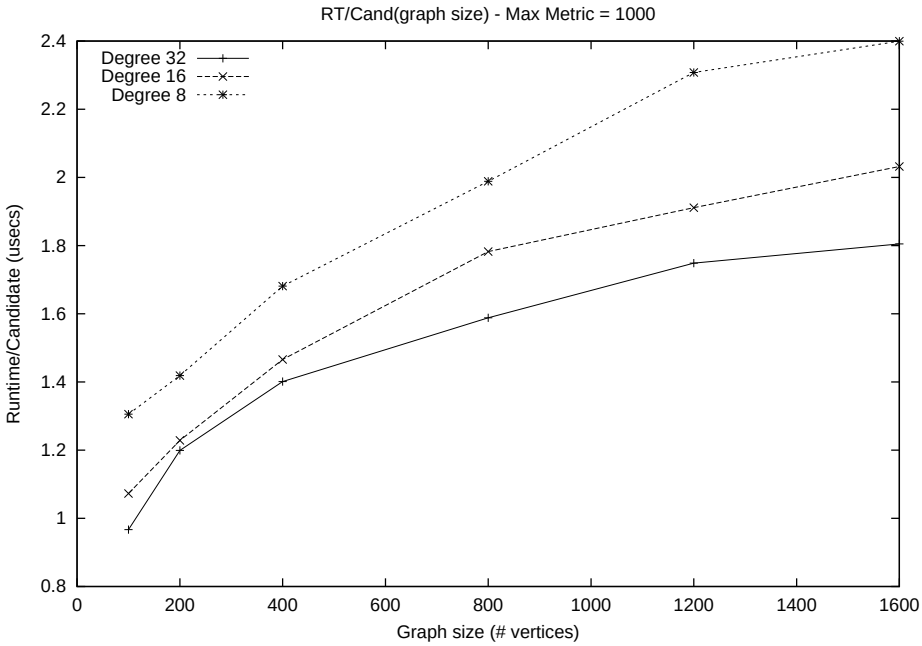

Figure 3.37: Enhanced Normalized Runtime(Size)

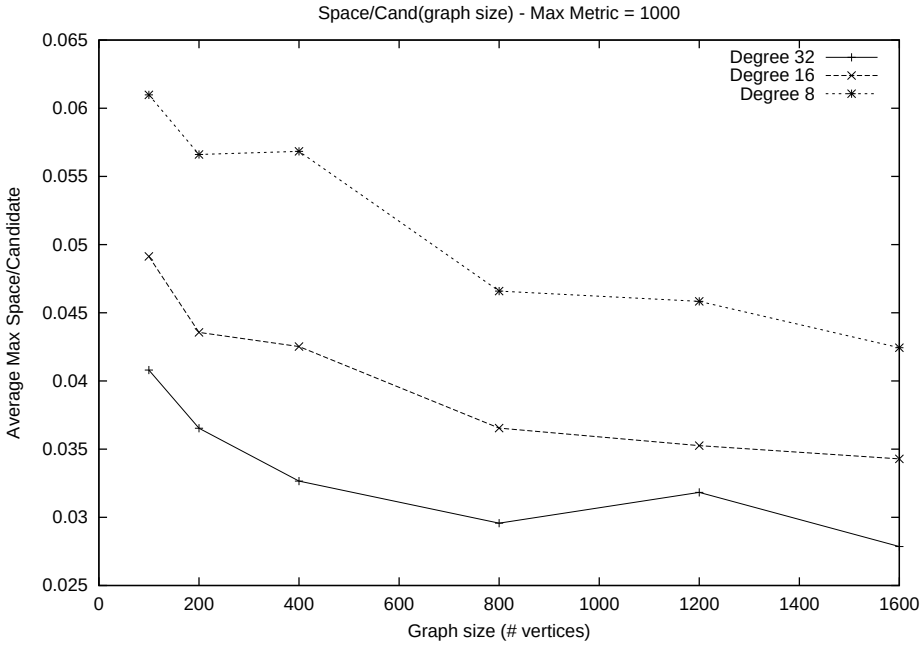

Figure 3.38: Enhanced Normalized Space(Size)

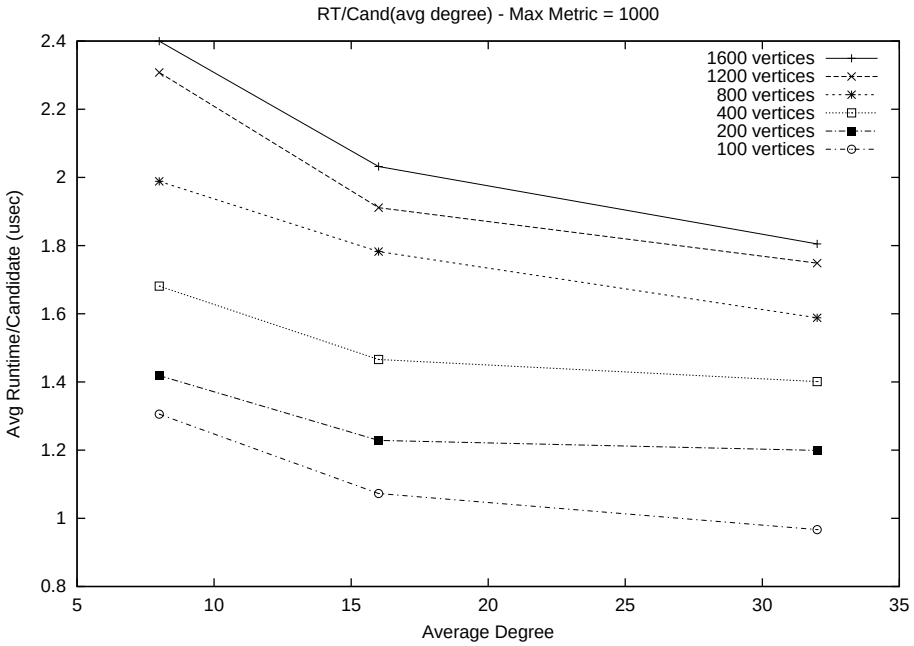

Figure 3.39: Enhanced Normalized Runtime(Avg Degree)

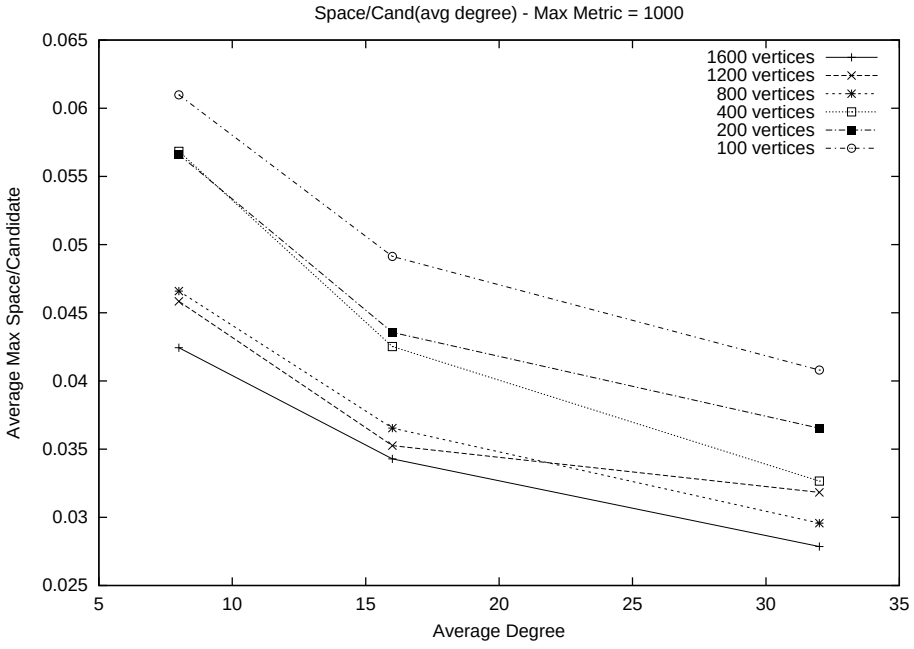

Figure 3.40: Enhanced Normalized Space(Avg Degree)

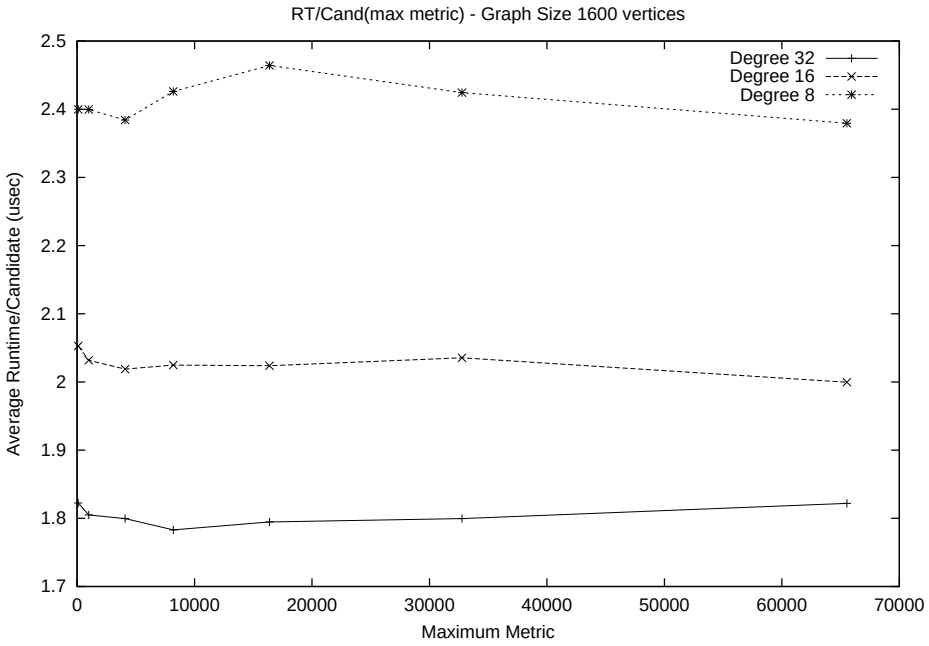

Figure 3.41: Enhanced Normalized Runtime(Max Met)

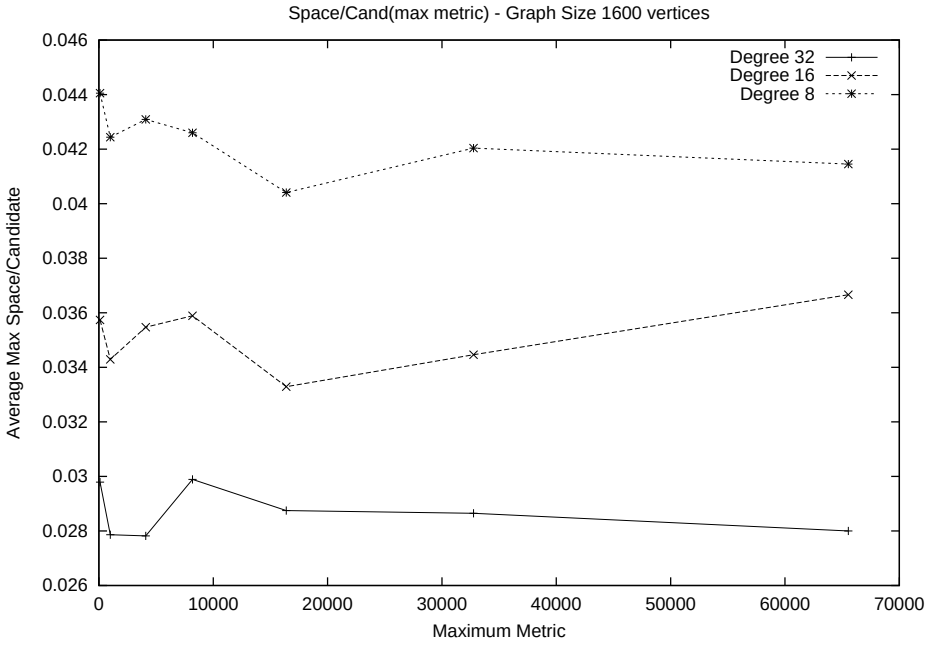

Figure 3.42: Enhanced Normalized Space(Max Met)

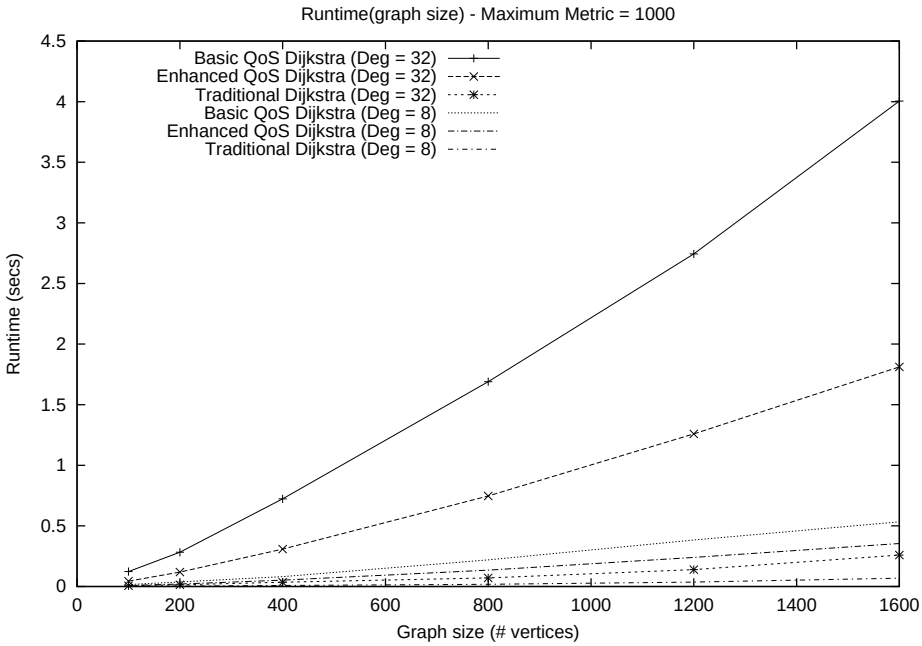

Figure 3.43: Compare Runtime(Size)

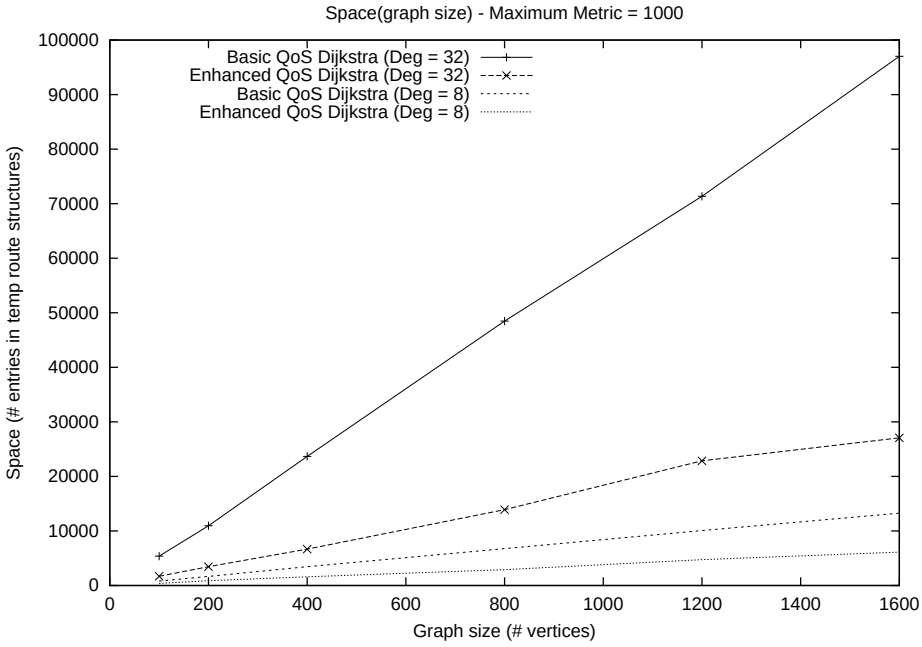

Figure 3.44: Compare Space(Size)

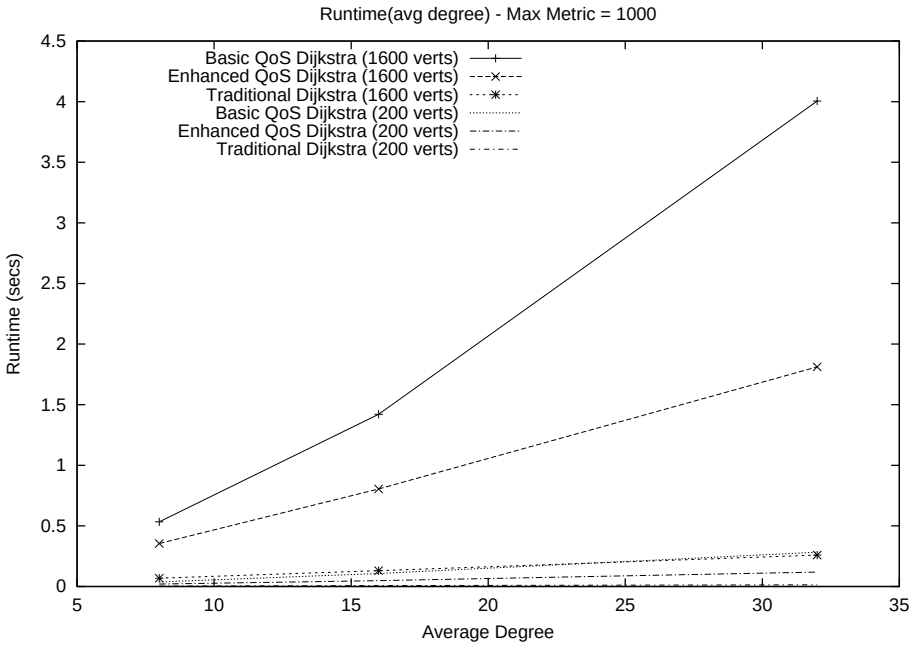

Figure 3.45: Compare Runtime(Avg Degree)

