AT&T Labs - Research

subject: **IP Network Configuration for Traffic Engineering**

date: **May 26, 2000**

from: **Anja Feldmann**
Dept. HA173000
anja@research.att.com
HA173000-000526-02TM

Jennifer Rexford
Dept. HA173000
FP A169
973-360-8728
jrex@research.att.com
HA173000-000526-02TM

TECHNICAL MEMORANDUM

Traffic engineering involves tuning a network's path-selection, link-scheduling, and buffer-management policies to the prevailing traffic demands and performance requirements. Effective traffic engineering requires an accurate, network-wide view of the configuration of the routers. However, network operators have limited tools to aid in configuring large backbone networks. Configuring IP routers is extremely complicated, due to the diversity of network equipment, the large number of configuration options, and the interaction of configuration parameters across multiple routers. Manual configuration of individual routers can introduce errors and inconsistencies with unforeseen consequences for the operational network. In this paper, we discuss a methodology for obtaining an abstract, network-wide view of an IP backbone network that supports a wide variety of traffic-engineering tasks. We show how the network model can be populated from router configuration files and explain how, almost as a by-product, to check the correctness and consistency of the network configuration. Then, we describe the operational requirements on collection of the configuration data, evolution of the network model, and customization to a particular network. Finally, we describe the tool, netdb, that has been developed in collaboration with AT&T IP Services.

The paper is available on the web at
http://www.research.att.com/~jrex/papers/netdb.tm.ps
http://www.research.att.com/~jrex/papers/netdb.tm.pdf

1 Introduction

An autonomous system (AS) consists of a collection of routers and links managed by a single institution, such as a company, university, or Internet Service Provider. Engineering the flow of traffic through an AS involves tuning the path-selection, link-scheduling, and buffer-management policies to the AS's underlying topology, traffic demands, and performance requirements. Ultimately, the efficient operation of the network depends on the configuration of the individual routers. Configuration controls the selection of a wide range of parameters that affect the allocation of resources (e.g., link bandwidth and buffers), the operation of the routing protocols (e.g., BGP policies and OSPF weights), and access to the network (e.g., packet filters). Network operators configure routers when installing new equipment, enabling new features, and adapting to network failures and shifts in traffic demands.

Configuring a large backbone network is extremely difficult for a variety of reasons:

- **Consistency of neighboring routers:** The correct operation of an IP network depends on consistent configuration across a collection of routers. Correct configuration of each individual router does not ensure the correct operation of the network. For example, two directions of a point-to-point link should belong to the same OSPF area. Similarly, a BGP session requires compatible configuration of the pair of BGP speakers.
- **Network-wide implications of local changes:** Changes in the configuration of a single router can have network-wide implications. For example, increasing a link's OSPF weight could substantially increase the traffic load in some other part of the network. Some BGP configuration errors can cause disruptions in service in other parts of the Internet. Evaluating the impact of changes in routing policies is crucial for traffic engineering and requires a network-wide view of the topology and traffic demands. This has become increasingly important with the emergence of customers and applications that require performance guarantees, and the growing complexity of peering relationships.
- **Complex configuration options:** Router vendors offer a wide array of configuration commands and options. For example, Cisco's Internet Operating System (IOS) has over 600 commands. Configuration options and default parameter settings vary across different router products and IOS versions, and multiple types of routers may be active in the network at the same time. Configuring a large IP network requires substantial domain knowledge and attention to detail. In practice, routers are configured by a small number of local experts.
- **Rapid changes to the network:** Large IP backbones experience frequent changes in network topology and configuration. Routine events such as the addition of a new customer or peer, the installation of a new link, the enabling of measurement functions, and the upgrading of software require changes in router configuration. In addition, link failures and traffic fluctuations may require the network operators to tune the configuration of the routing protocols to alleviate congestion.
- **Limited configuration tools:** Configuration typically involves manual interaction with a command-line interface, or a primitive Web interface. Basic commercial tools provide templates for certain configuration operations, such as the provisioning of new customers, and higher-level languages support the specification of interdomain routing policies [1] . However, support for operating a large backbone networks is fairly limited. This problem is exacerbated by the fact that commercial configuration tools often lag behind the release of new high-end routers and advanced features.

In theory, router configuration could be driven by a database containing all necessary information about the IP network. However, complete automation of router configuration and network operation is extremely difficult in practice. Manual intervention is necessary for some configuration tasks, such as installing new equipment and adapting to network failures and shifts in user demands. However, manual configuration of individual routers can introduce errors and inconsistencies with unforeseen consequences for the operational network. Consider an OSPF-based backbone that provides service-level agreements (SLA) for IP telephony calls. Reducing the OSPF weight of a single interface could have a dramatic impact on the flow of traffic, resulting in a violation of the performance guarantees. Instead, traffic-engineering tasks should draw on an accurate model of the current network configuration, and any configuration changes should be checked for possible errors. These errors can occur at two different levels of abstraction. First, a mistake in executing the configuration change can introduce inconsistencies in the state of the network. Second, the routing change may violate higher-level policies such as SLAs.

In this paper, we present a model that represents an IP backbone network at a level of abstraction that is appropriate for traffic engineering. The model includes elements for the physical components (e.g., routers and interfaces), physical and logical connectivity (e.g., links and BGP sessions), routing protocols (e.g., OSPF and BGP), and end-host addresses (e.g., access lists and static routes). The model provides an *abstract*, *network-wide* view of the network configuration. This has proven to be crucial for a wide variety of traffic-engineering tasks [2–4] , such as deciding where to place additional capacity in the operational network or adjusting the tunable parameters for path selection, link scheduling, and buffer management. A network-wide view is necessary for predicting how configuration and topology changes would affect the flow of traffic in the backbone. An abstract view is useful for hiding low-level configuration details that do not affect the resource-allocation policies at the IP level.

Configuring an IP backbone network requires an understanding of routing protocols and the basic operation of IP routers. Section 2 presents this background material as part of deriving our network model. The remainder of the paper focuses on how to populate this model based on configuration data from an operational network. In Section 3, we describe router configuration files and identify the dependencies that arise within and between these files. These dependencies enable us to establish the relationship between the elements of the model and identify possible configuration errors. In Section 4, we discuss the operational requirements for a tool that parses configuration data, builds the network model, and checks for violations of the model. Then, in Section 5, we describe our realization of these ideas in a tool, netdb, that serves as a network database and debugger for the AT&T Common IP Backbone. We conclude the paper in Section 6 with a discussion of ongoing work.

2 Network Model

In this section, we present a simple network model that captures information about physical components of the network (routers, interfaces, and links), routing protocols (static routes, OSPF, and BGP), and access control (packet filters and route filters), as summarized in Table 1. Although the Internet consists of thousands of autonomous systems (ASes) operated by various institutions, our model focuses on the operation of the routers within a single AS, including the connections to neighboring networks. As part of describing our network model, we present background material on IP addressing, router architecture, and routing protocols, from the viewpoint of an Internet Service Provider (ISP).

