IP Network Configuration for Intradomain Traffic Engineering

Anja Feldmann
Universität des Saarlandes
Saarbrücken, Germany

Jennifer Rexford
AT&T Labs–Research
Florham Park, NJ, USA

anja@cs.uni-sb.de jrex@research.att.com

Abstract

The smooth operation of the Internet depends on the careful configuration of routers in thousands of autonomous systems throughout the world. Configuring routers is extremely complicated because of the diversity of network equipment, the large number of configuration options, and the interaction of configuration parameters across multiple routers. Network operators have limited tools to aid in configuring large backbone networks. Manual configuration of individual routers can introduce errors and inconsistencies with unforeseen consequences for the operational network. In this paper, we describe how to identify configuration mistakes by parsing and analyzing configuration data extracted from the various routers. We first present an overview of IP networking from the viewpoint of an Internet Service Provider (ISP) and describe the kinds of errors that can appear within and across router configuration files. To narrow the scope of the problem, we then focus our attention on the configuration commands that relate to traffic engineering—tuning the intradomain routing protocol to control the flow of traffic through the ISP network. We present a case study of a prototype tool, developed in collaboration with AT&T IP Services, for checking the configuration of the AT&T IP Backbone and providing input to other systems for network visualization and traffic engineering.

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. 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 configurations 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.

- **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 network 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. Checking the router configuration for errors and inconsistencies is a crucial part of the day-to-day operation of a large network. In addition, having an accurate network-wide view of the current topology and configuration is a prerequisite for a variety of higher-level tasks, such as traffic engineering and capacity planning. 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 the many low-level configuration details that do not affect the resource-allocation policies at the IP level.

This paper describes our experiences in checking for configuration mistakes and constructing an abstract view of the network by parsing and analyzing the configuration data available from each router. Configuring an IP backbone network requires an understanding of routing protocols and the basic operation of IP routers, as discussed in Section 2. In Section 3, we describe router configuration files and identify the dependencies that arise within and across these files. Checking for configuration mistakes involves comparing the configuration of the routers against a network model. In Section 4, we discuss the challenges of defining a general model that captures all aspects of router configuration. We focus our attention on a relatively simple

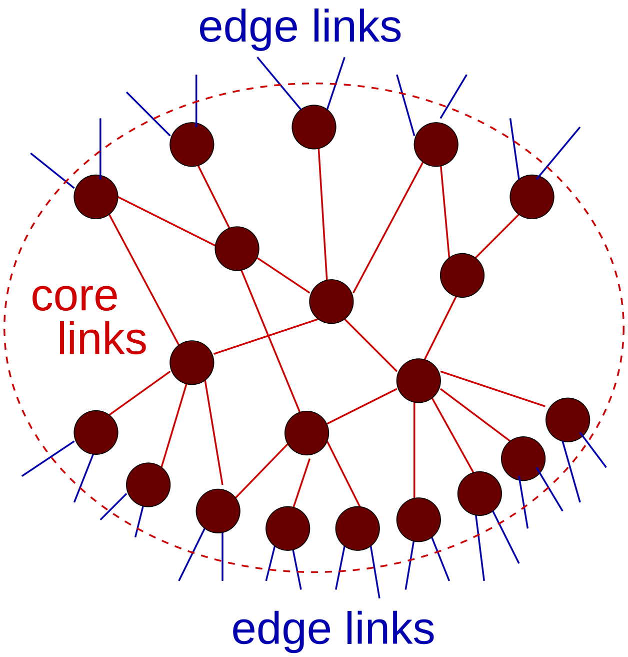

Figure 1: Topology of an IP backbone network

model that captures the key information needed to adapt the configuration of the intradomain routing protocol to the prevailing traffic. Then, Section 5 describes the application of the tool to check for configuration errors in the AT&T IP Backbone [2] and to provide an abstract, network-view view for a variety of traffic engineering studies [3, 4, 5]. We summarize the contributions of the paper in Section 6.

2 ISP Backbone Network

In this section, we present an overview of IP networking from the viewpoint of a large Internet Service Provider (ISP). The discussion focuses on a network that employs static routes and the Border Gateway Protocol (BGP) to exchange routing information with neighboring domains, and the Open Shortest Path First (OSPF) routing protocol to select paths within the ISP backbone.