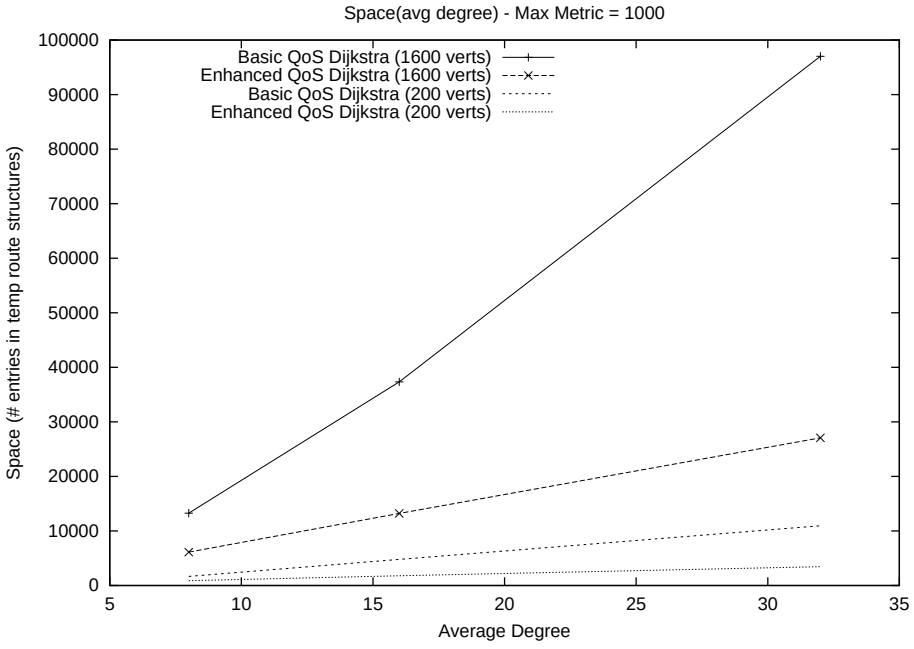

Figure 3.46: Compare Space(Avg Degree)

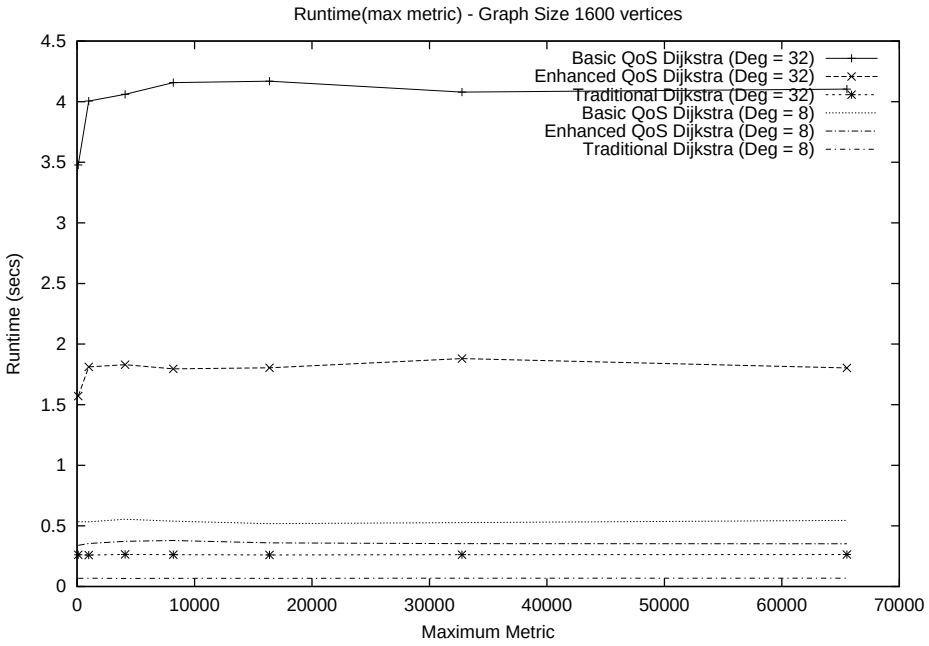

Figure 3.47: Compare Runtime(Maximum Metric)

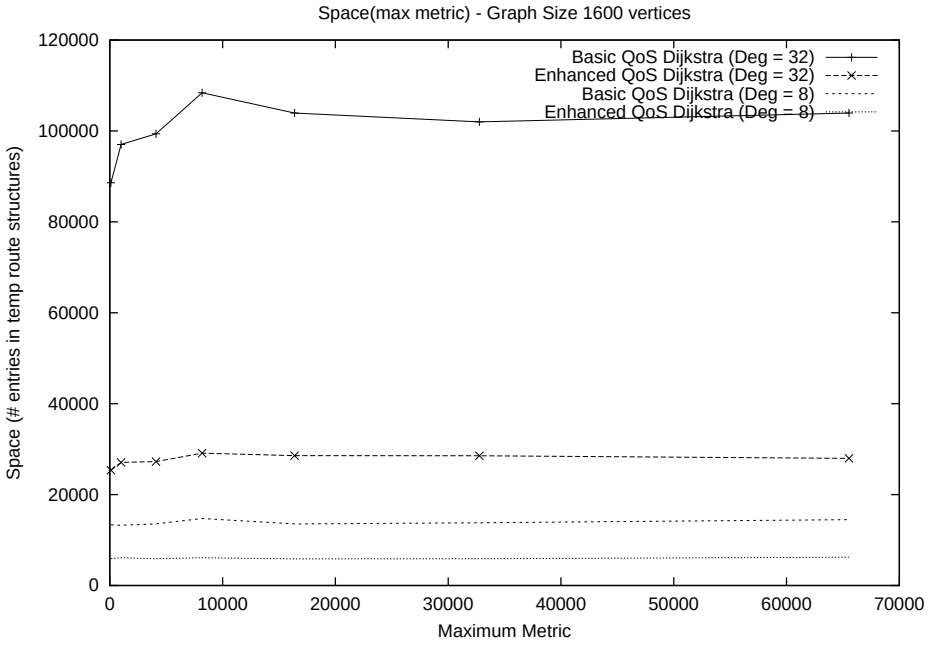

Figure 3.48: Compare Space(Maximum Metric)

3.8 Related Work

As discussed earlier, the first work explicitly addressing the problem of computing exact in the context of multiple metrics is that by Jaffe [42], discussed in Section 1. In addition, work by Chen and Nahrstedt [22] on algorithms for computing approximate solutions can be used, with appropriate parameters, to compute exact solutions, and recent work by Siachalou and Georgiadis [76]. All of these algorithms will only work in the context of performance constraints, and therefore are only useful for satisfying QoS requirements. The Jaffe and Chen algorithms work by pre-allocating maximally sized structures analogous to the TR structures described above, and iterating through the graph searching for the best routes for each TR entry. For example, with weights composed of distance and cost metrics, these solutions setup a cost matrix for each destination $\times$ distance pair, and traverse the graph attempting to fill in each entry with the cost of the best route found in the graph.

The Jaffe algorithm is a variant of Distributed Bellman Ford (DBF), and Chen presents both DBF and Dijkstra based algorithms. The drawbacks of these algorithms are their expected space and time complexity. The expected and worst-case runtime for Jaffe are the same at $O(n^4 b \log nb)$ (the commonly quoted $O(n^5 b \log nb)$ is the cost over the whole network), and the expected and worst-case space is $O(n^2 b)$ (the Jaffe algorithm computes routes between all pairs of nodes in the network). For the Chen algorithm the expected and worst-case values are $O(n^2 W^2)$ runtime and $O(nW)$ space.

In contrast, the *worst-case* performance of the equivalent TD-QoS-Dijkstra

algorithm presented above is $O(nW \log W)$ runtime and $O(nW)$ space, while the average case for typical, sparsely connected Internet topologies is significantly less. The gain provided by TD-QoS-Dijkstra is obtained from the use of efficient mechanisms that limit the growth in space and runtime complexity to only that required to process the given topology. In contrast, the Jaffe and Chen solutions use less adaptable algorithms that must make pessimistic assumptions to work correctly on arbitrary graphs. Unfortunately, the algorithms presented here still have the potential to exhibit exponential behavior on some combinations of topology and link weights. Therefore, the next section explores new solutions for controlling the cost of these computations.

The algorithms recently developed by Siachalou and Georgiadis are very similar to the QoS algorithms presented in Figures 3.5 and 3.30. Similar to the work presented here, their algorithms are generalizations of Dijkstra. Their *Algorithm I* is structured similar to TD-QoS-Dijkstra in Figure 3.5 except that it uses a single heap (H_a) in place of the balanced trees (B_i) and heap (T) of TD-QoS-Dijkstra. The drawback of their data structure is, since heap data structures must be pre-allocated, Algorithm I must pre-allocate heaps to handle the maximum number of routes that can be discovered by the algorithm which can grow to $na_{max}W$. With higher resolution metrics, for example tending towards 32 bit quantities, this requirement becomes prohibitive.

More interesting, however, is their *Algorithm II*. Based on the property of ordered discovery of routes with a given predecessor discussed in detail in Section 3.7, Algorithm II uses similar data structures and algorithm structures to the enhanced TD-

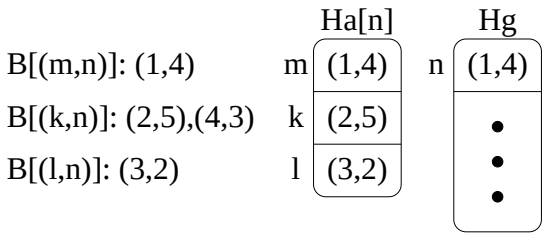

Figure 3.49: Error in Siachalou Algorithm II

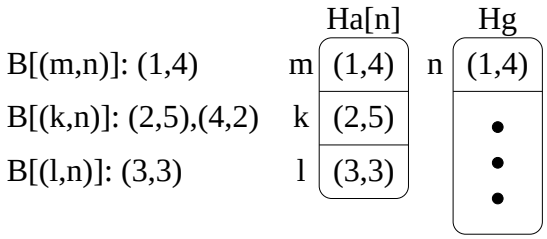

Figure 3.50: Error in Siachalou Algorithm II

QoS-Dijkstra algorithm. However, due to a flaw in Algorithm II's *obtain_minimum*() function, the algorithm is incorrect. Specifically, the textual description of the algorithm describes the purpose of lines 10–27 of the function as that of finding the new element in $H_a[n]$ to replace the entry for n in H_g, and removing all "impossible discontinuities" (equivalent to comparable routes as defined here) with respect to the new value for $H_a[n]$ from the $B[(*, n)]$ queues. This is, in effect, enforcing invariant 4 from Table 3.1. However, the pseudo-code does not correctly accomplish this. Specifically, the while loop at line 19, which is executed in the event that the current candidate for the new value to $H_a[n]$ is not a discontinuity (is $\sqsubseteq$ in terms used here) with respect to the previous value, only drains the queue for the predecessor of this route. As a result, discontinuities (incomparable routes, here) can be dropped, and non-discontinuities (comparable routes) can be kept. Figures 3.49 and 3.50 show two example configuration of the data structures that illustrate this problem. In Fig-

ure 3.49, *obtain_minimum*() replaces $(1,4)$ in H_g with $(4,3)$ when it should replace it with $(3,2)$; similarly, in Figure 3.50 H_g is updated with $(4,2)$ instead of the correct $(3,3)$.

A number of solutions have been proposed for computing exact routes in the context of multiple metrics for special situations. Wang and Crowcroft [87] were the first to present the solution to computing routes in the context of a concave and an additive metric discussed in Section 1. The Wang and Crowcroft algorithm could be modified to work in the context of administrative constraints. Ma and Steenkist [54] presented a modified Bellman-Ford algorithm that computes paths satisfying of delay, delay-jitter, and buffer space constraints in the context of weighted-fair-queuing scheduling algorithms in polynomial time. The Ma and Steenkist algorithm will not work in the context of administrative constraints. Cavendish and Gerla [17] 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. While determining if this condition holds can take exponential time, the algorithm has the property that, independent of whether this condition holds or not, if the algorithm finds a solution, the solution is correct. However, if a solution is not found it is still possible that a solution actually does exist.

Chapter 4

Algorithms for Approximate Solutions

As discussed in Section 1, the problem of computing routes in the context of multiple metrics is NP-complete. Therefore, it is unlikely that an efficient solution exists to the general policy-based routing problem (unless $\mathbf{P} = \mathbf{NP}$). In line with this expectation, the optimal algorithms presented in Section 3 are pseudopolynomial in that they are not polynomial in the length of the input, however they are polynomial in the length of the input and the largest value in the input, which is the average number of incomparable paths to nodes in the graph (given by A, W, or $A \times W$, depending on the algorithm) which will be designated by I in this section. As discussed in Section 1, the significance of these algorithms being pseudopolynomial is that this suggests an approach for controlling the run time complexity of the algorithms by limiting the range of I in the graph's input to the algorithms.

The concept of *incomparable paths* used in the definition of I is based on the link metrics used in a graph, and the fact that, in the context of multiple metrics used for policy-based routing, there is a set of possible path metrics that can only be evaluated in the context of a requested set of metric values; i.e. there is a set of path metrics that are *incomparable*. For example, in the context of link weights comprised of a latency and a cost metric, the two path weights of $10ms/10cents$ and $5ms/15cents$ are not comparable in the absence of a requested set of metrics. In the context of a request for a route with no more than $7ms$ of latency and $20cents$ cost, the first route is "bad" and the second "good". With a request for no more than $20ms$ latency and $20cents$ cost, both routes are "good," and the the first route, being "good enough" and less demanding on the network resources, would be the selected route. Lastly, with a request for no more than $7ms$ and $12cents$, both requests are "bad" and the request should be denied.

Given this concept of incomparable metrics, a higher resolution definition of I is possible where I is the lesser of the maximum number of incomparable path metrics and the actual average number of paths to each node in a graph. Based on this definition, previous algorithms for computing approximate policy-based routing topologies have focused on controlling the maximum number of incomparable path metrics. Both Jaffe [42] and Chen and Nahrstedt [22] 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 routes. Similarly, a number of researchers

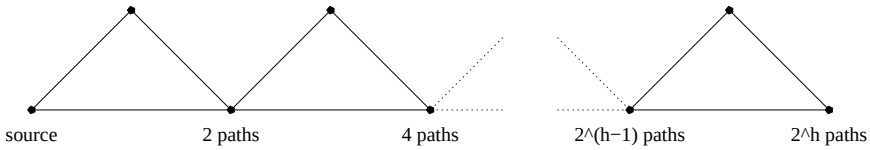

Figure 4.1: Graph with Exponential Number of Paths

[42, 56] have presented algorithms which compute routes based on a function of the multiple metrics comprising a link weight.

The goal of these two approaches is the computation of approximate solutions to the quality-of-service routing problem where the metrics measure performance characteristics of the links. As a result, they share the limitation that they won't work with administrative constraint metrics. Both of these approaches depend on the existence of a total-ordering over the set of possible metric values to allow a mapping of these values onto a smaller metric range while still maintaining the relative ordering of values in the original set (modulo those old values mapped to a single value in the new range). This assumption is not valid for administrative constraints. Therefore, there is no known algorithm for computing approximate routes in the context of administrative constraint metrics.

The approach presented in this section is to focus on the second component in the definition of I, and attempt to control the cost of the routing computation by limiting the number of paths considered in the computation for a given graph. The challenge in this approach is to, as much as possible, eliminate the less desirable paths from consideration while including the better paths. There are a number of measures of the "goodness" of the paths existing in a graph, including robustness (i.e. that all

nodes are still reachable in the presence of link failures) and optimality (i.e. that the best path exists in the set of considered paths). From the perspective of robustness there is anecdotal evidence that the correlation between path count and robustness in a graph is not strong. For example, while the path count in Figure 4.1 is high ($\sum_{i=1}^{h} 2^i$ paths), the robustness of this graph (edge connectivity of 2) is very low.