Object	Attributes
router	router name, {loopback IP address}, location, {*interface*}, {global setting}
interface	*router*, interface name, {(IP address, IP prefix)}, capacity, OSPF weight, queuing strategy, status (up/down), {*access list*}, {*static route*}
link	*IP prefix*, link type (backbone/edge), OSPF area, {*interface*}
access list	*IP prefix*, permit/deny, {(*interface*, packet/route, in/out)}
static route	*IP prefix*, tag (administrative weight), {*interface*}
BGP	*router*, remote peer (IP address), remote AS, iBGP/eBGP, {filter policy}, {*interface*},{session attribute}

Table 1: Network model

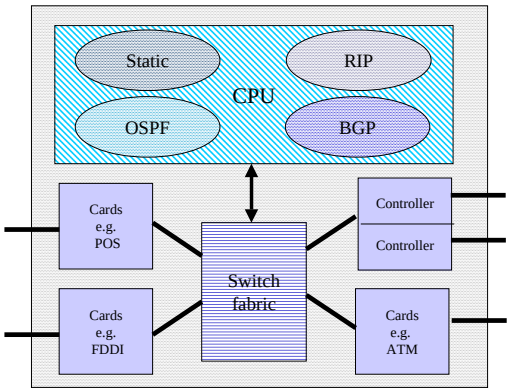

Figure 1: Components of an IP router

2.1 Router

An IP **router** receives incoming packets and directs them toward their destinations. A router typically consists of a route processor, a switching fabric, and a collection of interfaces, as shown in Figure 1. The route processor is identified by one or more **loopback IP addresses**, assigned by the network operators as part of configuring the router. The processor may run a variety of applications, such as a finger daemon or routing protocols, as indicated by **global settings**. The processor combines information from the intradomain and interdomain routing protocols to construct a forwarding table that is used to select the next-hop interface for each incoming packet. Rather than forcing each IP packet to travel through the route processor, most high-end routers have interfaces that can perform packet forwarding. The incoming interface directs the switching fabric to forward the packet to the appropriate outgoing interface.

Although most traffic travels directly from an incoming interface to an outgoing interface, certain packets must be directed to the route processor. The route processor typically handles packets that have IP or TCP options enabled and packets with expired time-to-live fields. The route processor handles any packets sent to or from the router's various interfaces, including the router's loopback IP addresses. This includes the routing protocol traffic exchanged with other routers. For example, OSPF requires the exchange of link-state updates with neighboring routers; similarly, the route processor may have one or more BGP sessions with other BGP speakers. Routine management functions also introduce traffic to and from the route processor. For example, the route processor may respond to ping requests and SNMP queries. Network operators may also Telnet to the loopback

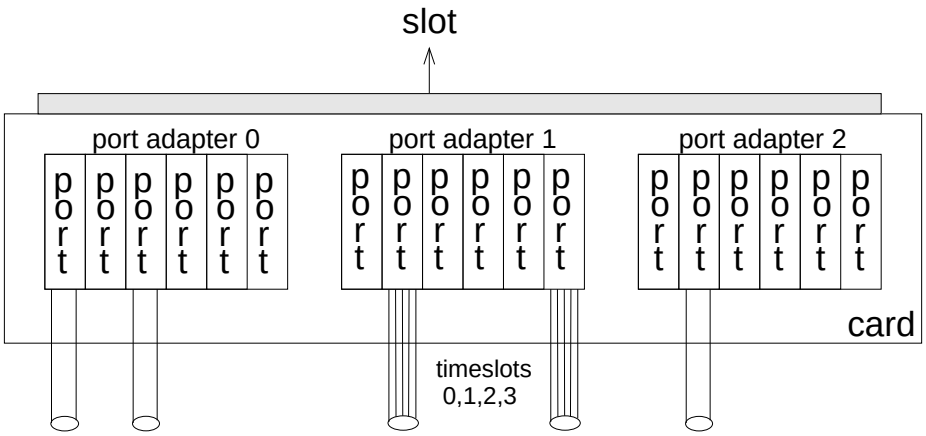

Figure 2: Card with three port adapters, each with multiple ports

address to configure or reboot the router. Knowing the loopback address, in addition to the router's **name** and geographic **location**, helps in keeping track of the assets in the network.

2.2 Interface

An **interface** receives incoming packets and queues and transmits outgoing packets. Each interface has a **name** that indicates its position in the router. Physically, an interface resides on a card that connects to a slot in the router's switching fabric, as shown in Figure 2. Following Cisco's IOS terminology, a single card consists of one or more port adaptors. A port adaptor has one or more ports that each connect to an underlying transmission medium, such as a fiber optic cable or an ATM link. In some cases, a port corresponds to an interface to a link connecting two or more routers. In other cases, the capacity of a single transmission medium may be subdivided into multiple time slots. For example, a port may subdivide its bandwidth into 16 time slots, each with 1/16 of the bandwidth. The router could be configured to assign one or more of these time slots to a single interface to a link. The abstraction is provided by a controller that directs outgoing packets to the time slots associated with this interface. Similarly, a single port to a layer-two ATM link may support multiple permanent virtual circuits, each corresponding to a different interface at the IP layer.

Each interface has a primary **IP address**, and may have one or more secondary IP addresses; each IP address is associated with a particular **prefix**. For example, an interface may have IP address 10.34.56.77 in the prefix 10.34.56.76/30. Associating an interface with multiple IP addresses provides a way to overlay multiple links or eBGP sessions on a single interface. An interface also has a **capacity** that depends on the bandwidth of the transmission medium. For a time-slotted transmission medium, the capacity depends on the number of time slots that have been allocated to the interface. Knowing the capacity is important for engineering the flow of traffic in the backbone. The interface also has an **OSPF weight** and a **queuing strategy**. An additional attribute indicates the **status** of the interface, whether it is up or down. An interface may be down due to a failure, or because the interface is in provisioning and has not yet been enabled to carry traffic.

2.3 Link

Neighboring routers exchange traffic over links. Each link is identified by an **IP prefix**, and each participating interface has a unique IP address with this prefix [5] . For example, the pre-

fix 10.34.56.76/30 consists of the IP addresses 10.34.56.76, 10.34.56.77, 10.34.56.78, and 10.34.56.79. The addresses 10.34.45.76 and 10.34.56.79 are typically reserved for the network address and the broadcast address, respectively. Addresses 10.34.56.77 and 10.34.56.78 can be used to identify the two ends of a bidirectional, point-to-point link. A shared medium, such as an Ethernet or an FDDI ring, may include more than two interfaces, resulting in a prefix with a smaller mask length. The IP addresses of the interfaces are assigned by the network operator as part of configuring the router. Interface IP addresses do not necessarily have any relationship to the loopback IP addresses of the incident routers. Likewise, the IP addresses of various interfaces at a router are not necessarily related.