2.1 Network Architecture

The Internet consists of thousands of autonomous systems (ASes) that interact to coordinate the delivery of IP traffic. An AS is a collection of routers and links administered by a single institution, such as a company, university, or ISP. The routers within the ISP backbone are interconnected using *core* links, as shown in Figure 1. An ISP has complete control over the configuration and operation of a core link, by virtue of controlling the adjacent routers. The *edge* links include *access* links that connect to the ISP's customers, such as business or university campuses, small regional providers, and local services like modem banks for dial-up users or a Web-hosting complex. The ISP may also have *peering* links that connect to other large providers via dedicated connections or public Internet exchange points. Configuration of access and peering links depends on interaction with the customers and the peers, respectively. In practice, an ISP may have multiple connections to the same customer or peer for load balancing and fault tolerance.

The delivery of traffic through an ISP network depends on the interaction between the various routers. A router typically consists of a route processor, a switching fabric, and a collection of interfaces, as shown in Figure 2. 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 and routing protocols. 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. Each entry in the forwarding table concerns a particular *network address* or *prefix*, represented by a 32-bit address and a mask length. With the advent of Classless Inter-Domain Routing (CIDR) [6], any mask length from 0 to 32 is permissible. For example, the prefix 172.12.4.0/24 has a network address of 172.12.4.0 and a 24-bit mask (i.e., 24 1s followed by 8 0s). This prefix consists of 256 IP addresses ranging from 172.12.4.0 to 172.12.4.255. A forwarding table has entries such as

172.12.4.0/24	10.1.2.118	Serial1/0/0:1
12.128.0.0/9	10.126.212.1	POS2/0

The first entry directs packets destined for 172.12.4.0/24 to interface Serial1/0/0:1 to reach the next hop with IP address 10.1.2.118. When a packet arrives, the router performs a longest prefix match on the destination address to find the appropriate forwarding-table entry for the packet. For example, a packet with destination address 172.12.4.103 would match the first entry. Then, the router forwards the packet to the appropriate outgoing interface. In some cases, the forwarding entry has multiple outgoing interfaces and the router can pick a single element from this set. The next router repeats the process, forwarding the packet closer to its destination.

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 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. In addition, the route processor may respond to ping requests and Simple Network Management Protocol (SNMP) queries. Network operators may also Telnet to the loopback address to query, configure, or reboot the router.

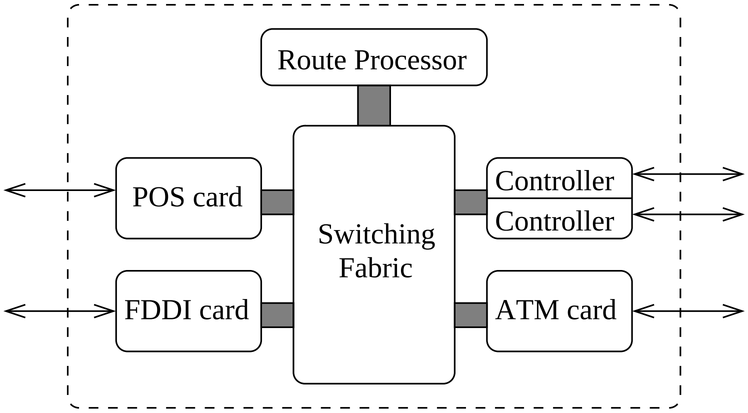

Figure 2: Components of an IP router

2.2 Interfaces and Links

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 3. 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 Packet-Over-SONET (POS), FDDI, or ATM link. In the simplest case, each port corresponds to a single interface connecting to one or more adjacent routers. In other situations, the capacity of a single transmission medium may be subdivided into multiple time slots. For example, a port may subdivide its bandwidth into 4 time slots, each with 1/4 of the bandwidth. The router could be configured to assign one or more of these time slots to a single interface. This 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 (PVCs), each corresponding to a different interface at the IP layer.

A *link* consists of two or more interfaces with the same network address [7]. For example, the prefix 10.1.2.116/30 consists of four IP addresses 10.1.2.116, 10.1.2.117, 10.1.2.118, and 10.1.2.119. Addresses 10.1.2.117 and 10.1.2.118 identify the two ends of a bidirectional, point-to-point link; an interface Serial1/0/0:1 with IP address 10.1.2.117 connects to a neighboring router that has an interface with IP address 10.1.2.118. The other two IP addresses, 10.1.2.116 and 10.1.2.119, represent the network and broadcast addresses, respectively. 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.