As illustrated in Figure 4.1, the number of paths in a graph is, worst case, exponential in the number of cycles in the graph. This suggests the strategy of indirectly controlling the number of paths in a graph by limiting the number of cycles in the graph. *Fundamental cycles* are the cycles created by adding an edge to a depth-first tree of a graph. A *fundamental set of cycles* is the set of cycles created by adding edges to a spanning tree of a graph that are in the graph but not in the spanning tree. It has been shown that all cycles in a graph are the sum of a subset of a fundamental set of cycles in the graph. This suggests a further refinement of the strategy for controlling the cost of a policy-based routing computation in the context of administrative constraints of limiting the number of fundamental cycles in a graph. Fundamental cycles are appealing to work with as they are free to count (there are $m - n + 1$ fundamental cycles in a graph), and inexpensive to enumerate (a fundamental set of cycles in a graph is a by-product of a depth-first search of the graph). Such a solution could be applied to either an on-demand or table-driven routing model.

We implemented a simple version of such an algorithm, where edges were deleted from a graph uniformly during a depth-first traversal of the graph. This algorithm was compared with the metric mapping solution proposed by Chen and Nahrst-

edt using the evaluation criteria used in [22]. Specifically, we generated a number of random queries on a random graph, and compared the success rate of the approximation solutions with the success rate from an optimal algorithm for a range of parameters for each approximation solution. The parameter for the Chen and Nahrstedt solution was the coefficient x for this algorithm where larger values for x result in a higher probability of finding a solution, as well as a higher cost of the computation. The parameter used for our new algorithm was the *fundamental cycle density* (FCD), defined as the ratio of the number of FCDs to the number of edges in a graph. Similar to x, larger values of FCD result in higher a probability of success, as well as a higher cost of the computation.

Figure 4.2 compares the run time of the optimal, FCD, and Chen/Nahrstedt algorithms, showing that the run time of FCD approaches that of Chen/Nahrstedt from below as the parameters of the two algorithms are increased towards higher-fidelity computations, and is significantly less than the optimal solution. Figure 4.3 shows that the success rate of the two algorithms are comparable, and approach that of the optimal solution. Figures 4.4 and 4.5 show the performance improvement of the FCD algorithm compared with the optimal QoS algorithm as both a function of the size and average degree of the graph. We interpret these results, from such a simple application of this approach, as very promising. We continue research into more sophisticated implementations of this strategy.

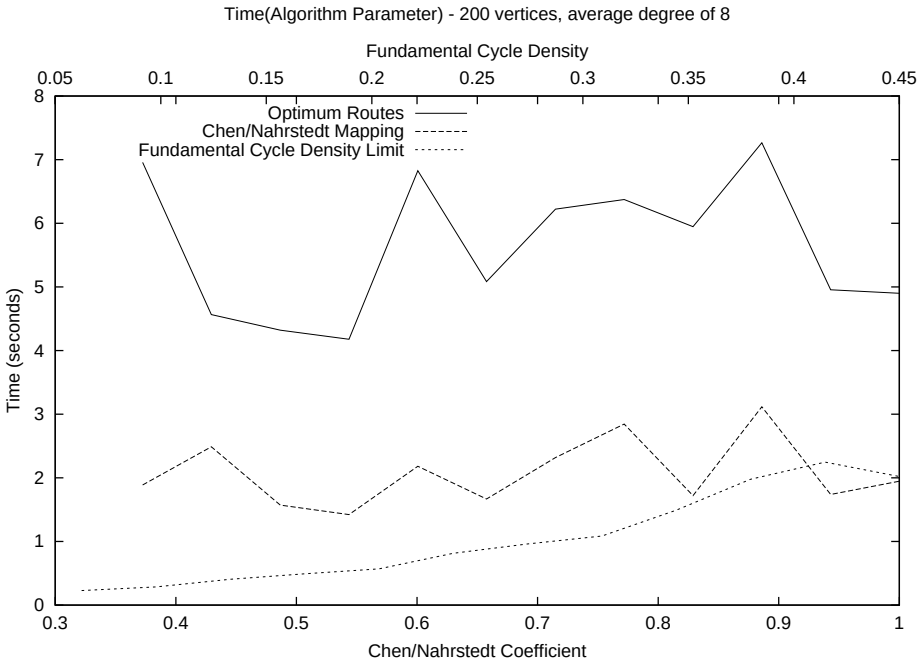

Figure 4.2: FCD vs. Chen Run Time

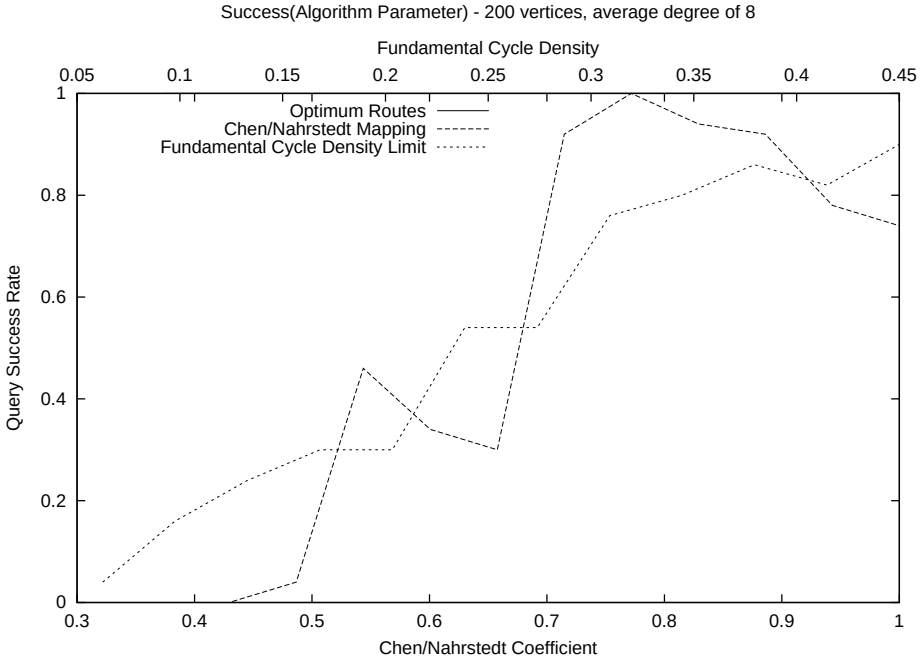

Figure 4.3: FCD vs. Chen Success Rate

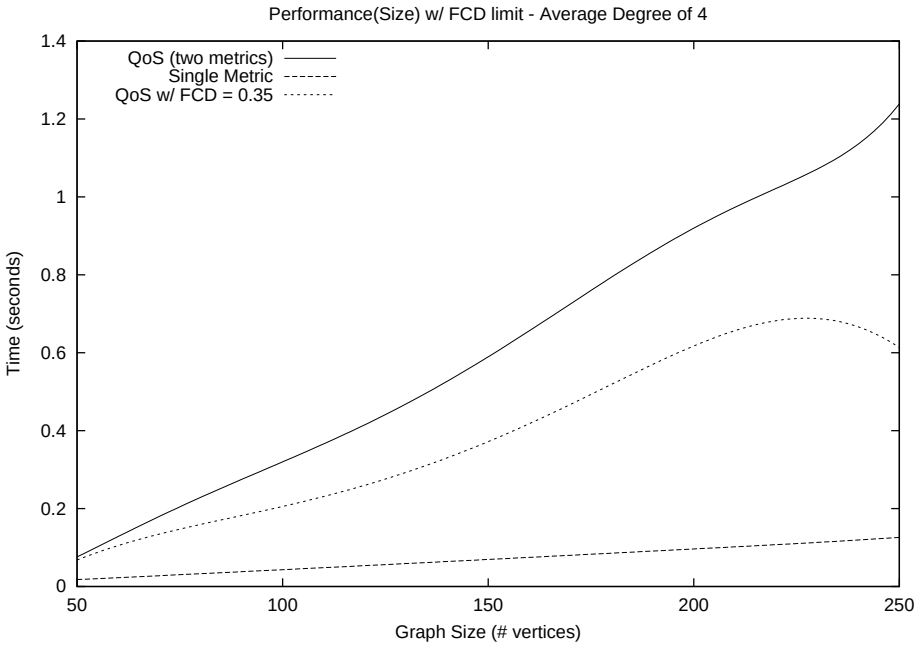

Figure 4.4: QoS with FCD – Performance(Size)

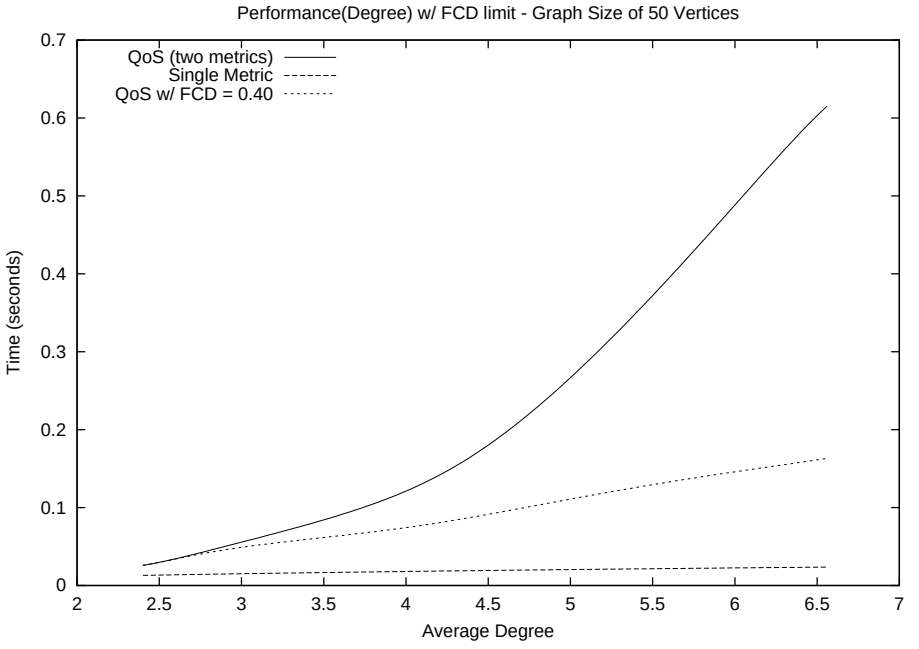

Figure 4.5: QoS with FCD – Performance(Degree)

Chapter 5

Traffic Expression Processing

A critical component of the cost of link predicate routing is evaluating the satisfiability of a path predicate. As explained in Section 2.2, this occurs both explicitly (when deciding whether to consider a new route to a destination) and implicitly in the form of validity testing (when deciding whether traffic served by one route is also fully served by another) in the algorithms presented in Section 3 which perform traffic engineering computations. As observed in Section 2.2, satisfiability is the prototype NP-complete problem. As such, the worst-case time complexity of algorithms that solve the satisfiability problem are not expected to be improved to less than exponential in the size of the tested expression (unless it is shown that $\mathbf{P} = \mathbf{NP}$). However, many strategies have been developed for containing this cost for practical applications.

Additionally, traffic expressions are good candidates for restricted solutions for a number of reasons. First, the syntax of link predicates is open to definition, allowing for the adoption of a restricted syntax that allows for more efficient pro-

cessing. While restricting the traffic expression syntax typically results in restricted expressiveness, this flexibility leaves open the option of customized traffic algebras for different applications which minimize the processing cost while providing the expressiveness required of the application. Second, for many important applications, the rate of change of link expressions can be kept quite low. Third, very simple syntax is used to construct path expressions from link expressions. Specifically, as shown in Figure 3.29, path expressions are built from the conjunction and negation of other expressions. Furthermore, at least one of the component expressions is known to be satisfiable.

Lastly, likely patterns of use of the capabilities could provide opportunities for optimized satisfiability tests. One possible pattern, called "prioritization" here, is to define a hierarchy of traffic classes where higher priority traffic classes have access to a superset of the topology available to lower priority traffic classes. As an example, the following prioritization policy could be defined over the HTTP, SMTP, and FTP services ranging from highest to lowest, respectively:

$$TCP\_PORT(HTTP),$$

$$TCP\_PORT(HTTP \vee SMTP),$$

$$TCP\_PORT(HTTP \vee SMTP \vee FTP).$$

All links in an internet could then be labeled to allow only HTTP traffic, a connected subset of these links could be labeled to allow SMTP traffic, and a connected subset of these SMTP links labeled to allow FTP traffic. As a result, higher priority traffic

would tend to have more robust and better performance service.

Another possible pattern, called "partitioning" here, is to dedicate subsets of a topology to carrying specific traffic classes. The motivation is the same as described in Section 2.2 of managing network bandwidth among multiple services. As an example, the following partition policy could be defined over the HTTP, SMTP, and FTP services:

$$TCP\_PORT(HTTP \wedge \neg SMTP \wedge \neg FTP),$$

$$TCP\_PORT(\neg HTTP \wedge SMTP \wedge \neg FTP),$$

$$TCP\_PORT(\neg HTTP \wedge \neg SMTP \wedge FTP).$$

Important links in an internet (e.g. long-haul links between city-level internets for an ISP) could then be dedicated to one of each of these traffic classes. As a result, the bandwidth demand of the three services could be satisfied under circumstances where it otherwise would not (i.e. when the separate bandwidth of each service is less than that of the link(s) it is assigned to, but the aggregate bandwidth of all services is greater than any one link).

In addition, the context in which satisfiability must be determined for traffic expressions has significant structure which can be exploited to help contain these costs. The remainder of this section reviews the strategies previously identified in the extensive body of research on controlling the cost of the general satisfiability problem, identifies a number of context-specific opportunities for further improving the efficiency of these tests in the processing of traffic expressions, and lastly presents a restricted

solution that allows for very efficient satisfiability testing.

In the following expressions are assumed to be in the standard *conjunctive normal form* (CNF), defined as:

$$(\bigwedge_{i=1}^{m}(\bigvee_{j=1}^{n_i} L_{i,j}))$$

where literals, denoted by $L_{i,j}$, are primitive propositions or negations of primitive propositions. Additionally, the following clausal representation of CNF expressions is used where a clause, which represents a disjunction of literals in a CNF expression, is defined to be a set of literals, and a CNF expression F is represented as a set of clauses:

$$\{\{L_{1,1}, ..., L_{1,n_1}\}, ..., \{L_{m,1}, ..., L_{m,n_m}\}\}.$$

The remainder of this section presents a sampling of strategies from previous satisfiability and proof theory research for optimizing the satisfiability test that appear promising for application to traffic expressions, one particular solution that provides extremely efficient processing of restricted but useful traffic expressions, and simulation results.

5.1 Optimization Strategies from Satisfiability and Proof Theory

Given the ability to define the syntax of link expressions, one possible approach to containing the cost of testing satisfiability of such expressions is to restrict the syntax of these expressions to forms defining relations with efficient algorithms for

computing satisfiability. Significant work has been done along this line, culminating in Schaefer's Dichotomy theorem [69]. Schaefer's theorem comprehensively defines the boundary between relations for which satisfiability can be determined in polynomial time, and those for which it is NP-complete. Specifically, the theorem shows that, for some Boolean expression F, SAT(F) is in P if F is in one of the six Boolean expression classes 0-valid, 1-valid, bijunctive, affine, Horn, or Dual-Horn, and is NP-complete otherwise. Unfortunately for the work here, Schaefer also showed that none of these properties is preserved under negation. As a result, while the properties required of path expressions for polynomial satisfiability testing are maintained by the statements in lines 7, 8, and 12, they are lost by the statements in lines 11 and 16. This leaves a possible strategy for exploiting these results of using a link expression syntax from one of the six classes identified in the theorem, keeping track of those expressions in such a way to maintain the chosen property, and using the polynomial time satisfiability test for those path expressions in which the property still holds.

Given the other characteristics of traffic expressions described above, and the expected use of traffic expressions, a number of other optimization techniques and strategies developed in the extensive research into satisfiability and proof theory over the past 30 years can also be applied to determining the satisfiability of traffic expressions. The remainder of this section presents a set (not exhaustive) of possible techniques from this body of results that can be applied to testing the satisfiability of traffic expressions.

Modern algorithms for testing satisfiability are based on the inference rule

called *resolution.* Fundamentally, resolution is a generalization of the inference rule that yields $(Q \vee R)$ from $(P \rightarrow Q)$ and $((\neg P) \rightarrow R)$ or, equivalently, from $((\neg P) \vee Q)$ and $(P \vee R)$. Resolution is applied by replacing two clauses containing negative and positive literals on the same variable with a single clause containing the remaining literals from the two clauses. The application of resolution to satisfiability is based on the two observations that resolution of a set of clauses preserves the satisfiability of the clauses, and that a set of clauses containing two clauses composed only of negative and positive literals on the same variable (e.g. $\{..., \{Q\}, \{\neg Q\}, ...\}$, called *unit conflict*, is unsatisfiable. Modern satisfiability algorithms work by repeatedly applying resolution to a set of clauses until either a unit conflict is encountered (in which case the original set of clauses was unsatisfiable), or all clauses are consumed (the set of clauses is empty, in which case the original set of clauses was satisfiable). While the use of resolution with one other inference rule (factoring) is sound (they only identify as satisfiable sets of clauses that are actually satisfiable) and complete (all satisfiable expressions are identified as satisfiable by the use of resolution and factoring), alone they do not provide an effective mechanism for satisfiability testing due to the possible cost they incur in coming to a conclusion. The reason for this is the uncontrolled use of resolution can result in a satisfiability computation wandering through unproductive lines of inference in the "theory space" defined by a set of clauses with the result that a conclusion is not reached within the time or space resources available to the computation.