Ultimately, then, each link is defined by an IP prefix, which includes one or more interface **IP addresses**. Each link is associated with a **type** — *backbone* or *edge* — that that depends on the number of interfaces. Network operators have complete control over each backbone link; configuration of an edge link requires coordination with the neighboring customer or peer. A backbone link connects two or more routers inside the AS, and an edge link connects to a neighboring customer or peer. Hence, an edge link has exactly one interface inside the AS, and a backbone link has two or more interfaces. For example, a large AS could have multiple edge links that connect to customers, such as business or university campuses, small regional providers, and local services like a modem banks for dial-up users or a Web-hosting complex. The AS may also have edge links that connect to other large providers via dedicated connections or public Internet exchange points. Each backbone link is associated with an **OSPF area**, as discussed below; in contrast, edge links do not participate in the intradomain routing protocol.

2.4 OSPF

The AS employs an intradomain routing protocol, such as OSPF or IS-IS, to select paths across the backbone [6] . Typically, customers and peers do not participate directly in these protocols. Under OSPF and IS-IS, each interface to a backbone link has an integer weight, assigned by the network operator. The routers exchange this information and compute shortest paths, where path length is defined as the sum of the link weights. Since these protocols typically use flooding to propagate link-state information, they do not scale to large networks. Therefore, large ASes typically introduce a routing hierarchy by dividing the backbone into multiple areas. Each backbone link belongs to a single OSPF area, and each interface to a backbone link has an OSPF weight. This structure is reflected in our model of links and interfaces, respectively. Ultimately, a router combines the information from the intradomain routing protocol (OSPF, IS-IS) with the interdomain reachability information (from static routes and BGP) to construct a forwarding table.

2.5 Static Route

Based on the forwarding table, the router can associate a packet's destination IP address with one or more "next-hop" interfaces. The scalability of the Internet routing infrastructure depends on the aggregation of IP addresses into prefixes, each consisting of a 32-bit IP address and a mask length. The terms prefix and IP network are often used interchangeably. **Static routes** provide a simple way to associate destination **prefixes** with edge **interfaces**. For example, suppose the AS has an edge interface to a customer with prefix 10.2.3.0/24. Then, the incident router could be configured with a static route to bind the prefix to the edge interface. This enables the customer to receive traffic destined to 10.2.3.0/24 without participating an interdomain routing protocol. To send traffic to the rest of the Internet, the customer could configure its network to direct any outbound traffic to the edge link. This obviates the need for the customer to acquire detailed routing information about the rest of the Internet.

A customer may connect to the AS with multiple interfaces to one or more routers for load balancing or fault tolerance. More generally, then, each static-route object concerns a particular prefix that is associated with a set of interfaces. The configuration of a static route ensures that the router knows to direct packets destined to this prefix to the appropriate next-hop interface. However, this does not ensure that the rest of the routers in the AS, and the rest of the Internet, know how to reach this destination prefix. To inform the rest of the network, the router can be configured to advertise the static route via an intradomain routing protocol (e.g., OSPF or interior BGP). The static route includes a **tag** (administrative label) that determines the attributes of these routing advertisements.

2.6 BGP

The AS also learns about destination prefixes via dynamic routing protocols, such as BGP. BGP is a distance-vector protocol that constructs paths by successively propagating reachability information [7, 8] . Each BGP advertisement concerns a particular prefix and includes the list of ASes along the path, as well as other attributes. BGP advertisements are exchanged over BGP sessions between pairs of routers. The two ASes would typically establish a BGP session between the incident routers; these routers are BGP peers. The ISP employs local policies to select a route for each destination prefix, and to decide whether to advertise this route to neighboring ASes. BGP policies can filter unwanted advertisements and assign local preferences, based on a variety of attributes. Then, the router executes the BGP decision process to select the best route to each destination prefix. BGP export policies determine whether, and what, to advertise to each BGP peer. Interior BGP (iBGP) sessions are used to distribute this information inside the backbone. A large AS may employ techniques such as route reflectors or confederations to avoid the overhead of having an iBGP session for each pair of routers (i.e., a full iBGP mesh).

Each **BGP** object corresponds to one end point of a BGP session. The attributes include the originating **router** of the BGP session and the **remote end point** of the BGP session. The remote end point is identified by IP address which may correspond to a particular interface or the loopback address. A BGP object also has a **remote AS**, a **peer group**, a set of **filter policies**, and a set of **session attributes**, capturing the various configurable parameters of a BGP session. The **iBGP/eBGP flag** distinguishes between interior and exterior sessions; a session with a remote end point inside the AS is classified as an iBGP session. The BGP object also includes a set of **interfaces** that defines how the router reaches the remote end point to exchange BGP messages. Interior BGP sessions rely on an intradomain routing protocol (e.g., OSPF) to direct traffic to the remote end point. BGP sessions with other ASes usually depend on explicit configuration of a set of interfaces that can carry the traffic toward the remote end point. For example, the remote end point may be reachable via a directly-attached network. In other cases, the router could be configured with static routes that indicate how to direct traffic toward the remote end point.

2.7 Filter

In addition to forwarding IP packets, an interface may perform various filtering functions. An AS may employ *packet filters* to control the traffic entering or leaving the backbone via a particular **interface**. **Access lists** identify which packets should be accepted or denied, based on fields in the IP headers such as source and destination addresses. To simplify operation, access lists are often specified using **IP prefixes**. For example, consider an interface that connects to a customer that has been allocated a known block of IP addresses. Packets entering the backbone via this interface can be filtered based on the source IP address, and packets leaving the backbone via this interface can be filtered based on the destination IP address. These addresses should lie within the customer's

range of IP addresses. Packet filtering helps detect spoofed source IP addresses and protects the customer from receiving traffic from unwanted sources [9] .

In addition, routers that participate in BGP may employ *route filters* to discard unwanted routing advertisements. A route advertisement consists of a destination prefix, an AS path, a next-hop AS, and a number of other attributes. Each BGP session can have route filters that discard advertisements based on any of these attributes. This plays an important role in protecting the AS, and the rest of the Internet, from misconfigured BGP policies in downstream routers. For example, suppose a customer connects to multiple service providers and mistakenly forwards advertisements from one large service provider to another. In the absence of route filtering, the two service providers may begin to exchange traffic via their common customer. An AS may also use route filters for load-balancing purposes. For example, suppressing certain advertisements from a neighbor on one or more BGP sessions allows the AS to direct traffic to alternate edge interfaces that may be less congested.