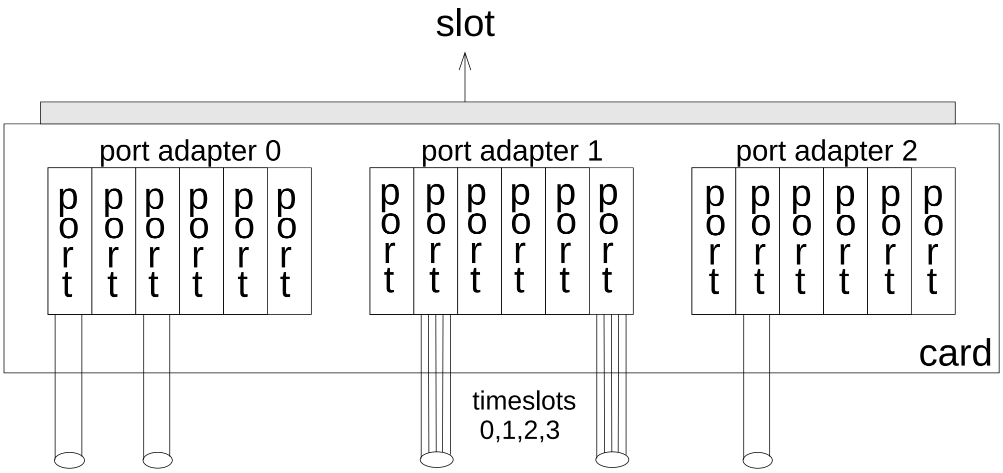

Figure 3: Card with three port adapters, each with multiple ports

2.3 Routing Protocols

Forwarding packets to their ultimate destinations depends on the complex interaction of various routing protocols. Static routes provide a simple way for a router to associate destination prefixes with particular outgoing interfaces. For example, a router could have a static route to interface Serial1/0/0:1 for prefix 172.12.4.0/24. Static routes enable an ISP to direct traffic to its customers without requiring the customers to participate in a routing protocol with the ISP. The customer's network can direct all outgoing traffic to the access link connecting to the ISP, and the ISP can configure static routes to direct inbound traffic to the access link. If a customer connects to the ISP in multiple locations, the ISP may have multiple static routes to associate the same prefix with multiple access interfaces at one or more routers. The ISP must ensure that other routers in the backbone and the rest of the Internet can reach the customer's destination prefix, as discussed below.

The AS also learns about destination prefixes via dynamic routing protocols, such as the Border Gateway Protocol (BGP). BGP is a distance-vector protocol that constructs paths by successively propagating reachability information [8, 9]. 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 a BGP session between a pair of routers. For example, a customer may have a BGP session with one of the ISP's routers to exchange routing information. BGP update messages travel over a TCP connection that provides ordered, reliable delivery of the update messages. An ISP typically has multiple BGP sessions with neighboring providers at different geographic locations. For each BGP session, the router 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 neighboring AS.

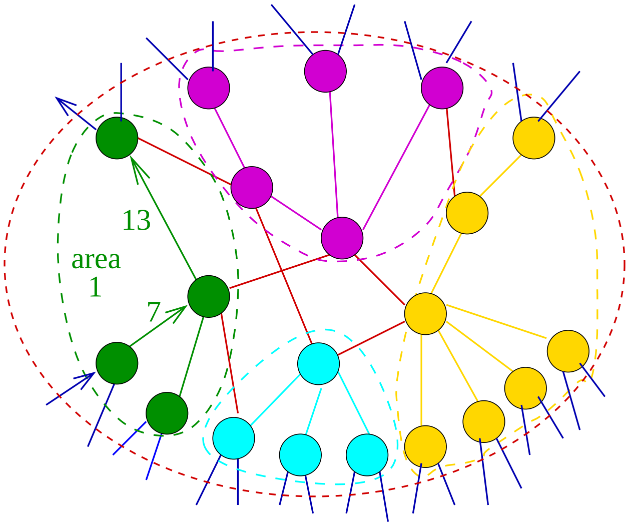

Figure 4: AS divided into four OSPF areas

In addition to communicating with customers and peers, an ISP may use BGP to distribute routing information inside the backbone. A pair of routers in the same AS can communicate via interior BGP (iBGP) sessions. For example, a router with a static route to prefix 172.12.4.0/24 can use iBGP to announce the route to the other routers in the ISP backbone. These routers may, in turn, propagate information about the route to neighboring ASes. The simplest way to convey routing information throughout the backbone is to have an iBGP session between each pair of routers (i.e., a full iBGP mesh). However, this approach would introduce considerable overhead in a large backbone network. Instead, a large AS may employ techniques such as route reflectors or confederations to distribute BGP advertisements in a hierarchical fashion.