To control this tendency, strategies have been developed for the application

of these inference rules that restrict their application to lines of reasoning expected to be more productive. The most powerful of these strategies is the *set of support strategy* [89]. In this strategy, given a set of clauses C whose satisfiability is to be tested, a subset A of C is identified that is known to be satisfiable (the set A could be thought of as the axioms of the set C), and the remaining set of clauses $S = C - A$ is defined as the *set of support.* Resolution is then restricted to apply only to sets of clauses that include one clause from S, and all resolvents (clauses resulting from the application of resolution) are then added to S. The soundness and completeness of resolution with the set of support strategy depends on the specific form of resolution used (e.g. binary resolution, described above, with factoring is sound and complete with this strategy). The set of support strategy significantly improves the efficiency of such inference rules at finding a solution. Intuitively this makes sense as the restriction of resolution to sets of clauses including elements of the set of support can be seen as a way of directing resolution towards finding a "proof" of the clauses in the set of support.

As for how this strategy can be applied to traffic expressions, the characteristic of these expressions that they are built from more fundamental expressions, at least one of which is known to be satisfiable provides the basis for using this powerful strategy to control the cost of testing the satisfiability of such expressions. Specifically, the construction of traffic expressions performed in line 7 of Figure 12 builds a new traffic expression as the conjunction of two known satisfiable traffic expressions, and those in lines 8, 9, 12, and 13 build new expressions as the conjunction of one known satisfiable expression (in all cases, the left hand expression). Also, note that in the

constructions done at lines 11 and 15 the satisfiability of the resulting expressions have previously been established at lines 9 and 13, respectively (see Section 2.3). Therefore, in testing the satisfiability of a new expression, a set of support can be established with the sub-expression of unknown satisfiability, thereby significantly improving the effectiveness of the test for satisfiability.

Another potentially effective optimization is based on a linear time solution developed by Knuth [49] for a special case of the satisfiability problem he called *nested satisfiability*. Nested satisfiability applies to a set of clauses which have a hierarchical structure. Specifically, given a linear order $<$ on the set of variables in a set of clauses, say one clause C_1 *straddles* another clause C_2 if there are literals σ, τ in C_1 and ξ in C_2 such that $\sigma < \xi < \tau$. Two clauses *overlap* if they straddle each other. For example $\{a, b, d\}$ and $\{a, c, d\}$ overlap, but $\{a, b, d\}$ and $\{a, b, c, d\}$ don't. A set of clauses is called *nested* if no two clauses overlap. A binary, nested relation $\succ$ can be defined on clauses as $C_1 \succ C_2$ if C_1 straddles C_2, but not vice-versa. The term "nested" is a little misleading here in that it implies something a little stronger than simply that the literals of one clause are "inside" those of another; specifically there is also a subset relation implied such that $C_1 \succ C_2$ **iff** the set of variables in C_1 and in the range of the variables in C_2 is a proper subset of those in C_2. Applied to traffic expressions, Knuth's nested satisfiability algorithm has potentially significant application to the prioritization form of traffic expressions where some classes of traffic are given access to greater portions of an internet's topology. For example, given four traffic classes a, b, c, and d, the following set of nested clauses could be used to define prioritization

style policy: $C_g = \{a, d\}, C_s = \{a, b, d\}, C_b = \{a, b, c, d\}$. Clause C_g (for the "gold" traffic classes) could be assigned to all links in an internet, the C_s (for "silver") to a subset of the links in the internet that still defines a connected graph, and C_b (for "bronze") to a similar subset of the C_s labeled links.

Another strategy for controlling the cost of this test is to use dynamic programming techniques. Dynamic programming has been characterized as "recursion with the addition of a caching strategy." These techniques would involve recording the results of tests for reuse in subsequent instances of the same test. Given the recurring nature of edge traversals in routing algorithms, there is the potential for significant savings from the reuse of test results. Additionally, all the above strategies benefit from the relatively static nature of link predicates which allows significant pre-processing work to be amortized over many subsequent routing computations.

5.2 An Efficient, Restricted Solution

An efficient solution to the SAT problem involves exploiting the isomorphism of set algebras and boolean algebras 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 this solution, each forwarding class defined by a single row in the boolean algebra's truth table is represented as an element in a set (e.g. as a bit in a bitmap). The boolean operations $\wedge, \vee$, and $\neg$ can then be mapped to the set operations $\cap, \cup$, and $\neg$ (which, for a bitmap, can be efficiently implemented as bitwise *and, or,* and *complement*). Similarly, satisfiability of a boolean expression translates

to testing for a non-empty set (and a non-zero bitmap). All of these operations can be implemented very efficiently for a small number of forwarding classes.

In this solution, the set can be interpreted in two ways. Either as a boolean algebra, with a small number of variables (e.g. sixteen variables defines 2^{16}, or 64K forwarding classes, which can be represented in an 8KB bitmap data structure). Alternatively, as a set of forwarding classes whose definitions are universally known by routers in a routing domain. The benefits of this solution are that it has a very efficient implementation, and maps reasonably well to the requirements of modest policy-based routing applications. The primary drawback is its limited scalability in terms of the number of traffic classes, and therefore traffic algebra variables it can efficiently support. For the full power of this traffic algebra approach to traffic engineering, less constrained solutions must be found.

Chapter 6

Intra-Domain Policy Routing Applications

This chapter presents a number of applications of the policy-based routing technologies presented above to routing systems deployed in the Internet today.

6.1 Unicast

The major component of these systems is a routing protocol whose function is to communicate connectivity information among the routing processes in an internet for use in computing routes for reaching existing destinations. The routing protocols used in most of today's computer networks are based on shortest-path algorithms that can be classified as distance-vector or link-state. In a distance-vector algorithm, a node knows the length of the shortest path from each neighbor node to every network

destination, and uses this information to compute the shortest path and next node in the path to each destination. A node sends update messages to its neighbors, who in turn process the messages and send messages of their own if needed. Each update message contains a vector of one or more entries, each of which specifies, as a minimum, the distance to a given destination. These route vectors implicitly describes the topology of the subset of the internet actually used by the routing process in forwarding traffic.

In contrast, in a link-state algorithm a node must know the entire network topology, or at least receive such information, to compute the shortest path to each network destination. Each node broadcasts update messages, containing the state of each of the node's adjacent links, to every other node in the network [34]. The difference between these protocols has been characterized as "in distance-vector algorithms a router sends information about the whole network to its neighbors; in link-state algorithms a router send information about its neighbors to the whole network" [60]. In aggregate, the link-state updates explicitly describe the complete topology of the internet.

From a performance perspective, the differences between these two classes of protocols revolve around what information must be exchanged in response to a link cost change, and how far must this information travel. In a distance-vector protocol information is exchanged describing the reachability of destinations and, in response to a link cost change, a router attached to the affected link must transmit information describing all destinations downstream from the changed link. In a link-state protocol

information is exchanged describing the state of a link in the internet and, in response to a link cost change, only information describing the link must be transmitted. Conversely, in a distance-vector protocol the transmitted information is only propagated to routing processes whose state may change based on the information, while in a link-state protocol link-state updates must be propagated to all routers in the internet. Therefore, while distance-vector protocols typically must transmit much more information in response to a link cost change, the propagation of this information is far more constrained in comparison with link-state protocols.

Based on this insight, a hybrid class of routing algorithms, called link-vector [35] has recently been defined which combines the strengths of the distance-vector and link-state classes of algorithms. In this class of protocols, similar to link-state protocols, only information describing changes to the state of links in an internet is exchanged; furthermore, similar to distance-vector protocols, this information is only propagated to routers that might use the information. As a result, in a link-vector routing system, the routing processes only carry information about the subset of the internet topology they actually use to forward traffic. This gives rise to another classification of link-state-based protocols as *complete* vs. *partial topology* protocols.

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 that exchanges information describing link state. Note, however, that for a system de-

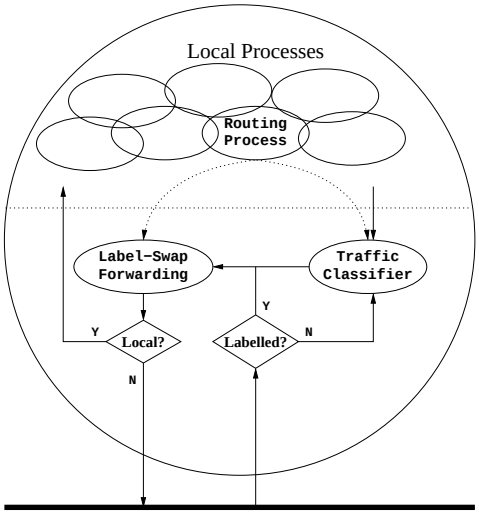

Figure 6.1: Traffic Flow in Policy-Enable Router

pending on on-demand routing computations a link-state, complete topology 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 link-vector, partial topology protocols as each routing process is ensured of learning from its neighbors of all links composing optimal routes to all destinations in the internet.

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 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. While this provides sufficient coverage for any internet (in the sense that a traffic classifier will be in place at all possible sources of traffic in an internet) it

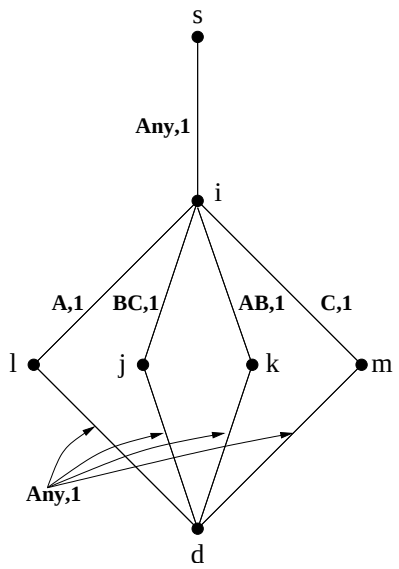

Figure 6.2: Next Hop Problem with Policy-Based Routing

may still be desirable to provide additional traffic classifiers (e.g. in switches) to distribute processing load. Additionally, traffic classifiers must have current definitions of primitive propositions used in traffic expressions, and the current mapping from traffic expressions to local labels for the ingress router. As illustrated in Figure 6.1, 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.

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 be assured of matching an active route in the selected next hop neighbor. As illustrated in Figure 6.2, 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 6.3 presents an enhanced version of the TD-TE-Dijkstra 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

```
algorithm Hop-by-Hop-TD-TE-Dijkstra
    begin
1   Push(<s, s, 0, 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    if (¬(ε_i → E_i))
        then begin
11        Push(<i, p_i, ω_i, ε_i>, P_i);
12        E_i → E_i ∧ ε_i;
13        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) ∧ ¬((ε_i ∧ ε_ij) → E_j)}
            begin
14          ω_j ← ω_i + ω_ij; ε_j ← ε_i ∧ ε_ij;
15          if (T_j = ∅)
16            then Insert(<j, i, ω_j, ε_j, n_i, l'_j>, T)
17          else if (ω_j < T_j.ω)
18            then DecreaseKey(<j, i, ω_j, ε_j, n_i, l'_j>, T);
19          Insert(<j, i, ω_j, ε_j, n_i, l'_j>, B_j);
            end
        end
    end
```

Figure 6.3: Hop-by-Hop TD-TE-Dijkstra.

hop to the predecessor to the path's destination. For example, from Figure 6.2, 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).

6.2 Multicast

As discussed in Section 2.2, an important application of traffic engineering technologies is the control of forwarding topologies to protect against the denial of network services, and the disclosure of network traffic. The network services can be denied by the forwarding of unauthorized traffic over regions of an internet, resulting in the compromise of policies regarding the use of network resources. The current Internet model of universal, undifferentiated access of network traffic to internet topology means every portion of an internet is exposed to such misuse from traffic sourced from anywhere in the internet. While recent DDOS attacks dramatically illustrate this vulnerability, protection against denial of services is not limited to protection against active attacks. Denial of service protection is also important for controlling the allocation of resources in an internet. For example, controlling access to portions of an internet topology among classes of users (e.g. the differentiation among "Bronze, Silver," and "Gold" customers), or among types of users (e.g. academic versus administrative users in an academic environment) would be a very powerful tool for a network service provider.

Similarly, disclosure of network traffic has the same cause (forwarding of network traffic over unauthorized regions of an internet), but with the compromise being of policies regarding access to the traffic rather than use of network resources. Again, the undifferentiated traffic forwarding model of today's Internet provides no means for controlling the distributed of traffic in an internet. As a result, either through accidential mis-configuration of an evolving internet, or through the active

attack and compromise of an internet's routing computation, sensitive information can be forwarded over inadequately protected infrastructure. Clearly the ability to control the topology used to forward traffic in an internet would provide important benefits to the Internet architecture, in general. However, closer analysis shows that the multicast services defined in the Internet architecture are critically handicapped without such capabilities.

The IP multicast model [26] is based on the notion of a *host group* identified by an IP multicast address. Multicast packets are delivered to all members of the host group specified by the destination multicast address. An *open* group model is assumed in that the sender does not know the membership of the group. Hosts can join and leave at will by communicating with a nearby multicast router. Multicast routers collectively compute efficient multicast forwarding trees, and forward multicast packets over these trees to all members of the destination host group.

The multicast model has several advantages over the alternative unicast and broadcast models for communicating with members of a host group. By building multicast forwarding trees that efficiently connect all host group members the multicast model avoids the overhead of transmitting packets over unnecessary links that would be incurred in using a broadcast service. By using a single forwarding topology for all traffic to host group members the multicast model avoids the redundancy of transmitting the same data over a given link multiple times that would be incurred in the use of a unicast service. Lastly, by using a single multicast address to identify all members of a host group the multicast model avoids the transmission overhead of per-group

member address state that would be incurred by the sender in the use of a unicast service.

A number of important applications can take advantage of the efficiency and scalability of multicast services. Live audio and video distribution, referred to as web-casting, involve a single source transmitting real-time audio or video to multiple receivers across an internet. Data push applications involve a single source transmitting information to a large audience or subscriber base. Examples of data push include stock tickers, PointCast-style news push services, software distribution, network news distribution, etc. Interactive content distribution, in the form of conferencing and group collaboration applications, involve the bi-directional use of the web-casting and data push technologies to allow interactive communication over an internet in forms such as interactive games, audio and video conferencing, etc.

In spite of the clear benefits provided to these (and similar) important applications, multicast services have seen only limited deployment as a production service in the Internet. A number of reasons for this limited deployment have been identified [28] including the lack of mechanisms for controlling the allocation and use of multicast addresses, the lack of commercial network management tools, the lack of billing mechanisms, and the lack of mechanisms for securing multicast communications.

However, on closer analysis of the basic multicast model, more fundamental limitations come to light. Specifically, the multicast model defines few requirements of the topologies to be used for multicast traffic forwarding. The only requirements implied by the model are that the topologies used for multicast traffic must reach

all group members with no redundant use of any individual links, and must optimize some metric (e.g. latency). This single class forwarding model, combined with the open group model used by IP multicast, is crippling for multicast services in that it cedes control of the topology computation to the user, rendering a multicast deployment inherently un-controllable. As a result the topology used for a given multicast destination can't be controlled without, in general, denying service to authorized recipients of the traffic, leaving multicast services inherently vulnerable to denial of service and disclosure of multicast traffic. The remainder of this section identifies the weaknesses of the multicast model, and develops a solution to them based on the traffic engineering capabilities developed in previous sections.

6.2.1 Symptom – IP Multicast is Expensive to Manage

The unusual one-to-many nature of multicast communication poses a management problem for those wishing to deploy multicast services. A number of efforts have been made to document effective techniques and tools for managing such services [3, 68, 83]. In general, these works identify the kinds of problems experienced by the users of multicast services, present troubleshooting techniques and strategies for identifying the underlying causes of the problems, and suggest courses of action for eliminating the causes or mitigating their effects.

While the various proposals differ in details, they are similar in their general outline. At the highest level, troubleshooting a problem with multicast services involves determining if the underlying cause is a configuration error or traffic conges-

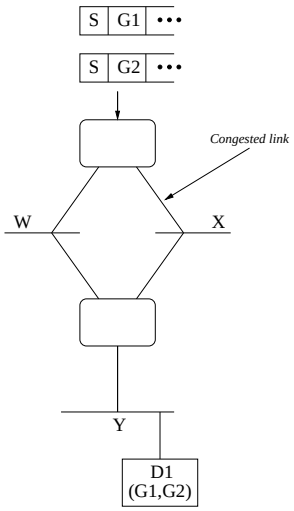

Figure 6.4: Multicast Traffic Control

tion. Configuration problems can include lack of support of the necessary protocols or the necessary versions of the protocols, traffic scoping problems, network configuration errors (e.g. NAT services blocking multicast traffic), etc. While identifying these problems may be difficult due to the tree-based nature of multicast services (as compared with the path-based nature of unicast services), resolving them once identified is straightforward; software is upgraded, users are educated, or configurations are changed.