3 Router Configuration

Traffic engineering involves tuning an operational network to the prevailing traffic demands. As such, traffic engineering relies on the fundamental assumption of having an accurate and consistent abstraction of the underlying network. The complexity of large IP networks and the paucity of commercial configuration tools means that this is often an unrealistic assumption. Hence, populating the network model must be closely coupled with checking for possible configuration mistakes. In this section, we present a brief overview of the information available in router configuration files. Then, we identify the errors that can arise from dependencies within and between configuration files.

3.1 Potential Data Sources

Populating the network model requires information about the physical components in the network, as well as the configuration of the routing protocols and access lists. The information could come from a variety of data sources:

- **SNMP MIBs:** The Simple Network Management Protocol (SNMP) can be used to retrieve MIBs (Management Information Bases) that summarize the state of the router, including basic traffic statistics. However, the MIBs typically do not provide the full details of the configuration of the routing protocols and access lists.
- **Topology discovery tools:** Active measurement tools, such as traceroute and pathchar, can be used to infer various properties of the network, such as the topology, routing, and link capacity without having access to the network equipment. However, these tools cannot glean the full range of configuration parameters that affect the flow of traffic through the network.
- **Route monitoring:** Passive monitoring of the routing protocol traffic, such as BGP advertisements and OSPF link-state updates, provides an effective way to track the topology and routing state in the network. However, IP routing protocols do not convey information about link capacity, access control lists, or BGP policies.
- **Router configuration files:** Router configuration files provide a detailed view of the configuration of the routers in the network, including information about physical and logical connectivity, link capacity, routing protocols, and access lists. However, router configuration files do not provide an up-to-date view of the current status of each of the components.

Ultimately, none of the four techniques provides a complete view of an IP backbone, and each has an important role to play in tracking the state of the operational network. For the purposes of populating our network model, router configuration files provide the most complete information — the same information that is available to the routers themselves. In addition, traffic engineering tasks often require the network operators to modify the router configuration. This heightens the importance of checking that the router configuration files are correct and consistent.

3.2 Router Configuration Files

Commercial IP routers have a large number of configuration options that control the operation of the route processor and the various interfaces. For example, configuration determines which routing protocols are enabled (e.g., BGP, OSPF, IS-IS, RIP), as well as the selection of parameters (e.g., OSPF weight and area for each interface) and policies (e.g., import and export policies for each BGP session). Configuration also determines if and how the capacity of single physical medium (e.g., an ATM trunk or a channelized packet-over-SONET link) is partitioned across multiple layer-three links, and what packet filters are applied to incoming and outgoing traffic on a particular interface. Configuration of a router is achieved by applying commands to the router's operating system. These commands can be specified via a command-line interface or a simple Web-based interface, or by uploading a configuration file to the router.

The current configuration of the network is reflected in the configuration file of each router. In an operational network, these files are routinely logged for backup purposes and, as such, provide a valuable snapshot of the state of the network. An abbreviated example of a Cisco router configuration file is shown in Figure 3. A router configuration file resembles a C program, and the configuration of an entire network is analogous to writing software for a distributed system. In this analogy, each router represents a single task, and the connections between routers (e.g., links, BGP sessions) represent the communication between these tasks. However, in contrast to most high-level programming languages, existing router configuration languages have a very loose structure, limited nesting, and a large number of syntactic elements. This complexity reflects the continuing evolution of IP router technology. As IP network technology matures, new router specification languages may provide better structure and richer abstractions.

Currently, Cisco's Internet Operating System (IOS) [10] serves as a *de facto* standard for router configuration. The configuration file includes global settings and a number of sections that relate to different aspects of the router. For example, Figure 3 has four global settings — identifying the IOS version ("version xx.x"), disabling of the finger daemon ("no service finger"), the specification of the hostname ("hostname router1"), and the selection of classless interdomain routing ("ip classless"). Global settings have implications for the entire router. The IOS version, hostname, and finger daemon relate to the general operation of the route processor. The "ip classless" command enables the router to forward packets based on classless interdomain routing (CIDR) [11] . Other global variables that are not specified in the configuration file have default values.

In addition to global settings, the file includes a number of *sections* that relate to different aspects of router configuration. For example, Figure 3 includes a controller section with a single controller entry and an interface section with three interface entries. The Loopback0 interface is associated with the route processor, whereas Serial1/0/0:1 refers to a serial interface associated with channel group 1 of the T1 1/0/0 (slot 1, port adapter 0, and port 0). The controller section indicates that channel group 1 is bound to timeslot 12 on the channelized T1. The Serial1/0/0:1 interface has IP address 10.1.2.117 in the 10.1.2.116/30 network (with the 30-bit mask specified by 255.255.255.252), and is associated with access-control list 70 that is specified later in the file. The access-control list allows traffic for the 10.1.2.116/30 prefix and for the customer's prefix 172.12.4.0/24. The interface entry also includes a "description" statement, indicating that the link connects to customer abcde,

```
version xx.x
no service finger
!
hostname router1
!
controller T1 1/0/0
     channel-group 0 timeslots 8-11 speed 64
     channel-group 1 timeslots 12
!
.......
interface Loopback0
     ip address 10.126.236.3 255.255.255.255
!
interface Serial1/0/0:1
     description link to customer ABCDE
     ip address 10.1.2.117 255.255.255.252
     ip access-group 70 in
     bandwidth 56
!
interface POS2/0
     description to router in NY
     ip address 10.126.212.2 255.255.255.252
     ip ospf cost 5
!
.......
router ospf 2
     network 10.126.236.3 0.0.0.0 area 0
     network 10.126.212.0 0.0.0.3 area 1
.......
!
```

```
router bgp 7018
     bgp dampening
     network 10.1.3.0 route-map XXX1
     .......
     aggregate-address 10.252.0.0 255.255.0.0
     redistribute static
     neighbor intra-att-cluster peer-group
     neighbor intra-att-cluster remote-as 7018
     neighbor 10.126.236.94 peer-group intra-att-cluster
     .......
     neighbor 10.1.2.118 remote-as 65001
     neighbor 10.1.2.118 route-map XXX3 out
     .......
     no auto-summary
!
ip classless
ip route 10.1.2.116 255.255.255.252 Serial1/0/0:1
ip route 172.12.4.0 255.255.255.0 Serial1/0/0:1
ip community-list 10 permit bbbb
.......
access-list 70 permit 10.1.2.116 0.0.0.3
access-list 70 permit 172.12.4.0 0.0.0.255
.......
route-map XXX1 permit 10
     match community 10
.......
!
.......
```

Figure 3: Sample IOS router configuration file

and a "bandwidth" statement indicating the link capacity. The POS2/0 interface (slot 2, port adapter 0) refers to a packet-over-SONET backbone link that participates in OSPF routing.