The AS employs an intradomain routing protocol, such as OSPF or IS-IS, to select paths across the backbone [10]. Typically, customers and peers do not participate directly in these protocols. Under OSPF and IS-IS, each interface to a core link is assigned an integer weight, as shown by the numbers 7 and 13 in Figure 4. The routers exchange link weights and compute shortest paths, where path length is defined as the sum of the link weights; in the figure, the lower-left router reaches the upper-left router via a path of cost 20. Since these protocols use flooding to propagate link-state information, they do not scale to large networks. To reduce overhead, a large backbone may be divided into multiple areas. Figure 4 shows an example of a network that is divided into four OSPF areas, each represented by dotted lines. Each core link belongs to a single OSPF area; the red links between areas belong to the backbone area (area 0) that interconnects the other areas. 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 the forwarding table.

2.4 Packet and Route Filters

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 control lists* (ACLs) 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 172.12.4.0/24. 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 in detecting spoofed source IP addresses and protects the customer from receiving traffic from unwanted sources [11].

In addition, routers that participate in BGP may employ *route filters* to discard unwanted routing advertisements. A route advertisement consists of a destination prefix, a list of ASes on the 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. To prevent this problem, an ISP can filter routes from customers that include the AS number of other large service providers. An AS may also use route filters to balance the load across a collection of links to the same neighboring domain. 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

Routers rely on input about the configuration of the hardware, the partitioning of resources, the routing protocols, and which filters to apply. The commands applied to a router are captured in a configuration file. In this section, we present a brief review of the information available in router configuration files using Cisco IOS as an example. Other router vendors, such as Avici and Juniper, produce configuration files with similar information. Then, we identify the dependencies within and across files and discuss how to exploit these dependencies to check for configuration mistakes.