In contrast, congestion problems are equally difficult to identify, but often impossible to resolve without denying services to some set of users. Once a congested link and the problem traffic have been identified a mechanism is needed for eliminating the traffic from the link. Fundamentally this means the link must be removed from the tree used for distributing traffic for the targeted group (or source/group). The reason the link is on the tree is that the router at one end of the link is seen by the router at

the other end of the link as the next hop to the core or source of the targeted traffic. To remove this link from the tree we must change the topology of the corresponding unicast routing tree. In general, this is not possible. Therefore, to relieve congestion on a link due to multicast traffic, it is necessary to reduce the volume of traffic for the targeted group (or source/group). This reduction in traffic is required in spite of the possible availability of bandwidth on an alternative path adequate to satisfy the un-constrained demands of the multicast users (see Figure 6.4).

Therefore, in general, the only way to manage the bandwidth demands on a multicast system is to deny services to some of the users of the system. This is true even in the case where there is adequate bandwidth in the internet for the offered load to the system. To avoid this limitation an alternative to the underlying unicast routing protocols is needed for topology discovery that allows network administrators to specify what traffic is allowed on which links, and computes the shortest path to a source or core *that is authorized to carry traffic for a specified group (or source/group).*

6.2.2 Symptom – IP Multicast is not Secure

The acute vulnerability of the current infrastructure for Internet multicasting has been recognized for some time, and many opportunities for malicious attacks or misuse have been identified. To date, the efforts to secure multicast services have resulted in solutions that, while innovative, fall short of adequately protecting the Internet's infrastructure. The purpose of a threat model [8, 23, 70], is to provide a framework for the identification of threats and countermeasures. A *threat* to a protocol

is defined by the actions taken to exploit vulnerabilities in the protocol, called the *attack*, and the *loss* resulting from an attack. A *countermeasure* is a mechanism that eliminates or mitigates the loss resulting from an attack.

A threat model identifies what resources in a system need to be protected, who the principals are in the system with authorized access to these resources, the capabilities expected of intruders in the system, and the security requirements of these resources. The *resources* are the elements of a system whose access by an intruder results in loss. The *principals* are the active entities in the system that have authorized access to the controlled resources. Together, principals and resources are, respectively, the subjects and objects of security policies. All controls desired in a system must be describable as an action performed by a principal on a resource. Careful thought must be given to defining these sets as they effectively define the world that can be secured.

Intruder's capabilities are the set of actions intruders are assumed to be capable of performing that have the potential to cause loss to a system. Typically intruders are assumed to be able to: position themselves at a point in the network through which all traffic of interest will pass; fabricate, monitor, delete, or replay messages; and subvert any principal in the protocol dialog, thereby obtaining cryptographic material it may possess.

Security requirements are the properties of a system whose compromise results in a loss. The desired properties include confidentiality, integrity, authenticity, non-repudiation, authorization, and availability [74]. *Confidentiality* is the protection of data so it is not made available or disclosed to unauthorized individuals, entities, or

processes. *Integrity* is the property that data has not been altered in transmission; that the data received is the same as the data sent. *Authentication* is the property of a system that the identity claimed by principals in the system is securely verified. *Non-repudiation* is the property of a system that principals cannot falsely repudiate a communication (i.e. a principal cannot deny authorship of a message it actually authored). *Authorization* is the property of a system that access by principals to resources in the system is authorized in the sense that it is allowed by the system's current security policies. Lastly, *availability* is the property of a system that resources in the system are accessible and usable upon demand by an authorized principal.

Security countermeasures involve the application of cryptographic and access control mechanisms to eliminate or mitigate losses resulting from attacks. *Confidentiality* is typically provided by encryption. *Integrity* and *authenticity* are provided by encryption, message authentication codes (MACs), or digital signatures [71]. Non-repudiation is provided by digital signatures. *Authorization* is provided by an access control mechanism such as access control lists (ACLs). *Availability* is provided by redundant infrastructure, and the enforcement of resource usage policies by the access control mechanism.

The group nature of multicast systems results in additional dimensions to the traditional security requirements; specifically authentication and confidentiality [57]. In multicast systems authentication is further differentiated into group and source authentication. *Group authentication* is the property that, while the specific identity of a principal is not verified, the membership of the principal in the group is. *Source*

authentication is the more traditional property that the principal's identity is verified. Similarly, confidentiality must be further differentiated to address the dynamic nature of the set of recipients of a multicast transmission, which changes over time. We call this *current membership confidentiality* (CMC). CMC can be further classified depending on whether the sequence of audiences need to be strictly partitioned such that confidentiality is established precisely on a join or leave, or within some period of time after a join or leave. We call these *strict* and *loose* CMC, respectively. Lastly, with these epochs introduced in the cryptographic key state by CMC, protection must be provided from compromise of a past epoch allowing an intruder access to all future epochs. This is called *perfect forward secrecy* (PFS) [40].

The *confidentiality* requirements specific to multicast are provided through the careful re-keying of authorized group members. Loose CMC is typically provided by re-keying the group on the join or leave of a member. Re-keying on a join is typically much easier than on a leave because following a join the existing group key can be used to distribute the new group key to existing members, and a unicast transmission can be done to transmit the new key to the new member. For loose CMC on a leave, however, the old key cannot be used because it is held by the leaving member. Instead, the new key must be distributed without the benefit of the existing group key. A number of innovative and efficient mechanisms have been developed to address this problem [16, 21, 38, 86, 88].

Providing strict CMC is more complicated. While strict CMC can be provided on a join through careful re-keying (by ensuring the new key is not sent to the

new member until the reliable multicast to the existing members is complete), no solution for strict CMC on a leave based solely on re-keying has been proposed. One proposed mechanism that goes beyond re-keying is to have normal communication require senders to encrypt data with an individual key they share only with a group controller, and send the message unicast to the group controller; the group controller then multicasts the message to the group using the current group key [57]. Using this solution group traffic can be transitioned to the new group key immediately on a decision to re-key. PFS is typically provided by a periodic "refresh" re-keying that breaks the chain of key information developed in the join and leave re-keying schemes referenced above. Special mechanisms have also been proposed for providing source authentication in a multicast environment [14].

A large body of recent work has focused on end-to-end security services [13, 14, 57]; the services provided in these solutions include group management or access control to authorize host access to multicast groups, the distribution of keys to authorized hosts, and the encryption of data traffic by hosts. A smaller body of work has attempted to address the security of the multicast network infrastructure [7, 73]; the additional services provided by these solutions include access control to authorize the use of network bandwidth by data traffic, the distribution of data traffic in an internet, and the cryptographic protection of multicast protocol control traffic between hosts and routers and among routers themselves.

These prior end-to-end and network infrastructure approaches to providing security in Internet multicasting are significant contributions; however, they all have

significant vulnerabilities that must be addressed for use in a production environment.

End-to-End Threat Model.

Most work to date on securing multicast services [15, 39, 57] has focused on providing end-to-end confidentiality, integrity, and authentication of the multicast traffic itself by encryption using a group key shared by all members of the group. Access to this group key is controlled by requiring potential group members to obtain authorization from an access control server (ACS) before being included in the distribution of group keys. The primary requirement of these countermeasures is a scalable, reliable re-keying mechanism. A number of innovative and efficient mechanisms have been developed to minimize the number of encryptions and messages required to re-key a set of users [16, 21, 86, 88]. In general these mechanisms work by computing a hierarchy of keys to cover the members of a group such that there is a one-to-one correspondence between the leaf nodes of the key tree and users in the multicast group. On authorizing itself with the group's access control server, a new member of a group is given the set of keys forming the path from the leaf node corresponding to the new user to the root of the tree. By the careful use of these keys it has been shown that re-keying costs scale with $log_d(n)$ (where d is the degree of the key tree, and n is the number of group members). Since key distribution is done by multicast, reliability of re-keying must be provided by reliable multicast [53].

Uniform Infrastructure Threat Model.

Some previous work has attempted to provide security of the routers composing the network infrastructure [7, 14, 39, 73]. The motivation for securing the network infrastructure can be broken down to two observations. First, the extent of the distribution of traffic for a multicast group in an internet *is* a security concern. The distribution of traffic, even encrypted traffic, to unauthorized subsets of the routers and hosts in an internet poses potentially serious denial-of-service and disclosure threats. For example, if it is not possible to prevent a host from joining a group at the routing level (independent of whether it can obtain the key for the group) then it is possible for any host, given the availability of a large enough number of multicast groups, to overload the network infrastructure in its region of an internet by joining (via the transmission of IGMP Report messages [33]) as many groups as is necessary to cause failure of the network. The only effect of the end-to-end countermeasures described above would be that the attacker will not be able to read the traffic, which does not stop the damage done to the services provided on that internet. Similarly, for some applications, the uncontrolled distribution of multicast traffic, even encrypted traffic, in an internet can pose a serious disclosure threat. If sensitive information is available from the analysis of encrypted traffic for an application, the the ability to obtain encrypted traffic for that application would allow an attacker to obtain such information from any point in the internet. These vulnerabilities make existing multicast systems un-deployable in any production network environment.

The second observation is that, due to the architecture of IP multicasting

services, the only entities able to control the extent of distribution of multicast traffic in an internet are the multicast routers. A host joins a multicast group by sending IGMP Report messages that include the desired group on a directly attached LAN. Routers on the LAN receive these messages, notice the new group desired on that LAN, and join the group using mechanisms specific to the multicast routing protocol. The join process followed by the routers typically only involves other routers, and doesn't require interaction with other principals in the multicast group such as a source for the group or the group creator. This anonymous style of group join has very good scaling properties, unfortunately it leaves only the routers as entities able to enforce any access control restrictions on the extent of distribution of multicast traffic in an internet.

From these observations it is clear that the ability to secure interactions involving the multicast routers is a requirement for many important multicast applications. To provide these protections previous proposals assumed that all routers and LANs composing the infrastructure of a network are authorized to carry traffic for all groups, and proposed countermeasures that limit construction of multicast trees so they are only built between authorized senders and receivers.

Differentiated Infrastructure Threat Model.

Both of the previous proposals assume the routing infrastructure as a whole is trusted for all groups, and security is defined as ensuring traffic is only forwarded between authorized sources and receivers. While this assumption and this definition

of security are adequate for many denial of service and disclosure threats, they are critically inadequate for many other important classes of such threats.

Denial of service occurs when unauthorized use of a resource results in its not being available for authorized use. The proposals presented above protect against the simplest scenario for such abuse where traffic for all multicast groups is authorized to consume bandwidth on all components of the network infrastructure, and authorized use of the infrastructure is based solely on a user's rights for membership in *any* group carried in the internet. However, there are many finer-grained security policies where traffic for a given group may only be carried by a subset of the network's infrastructure, and authorized use of the infrastructure is based on the particular traffic being transported. For example, in an environment where subsets of the infrastructure are deployed for specific uses along organizational lines (e.g. sales, engineering, human resources, etc. in a corporate environment), or along functional lines (e.g. class delivery, administration, specific research projects, etc. in an academic environment), and multicast groups are authorized to use the appropriate subsets of the infrastructure. In an environment with such security policies the proposed solutions would not mitigate the very significant threats of e.g. a spike in class delivery use of the network causing the complete denial of service of a given research project's use of its subset of the infrastructure. The number of ways to subset network infrastructure in this manner is only limited by imagination.

In a related sense, the assumption and definition of security used by previous proposals do not address threats of disclosure that are significant for some applications.

For example, in a military environment where information is assigned one of a hierarchy of security levels (e.g. un-classified, secret, top secret, etc.), the network infrastructure must be classified as to the highest level of traffic it is authorized to carry. In such an environment the transmission of traffic at a given level (e.g. top secret) over segments of the network rated below that level (e.g. un-classified) represents a serious disclosure attack, independent of whether the traffic is encrypted. In the previous proposals it would be possible for top-secret traffic to be routed over un-classified network infrastructure.

To address these significant remaining vulnerabilities the uniform network infrastructure assumptions must be replaced with the enhanced set of assumptions that only a subset of the network infrastructure is authorized to carry traffic for a given group, and security is provided by ensuring multicast traffic is only forwarded between authorized sources and receivers over authorized network infrastructure.

6.2.3 The Problem

The fundamental problem underlying the symptoms discussed above is that there is an incomplete understanding of the naming requirements of multi-destination communication. As a result, existing multicast systems do not provide adequate means for controlling the topology used to forward multicast traffic in an internet. Services based on these systems are impossible to effectively control and therefore are not manageable and inherently in-secure. Using Saltzer's model discussed in Section 2.4 to do a careful analysis of multicast communication the underlying cause of the symptoms

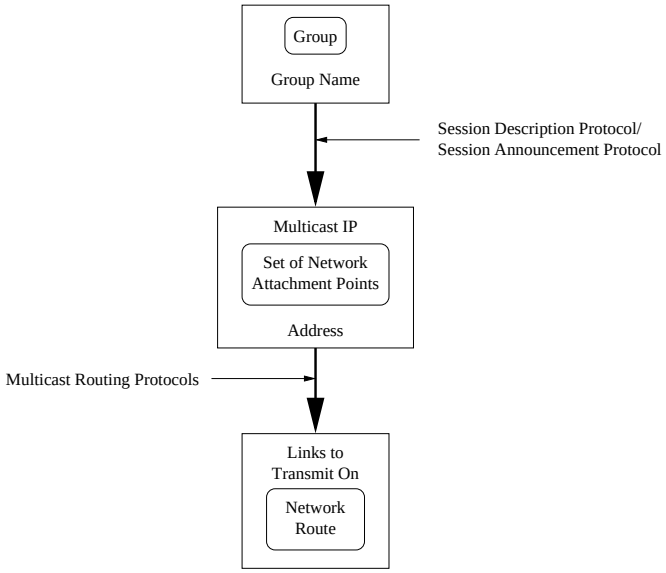

Figure 6.5: Naming Requirements of Multicast Internet Communication

described above becomes evident. Ignoring the issues relating to network attachment points and and their binding (which, generally, are trivial in multicast communication), the naming requirements of multicast communication can be described as shown in Figure 6.5.

Recalling that the problems described in the previous sections manifested themselves as an inability to control the topology used for multicast communication, we focus on the network attachment point to route binding function which determines the topology used by network traffic, and compare the unicast and multicast routing functions. In both instances the inputs to the routing computations are the topology of the internet and the location of addresses in that topology. To control the forwarding traversed by network traffic it must be possible to control both of these inputs to the routing function.

In unicast communication the topology of an internet and the distribution

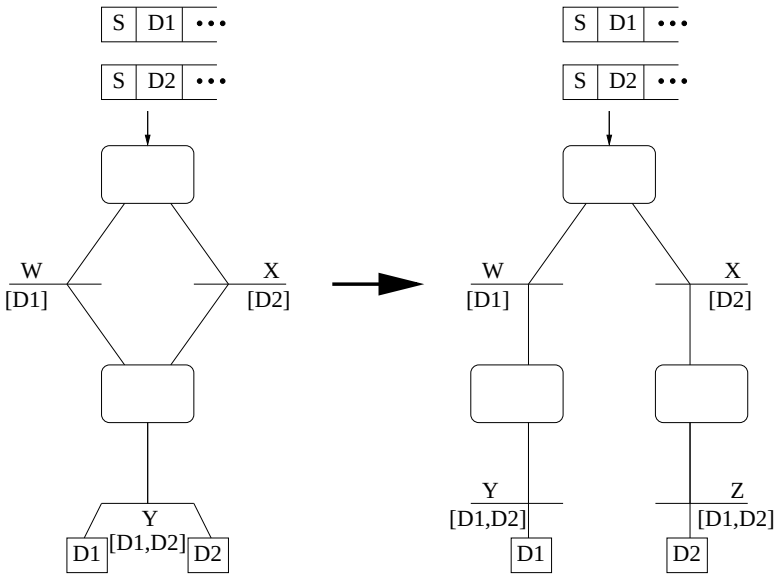

Figure 6.6: Unicast Traffic Control

of addresses in that topology are controlled by the network manager. This provides a basic mechanism for controlling the topology traversed by unicast traffic in an internet. For example, assume in the topology on the left of Figure 6.6 the routes from S to the subnet containing $D1$ and $D2$ go through subnet X. If subnet X becomes overloaded the network manager can correct this by re-organizing the topology as shown in the right-hand of the figure.

In contrast, in multicast communication the distribution of multicast group addresses in the topology of an internet is not controlled by the network manager. Rather it is determined by the users of the multicast service, and what groups they choose to join. This loss of control cripples the viability of production multicast services. For example, assume in the topology on the left of Figure 6.7 multicast traffic from source S to groups $G1$ and $G2$ go through subnet X (as determined by the unicast routing tables used to construct the reverse-path forwarding tree). If subnet

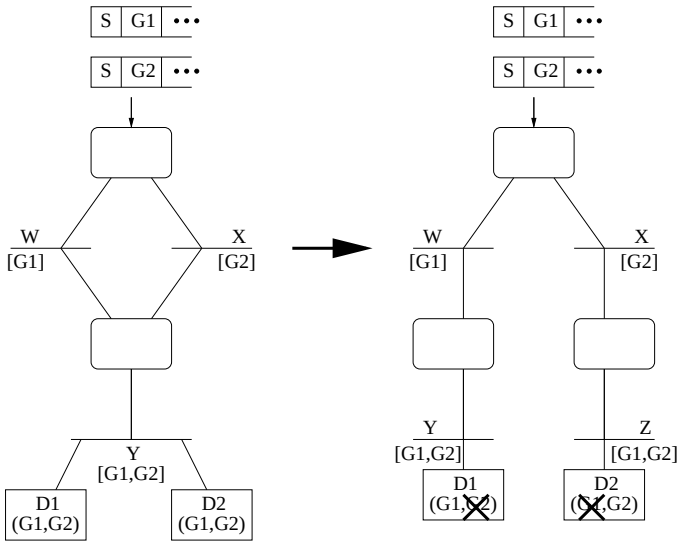

Figure 6.7: Multicast Traffic Control

X becomes overloaded there is no way to correct the problem without denying service to one of the hosts $D1$ or $D2$. This is directly a result of the incomplete control of the multicast address to route binding in the multicast architecture.