The router section has entries for the various routing protocols, such as OSPF, BGP, and static routes. Each statement in the OSPF entry associates a network address with an OSPF area. The Loopback0 interface (address 10.126.236.3 and a 32-bit mask) is associated with area 0 (the backbone area) and the POS2/0 interface (in network 10.126.212.0 with a 30-bit mask) is associated with area 1. The configuration file also specifies static routes that associate destination prefixes with a particular interface (e.g., the two "ip route" commands involving Serial1/0/0:1). The BGP entry identifies the AS number (7018) and includes a number of commands that enable/disable certain features (e.g., route dampening and auto-summary) and control the aggregation of route advertisements involving certain IP prefixes (e.g., the "aggregate-address" command). In addition, the "neighbor" commands are used to associate the router with a particular route-reflector group for iBGP (e.g., the "neighbor" commands involving the intra-att-cluster) and to configure eBGP sessions with a router in another AS (e.g., the "neighbor" commands with IP address 10.1.2.118 in AS 65001). Each BGP session is associated with import and export policies, specified via route-maps that appear later in the configuration file.

3.3 Dependencies Within a File

Interdependencies between various parts of the configuration file can result in unintentional errors. Parsing and checking a configuration file is similar to compiling a C program. The compilation process must check for references to undefined variables and inconsistent definitions of each variable. In addition, some compilers generate a warning when a variable is defined but never used, or when a variable is used without being initialized. Similar checks can be applied to router configuration files. In processing configuration commands, the router checks for basic syntax errors. However, the router does not detect other kinds of configuration mistakes. Instead, a configuration error could cause a router, link, or routing protocol to function incorrectly. For example, a misconfigured interface could result in a link that cannot carry traffic. In other cases, a configuration mistake could cause the router to use default parameter values. For example, a misconfigured access-control list could cause an interface to accept *any* traffic rather than performing the intended packet-filtering functions.

We distinguish two broad classes of errors — those that do not require any domain knowledge and those that do. The first class includes using undefined items or leaving items unused. The errors in the second class are concerned with the internal consistency of the router configuration. These errors arise from inconsistent definitions or dependence on default parameters. Although the router resolves such violations, the default handling may result in different behavior than intended by the human operator.

3.3.1 Domain-Independent Violations

Referencing undefined items: Many of the statements in a router configuration file refer to information specified in other entries. For example, the Serial1/0/0:1 entry refers to channel-group 1 defined in the 1/0/0 controller, and to access-group 70 defined in the access-list section. Similarly, the "neighbor 10.1.2.118 route-map XXX3 out" command refers to route-map XXX3. This introduces dependencies between the various parts of the configuration file that have implications on the population of the data model and the application of error checks. For example, the specification of interface Serial1/0/0:1 cannot be fully understood without parsing the controller and access-list sections. In addition, references to undefined items should be flagged as errors. For example, referencing an access-group 60, instead of access-group 70, should generate an error since

this access-control list is undefined. On the other hand, specification of an interface Serial1/0/0:2 is not possible since controller 1/0/0 does not define channel-group 2.

Unused items: In addition to missing or inconsistent definitions, the configuration file may define items that are never used. For example, the file might specify an access-list 80 that is never associated with an interface, or a route-map XX4 that is never associated with a BGP session. These extra definitions do not affect the operation of the router and, hence, are not errors. Still, generating warnings about unused items is useful to enable the network operator to remove unnecessary entries or fix mistakes. The unused entries could lead to erroneous assumptions or errors in the future. For example, the operator may associate a new interface with access-list 80 without realizing that some prefixes have already been associated with this access-control list. In addition, some unused entries may result from mistakes in configuring the router. For example, the operator may have meant to type "70" (to associate the access-list with Serial1/0/0:1) rather than "80."

3.3.2 Domain-Dependent Violations

Inconsistent definitions: The router configuration language permits inconsistent definitions. For example, the bandwidth of an interface can be specified in multiple ways, including the "speed" field in the channel-group command and the "bandwidth" command in the interface entry. These two definitions may not be consistent with each other, or with the actual capacity of the underlying communication medium. This may have several negative consequences. First, inconsistent definition of interface capacity can result in incorrect SNMP statistics for link utilization, since utilization is computed relative to the link capacity. Second, the interface capacity plays an indirect role in defining other configurable parameters. For example, an interface participating in OSPF has a *default* weight (cost) that is inversely proportional to capacity. Inconsistent definitions can also arise from the global settings. For example, suppose that configuration file did *not* include the "ip classless" statement. This could cause the router to discard packets destined to an IP prefix that is not aligned with octet boundaries.

Dependence on default parameters: Many configuration options have default parameter settings. In some cases, dependence on default parameters may be dangerous. Consider the process of configuring an interface to participate in OSPF. This involves assigning an OSPF *weight* in the interface section and an OSPF *area* in the router section. Inadvertently skipping either of these two steps has important consequences. If the router section does not assign an OSPF area, then the interface does not actually participate in OSPF (despite the fact that the interface entry includes an OSPF weight). On the other hand, suppose that the router section does assign an OSPF area. Then, the link does participate in OSPF. If the interface entry does not assign an OSPF weight, then a default value is used (inversely proportional to interface capacity). However, the operator may have simply neglected to assign an OSPF weight. The use of the default value could have significant impact on the selection of routes in the network. Generating a warning is useful to prevent such mistakes.

3.4 Dependencies Across Files

Correct operation of the network depends on the *interaction* between the various routers. Having a correct and consistent configuration file for each router is not sufficient. The interaction between routers implies additional dependencies. This is conceptually similar to the dependencies that arise in linking a collection of software modules. These software modules should avoid having multiple definitions of global variables or inconsistent interfaces between the modules.

Inconsistent definitions: A configuration file contains many entries that have significance at the *router* level. For example, defining access-list 70 in one file does not affect any other router.

Similarly, the same prefix could be associated with access lists in more than one router. For example, an ISP may connect to a customer over multiple edge interfaces at different routers; each of these interfaces may have an access list with the same set of customer prefixes. Other parts of the configuration file have network-wide significance. Consider a backbone link with interfaces on two routers. If the link participates in OSPF, then the two routers should agree on the selection of an OSPF area; otherwise, the link cannot carry traffic. The two routers may need to agree on a variety of other configuration options (e.g., cyclic redundancy check algorithm). Similarly, the correct functioning of an iBGP session requires consistent configuration of the two end points.