3.1 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 POS 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 5. 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) [12] 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 5 has four global settings—identifying the IOS version ("version xx.x"), disabling the finger daemon ("no service finger"), specifying the router's name ("hostname router1"), and enabling 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), which allows a IP prefix to have an arbitrary mask length [6]. 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 5 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-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 MoneyBags, and a "bandwidth" statement indicating the link capacity. The POS2/0 interface (slot 2, port adapter 0) refers to a core 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

```
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 MoneyBags
    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 5: Sample IOS router configuration file

(address 10.126.236.3 and a 32-bit mask) is associated with area 0 (the backbone area) and the POS2/0 interface (address 10.126.212.2 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). The "neighbor" commands are used 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. In addition, the “neighbor” commands are used to create labels, such as “intra-att-cluster,” that allow the operator to assign the same BGP attributes (such as the AS number) to multiple BGP sessions.

3.2 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 computer 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.

3.2.1 Domain-Independent Violations

Domain-independent violations include basic errors that do not require an understanding of the semantics of the configuration commands.

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 application of error checks. The specification of interface Serial1/0/0:1 cannot be fully understood without consulting 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.”

3.2.2 Domain-Dependent Violations

Domain-dependent violations relate to the internal consistency of the configuration of the router. Although the router resolves the violations, the default handling may result in different behavior than intended by the human operator.

Inconsistent definitions: The router configuration language permits inconsistent definitions. For example in the Cisco command-line interface, 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. In the example configuration of Figure 5, both statements are consistent for interface Serial1/0/0:1 since the channel-group 1 on controller 1/0/0 consists of one timeslot which on a T1 controller has a default bandwidth of 56 Kbits/sec and the bandwidth statement specifies a value of 56. Inconsistencies can have several negative consequences. First, inconsistent definition of interface capacity can result in incorrect SNMP statistics for link utilization. For example, if the interface capacity was mistakenly configured to 64 Kbits/sec, a traffic load of 50 Kbits/sec would result in a link utilization of 78% rather than the correct value of 89%. Second, the interface capacity may play 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 the link capacity (as defined via the bandwidth statement). 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 assigns an OSPF area, causing the link to 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.

Dependence on order: Many configuration options depend on the order of entries. For example, filter lists such as access-lists and route-filters depend on the order in which the entries are specified. For access-list 70 in Figure 5, the order of the entries is not important since the filters do not overlap. As another example, consider two access-list commands with “permit 10.1.2.0 0.0.0.255” followed by “deny 10.1.2.116 0.0.0.3”. The “deny” statement does not have any effect, since any packets matching this statement would have matched the previous “permit” statement. Flagging unnecessary access-list statements alerts the operator to inadvertent mistakes in configuring packet and route filters. For example, the operator may have intended the “deny” statement to precede the “permit” statement.

3.3 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 not have 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 core 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. In addition, 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. For example the two iBGP sessions should agree on the AS number of the autonomous system. Suppose an operator made a simple typographical mistake, such as replacing “intra-att-cluster” with “intra-att-bluster” in the iBGP session definition “neighbor 10.126.236.94 peer-group intra-att-cluster” of Figure 5. This would disable the iBGP session with the router with IP address 10.126.236.94 since the peer group intra-att-bluster is not defined.

Inconsistent references to remote nodes: For an eBGP session, one of the two end points resides outside of the ISP network, 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 item, they should have the same view of this item. 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.

3.4 Taking Advantage of Dependencies

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 routes. 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 across 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 network. 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 across files define a natural ordering for processing the configuration data to identify configuration errors.

Identifying mistakes requires an accurate snapshot of the configuration state of the network. The configuration files must be accurate in two key aspects. First, configuration data should be available for *all* of the routers in the IP backbone. Missing configuration files can lead to missing or incorrect error messages. For example, if the configuration file of the router in NY is missing, the link from interface POS2/0 will be incomplete, resulting in erroneous error messages. The tool may miss certain errors because only the information in the NY router configuration files reveal the fact that inconsistencies actually exist. 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 dependencies between parts of the configuration files 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, a tool for detecting configuration mistakes should note the 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. Any large AS should have a sufficiently large set of IP addresses available for assigning to the routers and interfaces.

4 Network Model

This section discusses practical issues that arise in developing a tool that checks for configuration mistakes in an operational network. Detecting errors requires comparing the configuration data against a network model. However, the complexity of IP networks and existing configuration languages hampers the development of a general network model. Rather than trying to capture all aspects of router configuration, we present a case study of a simple network model that is relevant for a variety of traffic engineering tasks. Then, we discuss how to extend the model as the network evolves and how to customize the model to accommodate local policies and conventions.

4.1 Scoping the Problem

Configuration languages, like Cisco IOS, have hundreds of commands that relate to a wide range of router products, network protocols, and interface technologies. Commercial tools [13] that check for configuration mistakes typically have to support the full range of configuration options. As a result, the development of these tools typically falls behind the deployed base of high-end router products. Yet, in practice, operational networks are remarkably homogeneous, in terms of the link-level technologies, router types, and configuration commands they employ. A tool for testing network configuration 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 actually used in the operational network. This offers a considerable reduction in complexity, and enables the tool to keep pace with the gradual introduction of new features and new router products.