As a result, among the requirements of an internet multicast communication service are the requirements that availability of the service is not required (because of the possible need to deny service to a subset of the authorized receivers), and that confidentiality with respect to traffic analysis is not required. These requirements limit the use of current Internet multicast services to prototype, "toy" deployments. To meet the requirements of a production, infrastructure-grade multicast deployment the multicast architecture must provide a means for controlling the topology used for forwarding multicast traffic in an internet. The remainder of this section presents a proposal for such a mechanism.

6.2.4 The Solution

Previous sections have identified the need to support the specification and enforcement of policies that authorize subsets of a network infrastructure to carry traffic for a given multicast group. Unfortunately, such policies pose a fundamental conflict with the assumptions underlying the design of modern multicast routing protocols.

Modern multicast routing protocols depend on the unicast routing tables to build shortest-path, loop-free trees for the distribution of data and control traffic for multicast groups. However, given the assumption presented above that only a subset of the network infrastructure is authorized to carry traffic for a given group, it is now possible that the link shared with the parent router selected by a new router to a group based on the unicast routing tables may not be authorized to carry traffic for the group. Depending on the multicast routing protocol in use this failure to build a tree for the new group will manifest itself in different ways with the end result of the denial of multicast service to portions of the internet.

As an example, the reverse-path forwarding mechanism used by sender-initiated multicast routing protocols (e.g. PIM Dense Mode [25]), whereby the unicast next hop router towards a new source for a group (specified as (S, G)) is selected as the parent of a given router, does not work in a secured multicast environment. Specifically, if a router picks a parent for an (S, G) that is not authorized to carry traffic for G then that router and its descendents will not receive traffic from that source. For this scheme to work the security policies must be the same among all multicast groups and the unicast routing; effectively no security at all. To receive (S, G) traffic a router

must select a parent that is authorized to carry traffic for the group. Similarly, the parent selection mechanism used in receiver-initiated multicast routing protocols (e.g. CBT [6], OCBT [30], PIM Sparse Mode [72]) suffers from problems similar to those of the sender-initiated reverse-path check mechanism described above. When selecting the parent for a group, these protocols choose the unicast next hop router towards the group rendezvous-point or core. Join requests are sent to the parent, join acks and other control messages are only accepted from the parent, and data is only forwarded over these links. Selection of a parent not authorized for the group will cause joins to fail because the parent won't be trusted to forward the message.

Therefore, support of such differentiated infrastructure policies requires an enhancement of the basic multicast model. Specifically, the topology discovery mechanism used in multicast systems must compute forwarding topologies for a group *that are authorized to carry traffic for that group.* In terms of Section 6.2.3 this requires a policy input to the routing computation that specifies what links in an internet are authorized to carry traffic for which multicast groups. In the context of the algorithms presented in Sections 3 and 4, this solution can be implemented as a traffic engineering problem where the links are labeled with administrative constraints specifying which multicast groups are authorized over each link, and multicast trees are built using paths authorized for the given group.

Chapter 7

Inter-Domain Policy Routing Applications

Inter-domain routing protocols are designed to perform policy-based routing in an internet of autonomous systems. An autonomous system (AS) is defined as a set of routers under a single technical administration, using an interior gateway protocol and common metrics to route packets within the AS, and using exterior gateway protocols to route packets to other ASs. In practice, this definition is relaxed to allow multiple intra-domain protocols and several sets of metrics, the focus being on a single administration. The two primary inter-domain routing protocols currently defined are the Border Gateway Protocol (BGP) [64] (and it's descendant, the Inter-Domain Routing Protocol (IDRP) [41, 65]); these are primary in the sense that they are currently the protocols used to maintain the global Internet routing topology. A third protocol, based on a different architectural model, called Inter-Domain Policy

Routing protocol (IDPR) [80] has also been defined.

BGP is an inter-domain, path-vector routing protocol. It is designed to perform policy based routing in an internet composed of autonomous systems. The path-vector class of routing protocols was developed to address looping problems inherent to the distance-vector based protocols. Called path-vector, these protocols are similar to distance-vector protocols in that they are based on the Distributed Bellman-Ford algorithm; they differ in that the path traversed in the computation of a given route is carried with the computation as it propagates through a network for use detecting loops and for policy decisions. BGP exploits the path vector computed in these protocols to carry additional policy information for use in controlling the propagation and selection of routes based on domain-based policies.

In addition to their underlying algorithms, routing protocols differ in the basic function they serve in a network. In a simple network environment, owned and administered by a single entity the primary purpose of a routing protocol is to maintain connectivity information in the presence of topological changes. In such an environment the primary information processed by a routing protocol is reachability and link cost information. Protocols used in such environments are called intra-domain routing protocols. RIP [55] and OSPF [59] are popular intra-domain routing protocols. In contrast, inter-domain protocols are designed to perform policy based routing in an internet of autonomous systems.

The remainder of this chapter reviews and analyzes problems with the path-vector protocols currently used for inter-domain routing in the Internet, and proposes

an alternative architecture based on the IDPR protocol enhanced with the policy-routing algorithms presented above. Section 7.1 analyzes the security of BGP, presents a strategy for securing the protocol, reviews previous work in this area, and evaluates the results. Section 7.2 reviews convergence problems recently discovered in the BGP protocol. Lastly, Section 7.3 proposes a new inter-domain, policy-based routing architecture based on the IDPR protocol, enhanced with the routing algorithms presented above.

7.1 BGP Security Problems

There are four basic components in a BGP system: speakers, peers, links, and border routers [64]. A *BGP speaker* is a host in the network that executes the BGP protocol. *BGP peers* are two BGP speakers that form a connection and engage in a BGP dialog. A BGP peer is either an internal or external peer, depending on whether it is in the same or a different AS as the reference BGP speaker. The connections between BGP peers are called *links*, with internal and external links being defined similarly to internal and external peers. BGP links are formed using a reliable transport protocol such as TCP. This eliminates the need to implement transport services such as retransmissions, acknowledgments, and sequence numbers in the routing protocol.

A *border router* is a router with an interface to a physical network shared with border routers in other autonomous systems. Similar to BGP speakers, border routers are either internal or external. Note that BGP speakers need not be border routers

(or even routers of any kind). It is possible that a non-routing host could serve as the BGP speaker, gathering routing information from internal or other external routing protocols, and advertising that information to internal and neighboring external border routers. This feature is currently in use in the Route Servers of the Routing Arbiter project [31].

We make the following assumptions in designing security mechanisms for BGP:

- The BGP version 4 protocol as defined in RFC1771 [64].
- A BGP speaker can trust its internal peers.
- A BGP speaker can trust information it receives from external speakers only concerning links incident on the AS to whom the external speaker belongs.
- Intruders have capabilities as described in Section 7.1.1.
- Key distribution is based on domain names, which can be efficiently and securely determined given an IP address of a host, and the key distribution mechanism provides a controllable refresh rate. [1]

7.1.1 BGP Threats and Vulnerabilities

We now identify the threats to which BGP is susceptible, and the vulnerabilities these threats exploit. We consider separately threats to the flow of routing

[1] The DNS Security Extensions [29] might meet these requirements.

traffic and threats to the flow of data traffic that involve portions of the routing infrastructure. We describe attacks in terms of different classes of internet nodes including authorized BGP speakers and intruders. Authorized BGP speakers are those nodes intended by the authoritative network administrator to perform as a BGP speaker.

Intruders

We assume that an intruder can be located at any point in the network through which all traffic of interest flows, and that the intruder has the capability to fabricate, replay, monitor, modify, or delete any of this traffic. Interpreting this description for a BGP environment, we identify the following four general classes of intruders:

Subverted BGP speaker: A subverted BGP speaker occurs when an authorized BGP speaker is caused to violate the BGP protocols, or to inappropriately claim authority for network resources. This typically occurs due to bugs in the BGP software, mistakes in the speaker's configuration, or by causing a BGP speaker to load unauthorized software or configuration information, which can be achieved by many means, depending on the design and configuration of the BGP speaker.

Unauthorized BGP speaker: An unauthorized BGP speaker exists when a node that is not authorized as a BGP speaker manages to circumvent any access control mechanisms in place, and establish a BGP link with an authorized BGP speaker. How this is achieved depends on the design and configuration of existing access control mechanisms.

Masquerading BGP speaker: A masquerading BGP speaker occurs when a node successfully forges an authorized BGP speaker's identity. This can be accomplished using the IP spoofing [58] or source routing attacks.

Subverted link: There are a number of forms that a subverted link can take. One is to gain access to the physical medium (e.g. copper or fiber optic cable-plant, the "air-waves", or the electronics used to access them) in a manner that allows some control of the channel. In addition, a link may be subverted by compromising lower layer protocols in use on the link in a manner that allows control of the channel. An example of such an attack is the TCP session hijacking attack [43].

Threats to Routing Information

Under the correct circumstances an intruder can fabricate, modify, replay, or delete routing traffic. With these capabilities, an intruder can compromise the network in a number of ways. The modification or fabrication of routing updates allows an intruder to reconfigure the logical routing structure of an internet, potentially resulting in the denial of network service, the disclosure of network traffic, and the inaccurate accounting of network resource usage. The replay or deletion of routing updates blocks the evolution of subsets of the logical routing structure (in response to topological or policy changes), or resets it to an earlier configuration with results similar to above. The vulnerability exploited by these attacks is the lack of access control, authentication, and integrity of BGP message contents.

In addition, it is relatively easy for an intruder to gain access to routing

traffic. The information available from this traffic includes the appropriate next hop to reach a destination, and the path taken by traffic to different destinations. The next hop information is available from other sources, such as monitoring authorized traffic to the desired destination for the next hop it uses, and therefore cannot be protected solely by measures directed at the routing traffic. However, in some circumstances, the path used to reach different destinations may be considered confidential. The vulnerabilities exploited by these attacks are the lack of confidentiality of peer links and the level of trust placed in BGP speakers.

Threats to Data Traffic

It is relatively easy for an intruder to snoop or disclose data traffic. The vulnerability exploited to accomplish this is the lack of end-to-end or link encryption services for data traffic. We will not address the possible countermeasures to these attacks, because they should be implemented, in the link, network, or transport layer data transfer protocols such as Ethernet, IP or TCP, which is beyond the scope of our intended modifications to BGP.

It is also relatively easy for an intruder to fabricate, modify, replay, or delete data packets. The effectiveness of these attacks at deceiving or disrupting the source and destination processes depends on the end-to-end protocols in use at the transport layer and above, and is not a routing-protocol issue. However, the effectiveness of these attacks at deceiving the intermediate routing nodes is not an end-to-end protocol issue. Countermeasures to these vulnerabilities will depend on mechanisms in the network

or lower layers of the protocol hierarchy. The appropriateness and effectiveness of end-to-end vs. link layer security measures is a fundamental issue in the design of the Internet protocols [46, 66, 85]. While in general these issues do not involve routing protocol mechanisms, two exceptions include the ability to use multiple paths to a single destination, and the inclusion of authentication and access control mechanisms in the packet forwarding function [32]; we will not address these measures further in this section.

Goals for Securing BGP

In general, our goal in securing BGP is to provide authenticity, integrity, confidentiality, and access control of BGP message transmission. Referring to the previous sections, this goal translates specifically to preventing:

- The fabrication, modification, and replay of routing messages by all classes of intruder.
- The deletion of routing messages by subverted links and subverted speakers.
- The disclosure of routing messages by all classes of intruder.

In the following, we assume that access control is provided using the same naming and key distribution mechanism used to implement the authentication mechanism. The remaining access control design issues, such as the definition of the access control lists and their distribution mechanism, are orthogonal to the countermeasures presented here, and are not discussed further.

7.1.2 BGP Security Countermeasures

Two classes of communication occur in BGP and routing protocols in general: between neighboring speakers, and between a given speaker and an arbitrary set of remote speakers determined dynamically by routing decisions. That communication between neighboring speakers is composed of routing updates for destinations that the sender has determined are appropriate to send to the receiver. The communication between a speaker and remote speakers is composed of the fields of routing updates which describe a given destination. Accordingly, we present the following two classes of countermeasures:

BGP Message Protection Countermeasures:

- Encrypt all BGP messages between peers using session keys exchanged at BGP link establishment time. This encryption provides integrity and authenticity of all path attributes whose values are valid for at most one AS hop, and confidentiality of all routing exchanges.
- Add a message sequence number to protect against replayed or deleted messages.

BGP Update Field Protection Countermeasures:

- Add an `UPDATE` sequence number or timestamp to protect against replayed `UPDATE` messages.
- Add a `PREDECESSOR` path attribute indicating the AS prior to the destination AS for the current route. This allows the verification of the path

information using the AS_PATH path attribute.

- Digitally sign all unchanging UPDATE fields whose values are fixed on creation by the BGP speaker originating or most recently aggregating the route. This provides for the integrity and authenticity of not only these fields, but also of the full AS_PATH.

The rest of this section presents a more detailed description of these countermeasures, and an analysis of the effectiveness of these countermeasures against the threats and vulnerabilities identified previously.

BGP Message Protection Countermeasures

The purpose of these countermeasures is to provide authentication, confidentiality, and integrity of the routing messages between BGP peers, which compose the first class of communication described above. Specifically, the message encryption and message sequence number provide corruption detection, sequencing, acknowledgment, and retransmission mechanisms. While these mechanisms are redundant to those provided by TCP, they are required due to the insecurity of the TCP mechanisms [27, 45]. As discussed by Tardo [82], these countermeasures are most appropriately provided at the network or transport layers. These BGP countermeasures would no longer be required if a secure network [47] or secure transport protocol [44, 63] were used.

Message Encryption Upon establishment of each BGP link, a session key is exchanged by the peers to encrypt each BGP message transmitted over that link. This

encryption provides confidentiality of messages, as well as authenticity and integrity of KEEPALIVE messages, NOTIFICATION messages, and some of the path attributes carried in UPDATE messages.

A number of path attributes carried in UPDATE messages are modified in each AS they transit. These include the NEXT_HOP, MULTI_EXIT_DISC, and LOCAL_PREF attributes. The use of peer-to-peer encryption for authenticity and integrity of these path attributes is based on two observations: (a) the recipient of these path attributes receives them from either the most recent modifier or via a single relay that is an internal peer, and (b) our assumption that internal peers are trusted. Given these, peer-to-peer encryption provides a high degree of security in an efficient manner. On detection of corrupted information, the link is terminated using a NOTIFICATION message.

Message Sequence Number A sequence number is added to each message; it is initialized to zero on establishment of a BGP link, and is incremented with each message. On detection of a skipped or repeated sequence number, the BGP link is terminated with a NOTIFICATION message. The size of the sequence number is made large enough to minimize the chance of it cycling back to zero. However, in the event that it does, the link is terminated and a new link is established, resetting the sequence number to zero and establishing a new session key.

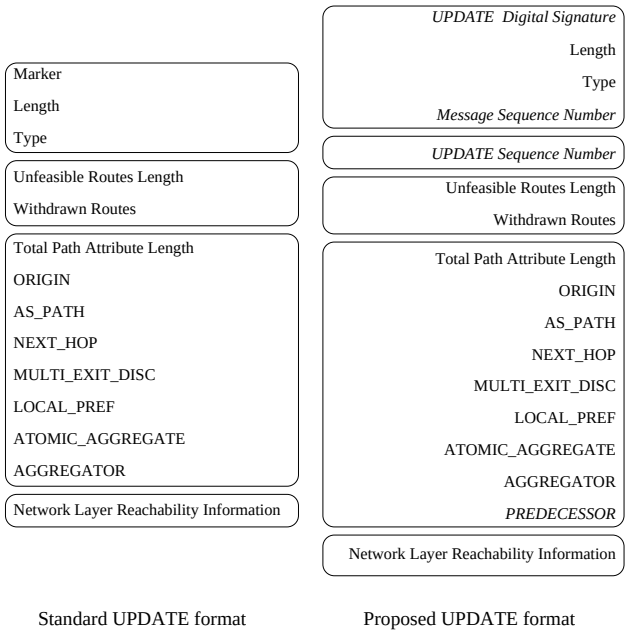

Figure 7.1: Proposed UPDATE Message Changes

UPDATE Field Protection Countermeasures

The countermeasures presented in this section protect the communication between a given speaker and a set of remote speakers. These countermeasures provide only for authenticity and integrity of this communication, because confidentiality of this communication is unnecessary as the potential recipients include all authorized BGP speakers in an internet. As discussed by Tardo [82], these countermeasures provide authentication and integrity of fields within a message, and are most appropriately implemented in the presentation layer.