Inconsistent references to remote nodes: For an eBGP session, one of the two end points resides outside of the backbone, on a router controlled by another organization. This limits the opportunity to check the consistency of the configuration. In some cases, an ISP has multiple eBGP sessions to the same remote end point (i.e., the same remote IP address). These eBGP sessions should refer to the same remote autonomous system. For example, suppose one router has the statement "neighbor 10.1.2.118 remote-as 65001" and another router has the statement "neighbor 10.1.2.118 remote-as 65002." This would be a mistake. If two configuration files refer to the same object, they should have the same view of this object. In fact, this observation can be applied in a variety of situations, including iBGP sessions where the remote end point resides inside the autonomous system.

The dependencies across configuration files have several important implications. First, the dependencies within a file enable us to identify the relationships between the router, interface, access-list, and static route objects. For example, an interface is associated with a particular router, a set of access lists, and a set of static routes. This configuration information is spread throughout the file. Second, the dependencies between files can be exploited to derive abstractions such as links and BGP sessions that represent the physical and logical connectivity, respectively, between the various routers in the backbone. For example, each link is associated with a prefix (e.g., 10.1.2.116/30) and one or more interfaces. The interfaces belong to different routers and, hence, are specified in different configuration files. Yet, these interfaces can be associated with each other based on the common prefix. Similarly, each iBGP session is associated with a pair of end points, identified by IP addresses. The configuration file of each router identifies the IP address of the other end point in the "neighbor" statements. Third, the dependencies within and between files define a natural ordering for processing the configuration data as we create the objects in the network model and identify configuration errors, as discussed in more detail later in Section 5.1.

4 Operational Considerations

Router configuration files provide a wealth of information that can be used to populate the network model. In this section, we discuss the practical issues that arise in developing a tool that populates the network model and identifies configuration errors in an operational IP backbone. Operational demands and conventions impose additional requirements on the collection of configuration data, the evolution of the network model, and the customization of the tool.

4.1 Access to Configuration Files

Any tool for populating the network model will be based on assumptions about access to the router configuration data. The configuration files must be accurate in two key respects. First, configuration data should be available for *all* of the routers in the IP backbone. Missing configuration files can lead to incorrect error messages and mistakes in the network model. Second, the files must provide a *consistent* snapshot of the configuration of the network. That is, the configuration should not change

while the files are collected from the various routers! For the most part, these two requirements are satisfied by a routine backup strategy. The entire set of routers can be polled during a short period of time to log the configuration files. Care can be taken to ensure that the configuration changes do not occur during this time period. In the unlikely event that one or more routers are reconfigured, the logging process can be repeated to provide a consistent view of the network.

Identifying the relationships between objects in the network model depends heavily on the notion of IP addresses. IP addresses are used to identify routers and interfaces, and to associate these components with links, BGP sessions, and access-control lists. The correct functioning of a tool depends on the assumption that the IP addresses are unique. Using the same IP address for two active interfaces constitutes a serious misconfiguration of the network. Assigning an existing IP address to an inactive interface is permissible but not advisable — a simple command to enable the inactive link could introduce a serious problem in the operational network. Indeed, the tool should detect duplication of an IP address and flag this as an error. The assumption that every interface, once provisioning has completed, has at least one IP address, and that these IP addresses are unique, is quite reasonable. An AS should have a sufficiently large set of IP addresses to assign to the routers and interfaces.

4.2 Extending the Network Model

Despite capturing the key components of an IP backbone, the model in Table 1 does not cover all aspects of IP networking. For example, the model does not include information about all possible routing protocols (e.g., IS-IS, RIP, and MPLS are not included), and the existing objects do not include all possible parameters. Rather, the network model provides a basic framework which can be augmented as needed. Changes occur on a variety of time scales:

- **New parameter settings:** Day-to-day operation of the network results in changes in the values of parameters for existing objects. For example, the operator may tune an interface's OSPF weight in response to network congestion. Similarly, installing additional interface or router would result in additional instances of the various kinds of objects.
- **New parameters:** Extensions to the definition of an object occur less often, only when a new feature or command is introduced into the network. For example, the network operator may enable weighted RED (Random Early Detection) on an interface. This would require adding new parameters to the interface object to represent the tunable parameters for WRED. Adding a new attribute to an existing object requires a minor extension to the network model.
- **New objects:** The IP network occasionally undergoes significant architectural changes, such as the introduction of new routing protocols (e.g., MPLS or multicast). New protocols require the addition of new objects. Fortunately, major changes do not occur very often in an operational network. In addition, these architectural shifts are planned well in advance of the actual changes in the configuration files.

The development of effective tools is hampered by the substantial complexity of router configuration files. Configuration languages, like Cisco IOS, have hundreds of commands that support a wide range of router products. Yet, in practice, operational networks are remarkably homogeneous, in terms of the link-level technologies, router types, and configuration options. A tool for populating a network model can, and should, capitalize on this homogeneity. Rather than understanding, and modeling, all possible configuration options, the tool should focus on the limited set of commands applied in the operational network. In addition, a tool for populating a network model for traffic engineering can, and should, ignore commands that do not affect the network model. This offers a

significant reduction in complexity, and enables the tool to keep pace with the introduction of new features and new router products. In contrast, commercial tools [12] have very complex network models since they cannot be tailored towards a specific operational network. Therefore, they must handle the full range of possible configuration options. Hence, such tools typically fall behind the deployed base of router hardware and software.

At the same time, the tool should not prohibit the use of new commands. A relatively simple approach is to maintain a list of the configuration commands supported by the tool. Each command has well-understood implications on the network model and the error checks. Upon encountering a new command, the tool should generate an error message to alert the network operator to consider the implications (if any) on the network model. In contrast, some commercial tools parse unknown commands without performing checks, assuming that these commands relate to new router features that are not supported in the current version of the tool. However, support for error and consistency checking is most important precisely when new router features are enabled! Generating an error would enable the tool to detect inadvertent use of a new command or feature in the production network. Flagging an error also ensures that people, and systems, that rely on the network model are informed when the underlying model changes. When this occurs, the tool can be extended to support the changes to the model and to incorporate new error checks.

4.3 Customization

Operating a large IP backbone requires more than correct configuration of the routers. Often, network operators obey local guidelines in configuring the routers. The network model and consistency checks can be customized based on these conventions. These extensions could draw on information from a variety of sources:

- **Router hostname:** By convention, the router hostname may convey additional attributes about the router. For example, the name could include the city where the router is located. This provides an inventory of the equipment in each city and supports visualization of the network topology. The hostname may also include information about the router type or version, which may be used to define additional consistency checks.
- **Input files:** Additional information could be conveyed in separate data sources. For example, an input file could indicate the location (in latitude and longitude) of each city or router. Similarly, an input file could indicate what type of commercial relationship the AS has with each of the neighboring autonomous systems. This information can be used to classify each edge link as a customer, peering, or provider link [13] , based on the remote AS of the BGP session. The classification may have implications on the desired BGP import and export policies [14, 15] . Indeed, policies such as BGP import and export policies or default access lists could be contained in a separate input file.
- **Description fields:** Extra information may be embedded in description fields in the router configuration files. For example, the description could indicate or the length of each link (e.g., in terms of fiber miles or propagation delay). For edge links, the description field could include the name of the neighboring customer or peer. The description fields could also be used to convey information about the underlying layer-two topology (e.g., which ATM links are traversed by a particular PVC); this information is not typically available at the IP level.
- **Configuration parameters:** The network operator may assign particular meaning to certain configuration parameters. For example, a certain (large) OSPF weight could be used to identify new links that are still in the testing/provisioning phase. Likewise, BGP parameters, such as community sets and local preferences, may have specific values (or ranges of values) that can be used to classify the router, the BGP session, or the remote AS.

Exploiting local conventions is a crucial part of applying the tool to an operational network, since several important parts of the network model are not visible at the IP level (e.g., geographic locations, propagation delays, layer-two connectivity, commercial relationships with neighboring ASes, and customer/peer names). The tool should be able to incorporate this information.

Besides allowing extensions to the network model, local conventions could be used to define additional error and consistency checks. These guidelines include default global settings, access-control lists, route maps, communities, and AS paths that should appear in every router configuration file. In addition, the checks could be customized based on the type of the router or link. For example, an eBGP session with a customer may employ different import and export policies than a session with a peer. These BGP policies can be codified in separate input files, and checked against the route-maps specified for each BGP session in the router configuration files. Similarly, conventions for iBGP configuration can be codified in a separate input file. For example, an AS may use route reflectors to reduce the overhead of distributing BGP information inside the backbone. An AS may have policies for configuring the route reflectors (e.g., in a full mesh) and the clients (e.g., every client communicates with at least two route reflectors).

5 Netdb

In collaboration with AT&T IP Services, we have developed a tool, netdb, that populates the network model and detects possible configuration errors for the AT&T Common IP Backbone. The network consists of hundreds of routers and thousands of links, including connections to the WorldNet modem banks, the Web-hosting platform, business customers, and other tier-one network providers. Netdb is a Perl script that parses configuration files in Cisco IOS format. Running netdb on the configuration files for the operational network requires less than two minutes. In this section, we describe the design and operation of netdb and present sample error messages that illustrate what kind of configuration problems that the tool can detect.

5.1 Parsing and Debugging

Router configuration files have poor structure, with unusual nesting and forward and backward dependencies. To aid in parsing, netdb reads through the files to create a table that stores every line, as shown in Figure 4. Then, for each router, netdb creates data structures for global settings and sections — controllers, interfaces, access lists, route maps (including communities and AS paths), static routes, OSPF routes, and BGP routes. The set of known global settings and section names is relatively small, and is provided in an input file. Each section can include multiple entries. For example, the interface section in Figure 3 has three interface entries. For each router and each section, netdb stores a list of entries. For each entry, netdb stores the set of commands from the router configuration file. For example, the interfaces for the sample configuration file of router 'router1' would contain "Loopback0|Serial1/0/0:1|POS2/0" while the commands for "router1|POS2/0" would be "description|ip address|ip ospf", where "|" is used as a field separator. Netdb stores the value associated with each attribute in a hash table. For example, "router1|POS2/0|ip address" would map to "10.126.10.2 255.255.255.252".

Netdb generates an error upon encountering an unfamiliar global setting or section name. After identifying section boundaries, netdb parses and checks the global settings, and checks if all desired global variables have been specified. As part of processing each global setting, netdb assigns the associated parameter in the network model. This process repeats across all of the routers. Next, netdb parses each of the sections, one at a time, across all of the routers. The processing order derives from the dependencies identified in Section 3 — controllers, access lists, interfaces, other

```
read configuration files of all routers
read keywords for global settings and section names
forall routers {
    identify section boundaries
    parse global variables
    check global variables
}

foreach section in (controllers, access lists, interfaces,
                other filter sections, static routes, OSPF, BGP) {
    read section keywords
    read customization input files
    forall routers
        parse section and check keywords, network model violations
    forall routers
        perform error checks
}

forall routers {
    forall objects
        report unassigned attributes
    forall statements
        report unused statements
}
```

Figure 4: Netdb processing order

filter sections, static routes, OSPF, and finally BGP. Earlier sections do not depend on later sections (e.g., a controller specification does not depend on the access lists). Often, a single error (such as an incorrect interface IP address), manifests itself in multiple places in the configuration files. Respecting the dependencies between sections permits netdb to process the configuration files in a single pass, and to identify errors at the lowest possible level.

In addition, processing the sections in this order simplifies the process of populating the network model and identifying errors. For example, consider an interface entry that refers to an access-control list. By the time the interface entry is processed, all of the access-control lists have already been parsed. Netdb flags an error if the interface entry refers to an access-control list that does not exist. Otherwise, netdb associates the interface object with the appropriate access-control list. Similarly, consider the processing of the OSPF "network" statement that associates an IP prefix with a particular OSPF area. By the time the OSPF section is processed, netdb has already parsed all of the interface sections across all of the routers. As such, netdb already knows which interfaces have IP addresses in this prefix. If two or more interfaces fall within this prefix, netdb can infer the existence of a backbone link in a particular OSPF area. Otherwise, netdb can flag an error. If another router's configuration file has an OSPF "network" statement that applies to the same IP prefix, netdb can check that the two statements assign the same OSPF area.

In the process of parsing a section, netdb creates objects, assigns the attributes of existing objects, and generates error messages. The link objects are the only class of objects that does not correspond to a section. Link objects are created as netdb parses the interface entries within the interface section. For each interface entry, netdb considers the IP prefix associated with the interface's IP address. For the first occurrence of a prefix, netdb creates a new link object. The link object is associated with the IP prefix and with the IP address of the particular interface. If netdb encounters the prefix in another interface entry, the existing link object is extended to include the IP address in the set of interface addresses.

Following our philosophy of extensibility, each section has an input file that specifies the set of expected keywords. For example, a file for OSPF-related keywords could include

```
1|ospf log-adjacency-changes
2|network
2|neighbor
```

The first field indicates whether the keyword should appear at most once ("1") or can appear multiple times ("2") in an entry. For example, a single OSPF entry can have multiple network and neighbor statements. A keyword may consist of multiple words, separated by white spaces. Hence, in parsing the configuration files, netdb performs a longest-prefix match to associate each statement with the appropriate keyword.

Customization also follows our philosophy of extensibility, each section may have one or more input files that specify additional information or policies and default values.