Figure 7.1 illustrates the proposed modifications using the `UPDATE` message, which includes all proposed new fields, as a model.

UPDATE Sequence Number or Timestamp Sequence information is added to each `UPDATE` message to protect against the replay of old routing information. This sequence information is generated for each route output from the BGP decision process, and can be in the form of a sequence number or a timestamp. While a number of `UPDATE` messages may be generated for each route (one message per peer of the originating speaker), only one sequence number or timestamp is used for all of them.

This sequence information is necessary because a remote speaker may receive the same route in multiple updates, each describing the same destination but representing different paths, and all of these `UPDATES` must be considered valid. This implies that `UPDATES` for a given destination must be considered valid if their sequence information is greater than or equal to the current sequence information. Note that sequence information must be maintained and validated on a per speaker basis. An invalid `UPDATE` message is dropped silently.

In a BGP environment, sequence numbers would have a potentially long life. Given the recommended value of BGP's `MinASOriginationInterval` timer (15 seconds) the sequence number can be relatively small and still be assured of not cycling. Setting this timer to as low as 8 seconds, and assuming a new `UPDATE` is originated at the end of every interval, a four octet sequence number would last for over 1000 years. The main difficulty introduced by a sequence number consists of maintaining it in the context of arbitrary software and hardware failures. Techniques such as those proposed by Perlman [62] could be used; however, if cycling of the sequence number

must be supported, the following process can be used:

> Each BGP speaker maintains an `UPDATE` message sequence number database on a per BGP speaker $< domainname, publickey >$ pair basis. When the cycling of a sequence number approaches, a new public-key pair is generated. The key distribution mechanism and BGP speaker are updated with the new key pair, and the speaker's `UPDATE` sequence number is reset to zero. On detection of a change in the public key for an originating speaker, the receiving speaker will add an entry to its `UPDATE` sequence number database for the new originating speaker $< domainname, publickey >$ pair with a sequence number of zero. It will continue to use the old sequence number entry until a sequence number failure occurs where the digital signature validation succeeds using the new entry. At this time the old entry is purged, and the conversion to a new sequence number is complete. Further work is needed on a mechanism to load the database of a newly-booted BGP speaker.

A timestamp could be used instead of a sequence number. The main benefit of a timestamp would be the ease of administration provided by the well-defined external reference for use in resetting lost state. The life of even a small timestamp, while not as long as for sequence numbers, is still significant; assuming a granularity as small as one second, a four octet timestamp still has a life longer than 130 years.

New PREDECESSOR Path Attribute To ensure the authenticity of the `AS_PATH` attribute, we augment `UPDATE` messages with a `PREDECESSOR` attribute identifying the AS prior to the destination AS for the current route. We call this AS the predecessor to the destination AS. By including the predecessor information, and a digital signature of this information calculated by the originating speaker (described in Section 7.1.2), the authenticity and integrity of the complete path reported by a speaker to any destination can be established by the speaker's neighbors. Specifically, this can be done by means of a path traversal of the verified predecessor information reported by the

route.

The PREDECESSOR path attribute includes: the originating AS, the predecessor AS, an IP address of the originating speaker, and a type field. The originating AS must be the same as the AS in the AGGREGATOR and the first AS in the first AS_SEQUENCE segment of the AS_PATH path attribute, if these attributes exist. The predecessor AS must be the same as the second AS in the first AS_SEQUENCE of the AS_PATH attribute, if it exists. The IP address of the originating speaker must be the same as the IP address in the AGGREGATOR attribute, if it exists.

The TYPE field can take on the value of either ADD or DELETE. The ADD version of the PREDECESSOR attribute is generated by the speaker that originates the UPDATE message, which may either be the creator of an unaggregated UPDATE, or the last speaker to perform an aggregation of the routing information in the current UPDATE. The purpose of the ADD type of the PREDECESSOR path attribute is to identify the originating BGP speaker whose key is used to digitally sign the UPDATE, and to identify the destination and predecessor information in the absence of AGGREGATOR and AS_PATH attributes (see below regarding transit-only UPDATES). The DELETE version of the PREDECESSOR path attribute serves the purpose of identifying a previously reported predecessor relationship that is no longer valid. Possible reasons for this change include the failure of an inter-AS link, or the termination of a transit traffic agreement. This segment type may be generated by either end of the deleted link; the originating AS field of the PREDECESSOR attribute specifies the generating BGP peer.

The predecessor information is used by each node to maintain a *predeces-*

sor table. The predecessor table is a column vector containing the predecessor to the destination and to each intermediate node on the chosen path to each known destination. The information maintained in the predecessor table is used to verify `AS_PATH` attributes. Before a speaker selects a route, that route's `AS_PATH` attribute should be verified by a walk through the predecessor table. This verification is done by traversing backwards through all `AS_SEQUENCE` segments in the `AS_PATH`, starting with the first AS in the first `AS_SEQUENCE` path segment, confirming that a validated predecessor table entry exists for each predecessor AS in the `AS_PATH`. The timing of this verification is not specified, and is influenced by the expected frequency of invalid `AS_PATH` attributes, expected load, and the performance requirements of the speaker. Options for when to perform this verification include on receipt of the `AS_PATH`, or on selection of the route for use. This check could also be performed on a statistical basis if loads are excessive.

Policy-based Handling of PREDECESSOR Attributes Smith, Murthy and Garcia-Luna-Aceves [78] have shown how predecessor information alone is adequate to secure intra-domain distance-vector routing protocols. This is possible in these protocols because the paths used to reach destinations downstream from a given node are extensions of the path used to reach the node itself. As a result, at most one update for any given node is passed along by upstream routers, and a chain of $< destination, predecessor >$ pairs uniquely identifies a path to any destination.

However, in inter-domain distance-vector routing protocols in which routing

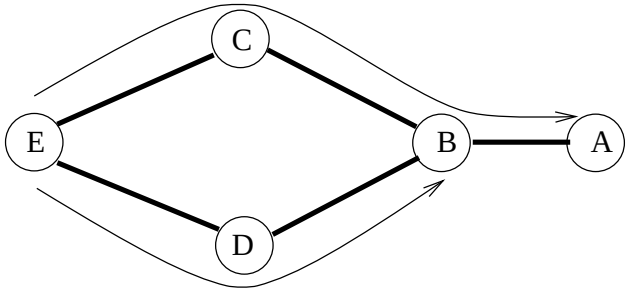

Figure 7.2: Need for Multiple Predecessors

decisions are made based on arbitrary policies, the assumption that paths through a node are extensions of the paths used to reach that node no longer holds. In policy-based routing, the path used to reach a node as a destination does not necessarily have any relation to the path used to reach a node as a relay to a different destination. This is illustrated in Figure 7.2. In the figure, due to policy-based decisions, a speaker for AS E has chosen path $< C, B, A >$ to reach destination A, and path $< D, B >$ to reach destination B. To validate both paths, the speaker for AS E needs two predecessor table entries for B; one through C and one through D. As a result, more than one update for a given node, each with a different predecessor, can be passed along by upstream speakers. The only restriction on handling updates being that at most one update from a given node is used to reach any one destination. This relaxed restriction results in two additional requirements of the protocol.

First, each speaker upstream from a given predecessor link must maintain a list of destinations that will be accepted over that link. To allow these destination lists to be kept current, speakers must include a list of destinations for which they will handle traffic in updates they generate. Additionally, as speakers learn of new

destinations, they must generate new updates (called transit updates) to add these destinations to the list of valid destinations for their predecessor links. A danger of transit updates is that they become hop-by-hop security for each `AS_PATH`. Fortunately, there are a number of factors that mitigate this problem. First, it is not necessary to re-authenticate the same predecessor link/destination pair as paths from the transit AS to the destination change over time. Second, it is not necessary to delete a valid destination for a predecessor link simply because the destination becomes unreachable. Lastly, the `MinASOriginationInterval` variable mentioned earlier causes updates to tend to consolidate, resulting in fewer updates describing more significant changes rather than more updates describing smaller changes. Further work is needed on these issues. Specifically in the areas of how to specify the destinations (AS, IP address prefixes or both), and how to invalidate predecessor link/destination pairs.

Second, a new mechanism must be defined for determining the correct predecessor for a given destination in the path-traversal described previously. With the relaxed restriction on the propagation of updates described above, it is now possible for an upstream speaker to have information describing multiple predecessor links to the same AS that are valid for the same destination. To allow upstream speakers to uniquely determine the correct predecessor link for the path from itself to the destination, each speaker includes information in each update it generates identifying the successor to the downstream AS of the predecessor link it uses for the destination valid on that link. Because each speaker selects only one successor for a destination, the existence of the desired destination in the list of destinations specified for both the

predecessor and successor links uniquely identifies the link as part of the current path, and secures the link from both the upstream and downstream ends.

To summarize, the relaxed restriction on the propagation of updates in policy-based routing requires the predecessor information used to secure non-policy-based distance-vector routing protocols to be expanded to include information identifying the destinations traffic is accepted for over that link, and information specifying the successor AS to be used in reaching each of these destinations. Specifically, each update now includes, in addition to the information listed in Section 7.1.2, a list of successor ASs and the destinations for which each AS will handle traffic.

This information is used by upstream speakers to maintain a more complicated distance table composed of, for each neighbor, a two dimensional matrix indexed by destination and originating AS pairs. Each element of this matrix contains a list of quadruplets of: predecessor AS, successor AS, destinations, and distance to the originating AS. An `AS_PATH` is verified by walking backwards through the path verifying that the appropriate overlapping predecessor/originator/successor entries exist, and that the intermediate distances are consistent. Note that in this context the originating AS can act as a relay or a destination (or both). To advertise destinations in its containing AS an originating speaker should include a null successor AS with its AS as a destination in the `PREDECESSOR` field.

UPDATE Digital Signature To ensure the integrity and authenticity of the unchanging `UPDATE` message information, it is digitally-signed by the originating BGP

speaker specified in the PREDECESSOR attribute. Without protection, trust of this information requires trust of BGP peers regarding information concerning links not incident on their AS. This is something we explicitly do not do. By including the PREDECESSOR attribute information in this signature we protect, in addition to the information in the current UPDATE, the full path information contained in the predecessor table described above.

The UPDATE message digital signature is stored in the Marker field of the header, and is calculated over the following fields: UPDATE sequence number, Unfeasible Route Length, Withdrawn Routes, ORIGIN, ATOMIC_AGGREGATE, AGGREGATOR, PREDECESSOR, and the NLRI. This definition of the digital signature assumes that these fields are only meaningful as a unit; that a change in one requires the re-computation of them all. If the protocol evolves to where this is not the case, and subsets of these attributes may be updated independently by different BGP speakers, additional sequence numbers and associated digital signatures will be introduced.

Countermeasure Effectiveness

We now analyze the impact of each countermeasure on the threats identified in Section 7.1.1. The message protection countermeasures provide protection against all nodes lacking the necessary cryptographic keys, specifically unauthorized speakers, masquerading speakers, and subverted links. The encryption of BGP messages protects them from fabrication, modification, and disclosure by these classes of intruders. The addition of a sequence number to BGP messages protects them from replay or deletion

by these intruders.

Similarly, the UPDATE field protection countermeasures provide protection against compromise by those nodes that do have the cryptographic keys, specifically subverted speakers. The digital signature of the Withdrawn Routes, ORIGIN, AS_PATH, ATOMIC_AGGREGATE, AGGREGATOR, NLRI, and new PREDECESSOR and UPDATE Sequence Number fields protects these fields from fabrication or modification by subverted speakers. The addition of the UPDATE Sequence Number protects against the replay of these fields by a subverted speaker. The addition of the PREDECESSOR path attribute provides a means of validating a link in the internet, which can then be used to validate each link in the AS_PATH attribute.

Referring back to Section 7.1.1 we see that we have achieved all but a few of our goals. Specifically, a subverted speaker is still able to fabricate destination information, delete routing updates, and disclose routing information. In retrospect, we can see that these goals conflict with our basic assumptions of trust in BGP speakers regarding policy and connectivity information concerning resources for which they are authoritative, and trust to handle routing information confidentially. We believe these vulnerabilities are unavoidable, because they are inherent to the requirements of the protocol.

Performance Analysis

The cost of these countermeasures is in the space for the new sequence numbers and digital signatures, and the time for computing encryption and digital signa-

tures, and verifying these protections. From the perspective of the actions occurring in a BGP system, the costs are the following:

Message generation and reception: **Space:** A new field is added for the peer-to-peer sequence number. **Time:** The cost of a symmetric key encryption and decryption of each message.

Initiation and reception of UPDATE messages: **Space:** The `Marker` field is used for the `UPDATE` message digital-signature. Each `UPDATE` message includes a new `UPDATE` sequence number and `PREDECESSOR` attribute. **Time:** The time to perform the computation and verification of the `UPDATE` message signature.

Route Selection: **Time:** The time to verify signatures for each link. This cost will only be incurred twice for each used link: once for the `ADD` and once for the `DELETE`.

While these costs are not constant per destination (due to the possible need for intermediate nodes to send "transit `PREDECESSOR`" path attributes), they do offer the potential for significantly lower costs than the linear growth in cost with path length of previous solutions. The factors contributing to this improvement were outlined in Section 7.1.2.

7.1.3 Related Work

Kumar [50] analyzes the security requirements of network routing protocols, and discusses the general measures needed to secure the distance-vector and link-state routing protocol classes. He identifies two sources of attacks: subverted routers, and subverted links. Since attacks by subverted routers are seen as difficult to detect and of limited value to the intruder, Kumar focuses his attention on securing protocols from attacks by subverted links. For distance-vector protocols, this translates into the modification or replay of routing updates. The specific countermeasures proposed by Kumar are neighbor-to-neighbor digital signature of routing updates, the addition of sequence numbers and timestamps to the updates, and the addition of acknowledgments and retransmissions of routing updates. Kumar and Crowcroft [51] perform a similar analysis of inter-domain protocols, and come to similar conclusions for providing security of distance-vector related routing protocols (they specifically address the path-vector routing protocol IDRP). The one addition they make is to encrypt neighbor-to-neighbor updates.

These results are similar to ours with the exception that we explicitly assume the existence of subverted routers, and provide countermeasures to protect against them. We feel this is necessary, because BGP speakers are potentially vulnerable to attacks from a number of sources, with potentially catastrophic results from success.

Murphy [60] outlines a solution for securing distance-vector protocols that involves including the information used to select a route, signed by the neighbor from which it received it, in the routing update it then signs and transmits to its neighbors.

Murphy points out that this requires the validation of a number of nested signatures equal to the number of routers in the path. This results in both update size and validation computation time problems as the size of the network grows. These problems result, fundamentally, from the redundant signing of link information for paths that are supersets of paths used to reach destinations traversed in the longer path. In contrast, we avoid these problems by signing only the component link information, in the form of predecessors, and performing a path traversal to validate full paths. This results in the use of constant space, and significantly reduced computation time.

Smith and Garcia-Luna-Aceves [77, 78] have presented security mechanisms for BGP and distance-vector protocols in general. The proposed solutions are similar to those presented here, without as detailed an analysis of the implications of policy-based decisions on the countermeasures.

Recent work by Kent et al [48] documented the design and implementation of a Secure Border Gateway Protocol (S-BGP). This design follows the model of the secure distance-vector solution suggested by Murphy [60] involving the nested signature of routing updates as they propagate through an internet. While a number of optimizations have been attempted to manage the cost of such a solution, the costs (computational, bandwidth, and storage) incurred in this solution are significant. From recent personal discussions it appears that there has been little interest on the part of network operators to adopt this solution, and work continues on improving the efficiency of the solution.

7.2 BGP Convergence Problems

A specific requirement of inter-domain policy-based routing is the support of AS-specific, *private* policies by the routing computation. An important implication of this requirement is that there is no guarantee that consistent route selection criteria are used by the different nodes in the internet. Being fundamental to the correctness, in particular the loop-free nature, of modern routing algorithms the lack of this consistency poses a major problem in the design of inter-domain protocols. For BGP this has proven to be a problem. Recent research [37, 84] has shown that the path selection algorithm used by BGP (and IDRP) is susceptible to persistent route oscillations (i.e. the routing computation will not converge) under certain conditions. The source of this instability is the simultaneous use of hop-by-hop forwarding and unconstrained path selection policies. While work has progressed on algorithmic and administrative mechanisms for constraining path selection policies to ensure convergence of BGP route computations, the solutions have proven to be painful either in the limitations put on usable policies, or in the administrative and logistical overhead required to ensure convergence. Based on these results the fundamental policy-routing model assumed by BGP must be questioned.

7.3 Solution – IDPR

In contrast to BGP, the Inter-Domain Policy-Based Routing protocol (IDPR) is an inter-domain, link-state routing protocol that uses source-specified forwarding.

This policy-routing model (specifically, link-state with source-specified forwarding) allows IDPR to avoid the convergence problems experienced by BGP. In an IDPR environment, two classes of routing policies are defined: transit and source. *Transit policies* define how resources in a domain can be used by transit traffic. By default, these policies are assumed to be public, and are flooded as part of the link-state distribution to all ASs in an internet. *Source policies* define how traffic originating from a domain is to be handled as it travels through an internet. By default, source policies are assumed to be private, and are not transmitted outside an AS. Routes are computed by IDPR routers in the source domain in compliance with the (in general) globally known transit policies, and the locally known source policies. Forwarding state is established using source-specified path setup. These computations are typically done on-demand, although provision is made for the pre-computation and path setup of routes for requests that can reliably be anticipated. Provision is also made for the restricted distribution of private transit policies, when needed.