After processing all of the sections, netdb has two remaining error-checking tasks. First, some of the objects may have attributes that have not been assigned. For example, each backbone link should belong to an OSPF area, specified in an OSPF "network" statement. Hence, netdb sequences through the link objects to generate an error for each backbone link that does not belong to an OSPF area. Second, some statements in the router configuration file may not relate to any existing object. For example, a configuration file may define an access-control list that is never associated with a particular interface or BGP session. As part of populating the network model, netdb keeps track of which statements have been applied one or more times. As the final stage of processing, netdb generates warnings for statements that were never applied.

```
router1: unknown interface keyword: hold-queue value: .....
router2: ROUTE-MAP ERROR: community 1000 undefined ROUTEMAP1: community 1000 1010
router3: OSPF ERROR: ospf network 10.127.6.132/30 should either be in area 14 or 3
router4: OSPF ERROR: network: 10.126.212.0 0.0.0.3 area 2 with only one IP address 10.126.212.2
router5: OSPF ERROR: network: 10.126.12.172 0.0.0.3 area 3 with no IP address
router6: BGP ERROR: cannot resolve IP: 10.11.12.56 from ...BGP statement...
```

Figure 5: Examples of netdb error messages

```
router1: GLOBAL ERROR: missing parameter BGP-COMMUNITY
router2: GLOBAL ERROR: incorrect parameter CEF value: ip cef
router3: ACL WARNING: default acl 6 missing
router3: ACL WARNING: default acl 7 differs from specification: deny 172.0.0.0/8
router4: ACL ERROR: VPN customer needs in and out ACL; (customer ABCDE)
router5: CONTROLLER ERROR: missing clock sync for interface Serial2/1/0:2
router6: BGP ERROR: wrong rr definition for rr client 10.126.236.3 with peer-group abc
```

Figure 6: Examples of netdb error messages for policy violations.

5.2 Error-Checking Examples

Netdb's error checks generate a variety of error messages, as shown by the sample output in Figure 5. The first example illustrates how netdb flags unknown entries. In the second example, netdb flags the fact that community 1000 is referenced but not defined. Both of these errors relate to mistakes in a single router configuration file. The third, fourth, and fifth examples illustrate OSPF problems — a backbone link that has two interfaces with different OSPF areas, the assignment of an OSPF area to a link that has only one interface (i.e., an edge link), and the assignment of an OSPF area to a link that has no interfaces. Detecting these three errors requires joining information across multiple router configuration files. The sixth example concerns an eBGP session where the associated router does not have a route to the BGP peer in the neighboring autonomous system. The router configuration file should either define a static route to the remote IP address or specify that the peer is reachable via an attached interface.

Netdb performs a wide variety of consistency checks; Figure 6 presents a small example. The first two messages show that particular global settings are not set to their desired default values. These kinds of mistakes can arise when a new router is added to the network. The third message concerns a missing access list. Unlike the errors discussed in Section 5.1, this message concerns the absence of a *default* access list, rather than a reference to a non-existent access-control list. The fourth message identifies an access-control list that differs from the expected configuration. The fifth message identifies a VPN customer for which either the access-control list on the input or on the output side was missing. The sixth message indicates that a particular interface entry is missing a statement related to clock synchronization. The seventh message concerns a misconfigured route-reflector client. Each of these messages identify violations of local conventions, rather than actual configuration errors. Yet, detecting deviations from local policies is critical to the "correct" operation of the network.

6 Conclusions

Traffic engineering is based on the fundamental assumption that its view of the network is accurate and that the abstractions it uses are internally consistent. In this paper we present a network model for traffic engineering and show how to check the actual configuration of the network with regards to

its consistency. We propose an approach to populating the network model by parsing the configuration files of all of the routers in the network. Our methodology capitalizes on the dependencies within and between configuration files to derive the network model and identify configuration errors. We consider the operational requirements for collecting configuration data, evolving the network model, and customizing the error checks. Our approach has been realized in a tool, netdb, that performs debugging and database functions for the AT&T Common IP Backbone.

As part of our ongoing work, we are investigating higher-level consistency checks that exploit having a network-wide view of the backbone topology and routing configuration. We can test a number of important assertions about the (physical and logical) topology by combining the network model with an accurate model of intradomain routing [2] . For example, we could test whether a layer-three link resides on at least one shortest path, and whether a BGP route reflector is close (in the OSPF sense) to each of its route-reflector clients. Similarly, the routing model could be combined with information about customer subscriptions for guaranteed services, such as IP telephony. By applying the routing model to the current network configuration, we can ensure that each link has sufficient bandwidth resources to accommodate the high-priority traffic. Applying these higher-level checks can help network operators handle the growing challenge of managing large IP backbone networks.

Acknowledgments

Many thanks to Jay Borkenhagen, Mike Langdon, Kevin McClarren and Han Nguyen, and many others within AT&T IP Services for their help in understanding the operation and configuration of IP routers. Thanks also to Joel Gottlieb, Albert Greenberg, Carsten Lund, Nick Reingold, and Fred True for their help in debugging the output of Netdb. We would also like to thank Joel Gottlieb, Lixin Gao, and Jan Bankstahl for their comments on an earlier version of the paper, and Tim Griffin for his detailed suggestions about the structure and focus of the paper.

References

[1] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra, "Routing Policy Specification Language (RPSL)." Request for Comments 2622, June 1999.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, "NetScope: Traffic engineering for IP networks," *IEEE Network Magazine*, March 2000.

[3] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," in *Proc. ACM SIGCOMM*, August/September 2000.

[4] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, March 2000.

[5] F. Baker, "Requirements for IP Version 4 Routers." Request for Comments 1812, June 1995.

[6] J. Moy, *OSPF: Anatomy of an Internet Routing Protocol.* Addison-Wesley, 1998.

[7] B. Halabi, *Internet Routing Architectures.* Cisco Press, 1997.

[8] J. W. Stewart, *BGP4: Inter-Domain Routing in the Internet.* Addison-Wesley, 1999.

[9] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing." Request for Comments 2267, January 1998.

[10] Cisco, "Cisco IOS Configuration Fundamentals," 1998. Documentation from the Cisco IOS reference Library.

[11] V. Fuller, T. Li, J. Y. Yu, and K. Varadhan, "Classless inter-domain routing (CIDR): An address assignment and aggregation strategy." Request for Comments 1519, September 1993.

[12] Cisco, "Cisco Netsys connectivity service manager." http://www.cisco.com/univercd/cc/td/doc/pcat/necosv.htm.

[13] G. Huston, "Interconnection, peering, and settlements," in *Proc. INET*, June 1999.

[14] C. Alaettinoglu, "Scalable router configuration for the Internet," in *Proc. IEEE IC3N*, October 1996.

[15] L. Gao and J. Rexford, "Stable Internet routing without global coordination," in *Proc. ACM SIGMETRICS*, June 2000.

Anja Feldmann

-HA173000-AF/JR **Jennifer Rexford**