The routing algorithms developed in Sections 3 and 4 can be applied to IDPR in two ways. First, these algorithms can be used to provide more efficient on-demand routing computations in the currently proposed IDPR architecture. Additionally, IDPR can be enhanced to provide more efficient support for public source policies. Such support would provide significant benefits in the case where widely desired, non-sensitive source policies existed. Fur such policies, IDPR could perform table-driven, hop-by-hop routing computations using the table-driven routing algorithms presented above. As described in Section 2.1, the benefits of such computations include more

efficient, responsive, and robust forwarding of such traffic. Furthermore, being fundamentally a link-state protocol, IDPR can be secured using straightforward techniques such as those proposed for securing other link-state protocols [61]. In general, the combination of the IDPR inter-domain, policy-based routing protocol, and the efficient policy-based routing algorithms presented here would provide an efficient, easily securable, and provably correct alternative to BGP.

Chapter 8

Conclusions

The goal of this dissertation was to explore the problem of efficiently supporting policy-based routing in the table-driven, hop-by-hop routing model used in the Internet architecture. We formulated an integrated policy-based routing architecture, where policy-based routing is defined as routing in the context of multiple metrics. Policy-based routing can then be seen to support traffic engineering by the computation of routes in the context of metrics specifying administrative constraints on the type of traffic allowed over portions of an internet. Analogously, policy-based routing supports QoS requirements by the computation of routes in the context of metrics describing performance-related characteristics of links in an internet. We presented a traffic algebra that formalized the notion of a traffic class, and identified an efficient set-based implementation of the traffic algebra for traffic expressions composed of restricted numbers of variables. We presented an extension of Sobrinho's path algebra [79] to formalize the notion of the best set of weights for a given destination.

We then presented a family of routing algorithms that compute exact solutions to policy-based routing problems using these algebras. The enhanced version of TD-TE-Dijkstra is the most efficient algorithm available for computing routes satisfying traffic engineering constraints on the traffic allowed to traverse subsets of an internet. The enhanced version of TD-QoS-Dijkstra is the most efficient algorithm available for computing routes satisfying QoS constraints on the paths allowed to carry traffic for a given flow. Policy-Based-Dijkstra is the first algorithm for computing routes that simultaneously satisfy traffic engineering and QoS constraints.

A new forwarding architecture based on distributed label-swapping was presented that efficiently supports multiple-paths per destination which is required for policy-based routing. In this architecture the routing protocol is enhanced to exchange the additional link metrics, and forwarding table information. The neighbor forwarding table information is used to compute the label-swap entries as well as the traffic classification rules used by each router. A significant innovation of the policy-based routing architecture presented here is the combination of the table-driven, hop-by hop routing model with label-swap forwarding mechanisms. This innovation is based on the insight that host addresses and labels are largely equivalent alternatives for representing forwarding state, and that the virtual-circuit nature of prior label-swapped architectures derives from their use of a source-driven forwarding model. The advantage of this distributed label-swap forwarding model over traditional address-based forwarding is that label-swap forwarding provides a clean separation of the control and forwarding planes, while address-based forwarding unnecessarily binds them to-

gether. In addition, enhanced versions of the routing algorithms were presented which compute routes that are compatible with this forwarding architecture.

We articulated a new model for the complexity of policy-based routing where the complexity of the computation is seen to be driven by the number of incomparable paths existing in a graph. Based on this new model the previous algorithms for computing approximate solutions to the policy-based routing problem can be seen as attempting to limit the number of incomparable routes possible in a graph through the manipulation of link weights. The limitation of this approach is that it does not work in the context of administrative policies. Taking the alternative approach of limiting the total number of paths in the graph, we developed the first algorithms for computing approximate solutions in the context of administrative constraints, and both administrative and performance constraints. Experimental results showed that the performance of this algorithm compared favorably with that of the previous solutions.

Lastly we explored the application of this new technology to a number of specific problems. We showed that the application of these algorithms and techniques to intra-domain policy-based unicast and multicast routing could provide significant improvements in manageability and security of IP-based networks. Further more, the application of these technologies to inter-domain policy-based routing offers a solution to the critical problems recently identified with the current path-vector solutions.

8.1 Future Work

The most significant future work to be done is relating to the traffic algebra and processing of traffic expressions. First, investigation needs to be made into efficient solutions to the satisfiability problem for traffic expressions. Similar to studies on the characteristics of expressions encountered in hardware design and software verification, and strategies for efficiently determining their satisfiability, investigation needs to be done into the typical structure of expressions encountered in a policy-based routing context, and strategies identified to efficiently process such expressions. Furthermore, investigation needs to be done into higher-order reasoning about these expressions for use in network analysis and design to enable such systems to be effectively deployed. Further research is needed into more sophisticated approximation solutions based on the model defined here. Lastly, work is needed into the application of this exciting new technology. Examples include: intra-domain policy-based unicast routing where the routing infrastructure, in effect, can bee seen as a distributed firewall; intra-domain policy-based multicast routing where significant manageability and security problems with the existing multicast architecture can be addressed; and inter-domain policy-based routing where a new approach can now be considered, based on the availability of these new algorithms, that is a more direct and provably correct solution to the problem.

Bibliography

[1] The ARPA Network. Session presented at the Spring Joint Computer Conference, May 1972.

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

[3] Kevin C. Almeroth. Managing IP Multicast Traffic: A First Look at the Issues, Tools, and Challenges. IP Multicast Initiative Summit, San Jose, CA, February 1999.

[4] George Apostolopoulos and Satish K. Tripathi. On the Effectiveness of Path Pre-Computation in Reducing the Processing Cost of On-Demand QoS Path Computation. In *Proceedings of the IEEE Symposium on Computers and Communications*. IEEE, June 1998.

[5] Daniel O. Awduche, Joe Malcolm, Johnson Agogbua, Mike O'Dell, and Jim McManus. Requirements for Traffic Engineering Over MPLS. RFC2702, September 1999.

[6] Anthony Ballardie. Core Based Tree (CBT) Multicast Routing Architecture. RFC 2201, September 1997.

[7] Tony Ballardie and Jon Crowcroft. Multicast-Specific Security Threats and Counter-Measures. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 2–16. The Internet Society, February 1995.

[8] Steven M. Bellovin. Security Mechanisms for the Internet. Internet Draft: draft-iab-secmech-00.txt, November 1998.

[9] Dimitri Bertsekas and Robert Gallager. *Data Networks*. Prentice Hall, 2nd edition, 1992.

[10] Steven Blake, David L. Black, Mark A. Carlson, Elwyn Davies, Zheng Wang, and

Walter Weiss. An Architecture for Differentiated Services. RFC2475, December 1998.

[11] Bob Braden, David Clark, and Scott Shenker. Integrated Services in the Internet Architecture: an Overview. RFC1633, July 1994.

[12] Bob Braden and Lixia Zhang. Resource ReSerVation Protocol (RSVP). RFC2209, September 1997.

[13] Ran Canetti, Pau-Chen Cheng, Dimitris Pendarakis, J.R. Rao, Pankaj Rohatgi, and Debanjan Saha. An Architecture for Secure Internet Multicast. Internet Draft: draft-irtf-smug-sec-mcast-arch-00.txt, February 1999.

[14] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *Proceedings of INFOCOM'99*. IEEE Computer and Communication Societies, 1999.

[15] Ran Canetti and Benny Pinkas. A taxonomy of multicast security issues. Internet Draft: draft-canetti-secure-multicast taxonomy-01.txt, November 1998.

[16] Germano Caronni, Marcel Waldvogel, Dan Sun, and Bernhard Plattner. Efficient Security for Large and Dynamic Multicast Groups. In *7th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '98)*, pages 376–383, June 1998.

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

[18] Vinton G. Cerf. The Catenet Model for Internetworking. IEN 48, July 1978.

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

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

[21] Isabella Chang, Robert Engel, Dilip Kandlur, Dimitrios Pendarakis, and Debanjan Saha. Key Management for Secure Internet Multicast using Boolean Function Minimization Techniques. In *Proceedings of INFOCOM'99*. IEEE Computer and Communication Societies, 1999.

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

[23] DARPA – Information Assurance, Security Focus Group. Security Architecture for the AITS Reference Architecture. Technical report, DARPA, http://web-ext2.darpa.mil/iso/ia/, December 1997. Revision 1.0.

[24] Bruce Davie and Yakov Rekhter. *MPLS: Technology and Applications.* Morgan Kaufmann, 2000.

[25] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ahmed Helmy, David Meyer, and Liming Wei. Protocol Independent Multicast Version 2 Dense Mode Specification. Internet Draft: draft-ietf-pim-v2-dm-01.txt, November 1998.

[26] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer System*, 8(2):85–110, May 1990.

[27] Whitfield Diffie. Security for the DoD Transmission Control Protocol. *CRYPTO '85*, pages 108–127, 1985.

[28] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Network*, January 2000.

[29] Donald E. Eastlake. Domain Name System Security Extensions. RFC2535, March 1999.

[30] Deborah Estrin, Dino Farinacci, Ahmed Helmy, David Thaler, Stephen Deering, Mark Handley, Van Jacobson, Ching gung Liu, Puneet Sharma, and Liming Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC 2362, June 1998.

[31] Deborah Estrin, Jon Postel, and Yakov Rekhter. Routing Arbiter Architecture. ftp://ftp.isi.edu/pub/hpcc-papers/ra/ra-arch.ps, June 1994.

[32] Deborah Estrin, Martha Steenstrup, and Gene Tsudik. A Protocol for Route Establishment and Packet Forwarding Across Multidomain Internets. *IEEE Trans. on Networking*, 1(1):56–70, February 1993.

[33] William Fenner. Internet Group Management Protocol, Version 2. RFC 2236, November 1997.

[34] J. J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE Trans. on Networking*, 1(1), 1993.

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

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

[37] Timothy G. Griffin and Gordon Wilfong. A Safe Path Vector Protocol. In *Proceedings INFOCOM 2000.* IEEE, March 2000.

[38] Thomas Hardjono and Brad Cain. Secure and Scalable Inter-Domain Group Key Management for N-to-N Multicast. In *1998 International Conference on Parallel and Distributed Systems (ICPADS '98)*, pages 478–485. IEEE, December 1998.

[39] Thomas Hardjono and Gene Tsudik. IP Multicast Security: Issues and Directions. *Annales de Telecom*, 2000.

[40] Dan Harkins and Dave Carrel. The Internet Key Exchange (IKE). RFC2409, November 1998.

[41] International Standards Organization. *ISO/IEC 10747: Information technology - Telecommunications and information exchange between system - Protocol for exchange of inter-domain routing information among intermediate systems to support forwarding of ISO 8473 PDUs*, August 1994.

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

[43] Laurent Joncheray. A Simple Active Attack Against TCP. *Proc. 5th USENIX UNIX Security Symposium*, pages 7–19, June 1995.

[44] Laurent Joncheray. Public Key Encryption Support for TCP. Internet Draft: draft-joncheray-encryption-00.txt, May 1995.

[45] Stephen Kent. Some Thoughts on TCP and Communication Security. MIT Laboratory for Computer Science, Local Network Note No. 6, April 1977.

[46] Stephen Kent. Comments on "Security Problems in the TCP/IP Protocol Suite". *ACM Computer Commun. Review*, 19(3):10–19, July 1989.

[47] Stephen Kent and Randall Atkinson. Security architecture for the internet protocol. RFC 2401, November 1998.

[48] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure Border Gateway Protocol (S-BGP) – Real World Performance and Deployment Issues. In *Proceedings Network and Distributed Systems Security Symposium.* ISOC, February 2000.

[49] Donald E. Knuth. Nested Satisfiability. *Acta Informatica*, pages 1–6, 1990.

[50] Brijesh Kumar. Integration of Security in Network Routing Protocols. *ACM SIGSAC Review*, 11(2):18–25, Spring 1993.

[51] Brijesh Kumar and Jon Crowcroft. Integrating Security in Inter-Domain Routing Protocols. *ACM Computer Commun. Review*, pages 36–51, 1993.

[52] Whay C. Lee, Michael G. Hluchyi, and Pierre A. Humblet. Routing Subject to Quality of Service Constraints in Integrated Communication Networks. *IEEE Network*, 9(4):46–55, August 1995.

[53] Brian Neil Levine and J.J. Garcia-Luna-Aceves. A Comparison of Reliable Multicast Protocols. *Multimedia Systems*, 6(5):334–348, August 1998.

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

[55] G. Malkin. RIP Version 2: Carrying Additional Information. RFC 1723, November 1994.

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

[57] Suvo Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 277–288. ACM, September 1997.

[58] Robert T. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software. Technical Report 117, AT&T Bell Laboratories, Murray Hill, New Jersey 07974, February 1985. ftp://netlib.att.com/netlib/att/cs/cstr/117.ps.Z.

[59] J. Moy. OSPF Version 2. RFC 1583, March 1994.

[60] Sandra L. Murphy. Presentation in Panel on "Security Architecture for the Internet Infrastructure". Symposium on Network and Distributed System Security, April 1995.

[61] Sandra L. Murphy. Digital Signature Protection of the OSPF Routing Protocol. *Proc. Symposium on Network and Distributed System Security*, 1996. http://bilbo.usy.edu/sndss/sndss96.html.

[62] R. Perlman. *Network Layer Protocols with Byzantine Robustness*. Report MIT/LCS/TR 429, Massachusetts Institute of Technology, October 1988.

[63] P.V. Rangan. Trust Requirements and Performance of a Fast Subtransport-Level Protocol for Secure Communication. *IEEE Transactions on Software Engineering*, 19(2):181–186, 1993.

[64] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 1771, March 1995.

[65] Yakov Rekhter. Inter-domain Routing Protocol (IDRP). *Internetworking: Research and Experience*, 4:61–80, 1993.

[66] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Trans. on Computer Systems*, 2(4):277–288, November 1984.

[67] Jerome H. Saltzer. On the Naming and Binding of Network Destinations. RFC 1498, August 1993.

[68] Kamil Sarac and Kevin C. Almeroth. Supporting Multicast Deployment Efforts: A Survey of Tools for Multicast Monitoring. *Journal of High-Speed Networking – Special Issue on Management of Multimedia Networking*, March 2001.

[69] Thomas J. Schaefer. The Complexity of Satisfiability Problems. In *10th ACM Symposium on the Theory of Computing*, pages 216–226, 1978.

[70] Bruce Schneier. Why Cryptography Is Harder Than It Looks. http://counterpane.se-com.com.

[71] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc, 2nd edition, 1996.

[72] Clay Shields and J.J. Garcia-Luna-Aceves. The ordered core based tree protocol. In *Proceedings IEEE INFOCOM'97*, April 1997.

[73] Clay Shields and J.J. Garcia-Luna-Aceves. KHIP – A Scalable Protocol for Secure Multicast Routing. In *Proceedings of SIGCOMM'99*. ACM, September 1999.

[74] Robert W. Shirey. Security Architecture for Internet Protocols. A Guide for Protocol Designs and Standards. Internet Draft: draft-irtf-psrg-secarch-sect1-00.txt, November 1994.

[75] John F. Shoch. Inter-Network Naming, Addressing, and Routing. In *Proceedings of COMPCON'78*. IEEE, September 1978.

[76] Stavroula Siachalou and Leonidas Georgiadis. Efficient QoS Routing. In *Proceedigns of INFOCOM'03*. IEEE, April 2003.

[77] Bradley R. Smith and J. J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. *Proc. IEEE Global Internet '96*, November 1996.

[78] Bradley R. Smith, Shree Murthy, and J. J. Garcia-Luna-Aceves. Securing Distance-Vector Routing Protocols. *Proc. ISOC Symposium on Network and Distributed System Security*, pages 85–92, February 1997.

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

[80] Martha Steenstrup. Inter-Domain Policy Routing Protocol Specification: Version 1. RFC 1479, July 1993.

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

[82] Joseph J. Tardo. Standardizing Cryptographic Services at OSI Higher Layers. *IEEE Communications Magazine*, 23(7):25–29, July 1985.

[83] Dave Thaler and Bernard Aboba. Multicast Debugging Handbook. Internet Draft: draft-mboned-mdh-05.txt, November 2000.

[84] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent Route Oscillations in Inter-Domain Routing. Technical Report CS-TR-96-631, University of Southern California, March 1996.

[85] Victor L. Voydock and Stephen T. Kent. Security Mechanisms in High Level Network Protocols. *ACM Computing Surveys*, 15(2):135–171, June 1983.

[86] Debby M. Wallner, Eric J. Harder, and Ryan C. Agee. Key Management for Multicast: Issues and Architectures. RFC2627, June 1999.

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

[88] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure Group Communications Using Key Graphs. In *Proceedings of ACM SIGCOMM '98*. ACM, September 1998.

[89] Lawrense Wos, George A. Robinson, and Daniel F. Carson. Efficiency and Completeness of the Set of Support Strategy in Theorem Proving. *Journal of the ACM*, 12(4):536–541, October 1965.