Even in a relatively homogeneous environment, checking the correctness of every aspect of network configuration is very difficult. Configuration commands are complicated and often interact with one another. In addition, these commands relate to a diverse collection of network mechanisms and protocols. In practice, the task of configuring an operational network does not belong to a single person or group. Rather, different aspects of network configuration may be handled by different groups. Each group may want to perform certain checks on the router configuration and run other network management applications that rely on a particular model of the operational network. Focusing on a specific class of applications simplifies the network model and allows the tool to ignore many configuration commands (or certain interactions between

Object	Attributes
router	router name, {loopback IP address}
interface	interface name, *router*, {IP address}, capacity, up/down, OSPF weight
link	*IP prefix*, {*interface*}, core/edge, OSPF area
BGP end-point	*router*, remote IP address, remote AS, {*interface*}
static route	*IP prefix*, {*interface*}
access list	{*interface*}, packet/route, in/out, {IP prefix, permit/deny}

Table 1: Network model (with italics for object names and "{}" for sets)

commands). As part of checking for configuration mistakes, the tool can populate the network model to represent the current state of the network, which can serve as an input to other network management applications. Over time, the tool can evolve to include additional configuration information and support a broader set of applications.

4.2 Network Model for Intradomain Traffic Engineering

Our motivating application is intradomain traffic engineering—tuning the configuration of the routing protocols to the prevailing traffic in the backbone. This application depends on having an accurate view of the topology and routing configuration of the ISP backbone, along with detailed measurements of the traffic traveling through the network; the techniques for collecting the measurement data [4, 14] are not discussed in this paper. An ISP network does not connect directly to the hosts responsible for sending and receiving this traffic. Instead, the traffic enters and leaves the network at edge links that connect the ISP to other domains. As a result, traffic engineering in an ISP network requires a way to associate source and destination IP addresses with particular ingress and egress links. To support intradomain traffic engineering, our simple network model focuses on the physical components (e.g., routers and interfaces), physical and logical connectivity (e.g., links and BGP sessions), the intradomain routing protocol (e.g., OSPF), and information about end-host addresses (e.g., packet filters, route filters, and static routes).

Each *router* is identified by a unique name and one or more loopback IP addresses, as shown in Table 1. A router is associated with a collection of interfaces. Each *interface* is identified by a unique name and one or more IP addresses. The interface has a capacity and a status (either up or down). An interface that is part of a core link is associated with an OSPF weight. Each *link* is identified by an IP prefix and is associated with one or more interfaces in the ISP network. The link is classified as either a core or edge link; a core link is associated with an OSPF area. Each *BGP end-point* is identified by the associated router and the IP address of the remote end-point. The remote end-point belongs to some AS. If the two end-points belong to the same AS, then the session is an iBGP session; otherwise, the session is an eBGP session. The router forwards BGP advertisements toward the remote end-point via one or more interfaces.

Each *static route* is identified by an IP prefix and is associated with one or more interfaces. Each *access list* consists of an ordered list of IP prefixes, where each prefix is either permitted or denied (to accept or reject traffic, respectively). The access list is used either as a packet filter or a route filter in either the inbound or outbound direction for one or more interfaces. In practice, BGP sessions may have quite diverse and complex import and export policies. We focus specifically on route filters that indicate which destination prefixes may be advertised on a particular BGP session. In addition, our model focuses on a single AS that uses static routes and BGP to connect to neighboring domains, iBGP to distribute reachability information inside the backbone, and OSPF to route across the backbone. In constructing an abstract model, we have modeled interfaces without capturing the many different layer-two technologies that exist in an operational network. Hiding these low-level details results in a much simpler model for traffic engineering. The model also omits numerous parameters related to the packet scheduling and buffer management policies for each interface.

4.3 Extending the Model

Focusing on a single application results in a simpler network model and allows the configuration tool to consider a smaller set of configuration options. However, new router features and configuration commands arise as a network continues to evolve. 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. In addition, ignoring a new command may result in an incorrect model of the network, which would affect other network management applications. Yet, a tool for checking for configuration mistakes should not preclude the use of new commands. A relatively simple approach is to maintain a list of the configuration commands supported by the tool. 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. This approach detects 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 new commands arise, the tool can be extended to support the changes to the model and to incorporate new error checks. In some cases, a new command can be ignored since it does not affect any aspects of the network model. In other cases, the new command may require extending the software to incorporate a new way of defining an existing parameter. Finally, in some cases, a new command may require a change to the underlying model. Possible changes in the network include:

- **New parameter settings:** Day-to-day operation of the network results in changes in the values of parameters of existing objects. For example, the operator may tune an interface's OSPF weight in response to network congestion. This does not require any changes to the network model or the tool for checking the configuration.

- **New instances of objects:** Installing additional hardware or software component such as interfaces or routers would result in additional instances of the various kinds of objects. This does not require any changes to the network model or the tool for checking the configuration.
- **New ways of deriving existing parameters:** A network operator may use a different configuration command, or set of commands, to specify an existing parameter in the model. This requires a change to the software for generating error checks and populating the network model, without requiring modifications to the underlying model.
- **New parameters:** Extensions to the network model only arise when the operators employ a new feature that relates to an object in the network model. In many cases, the new feature may not be relevant to the higher-level applications of the model. In other cases, the tool and the model can be extended to incorporate an additional attribute (e.g., a buffer threshold for Random Early Detection for each interface).
- **New objects:** The AS occasionally undergoes significant architectural changes, such as the introduction of a new routing protocol (e.g., MPLS or multicast). New protocols, if relevant for traffic engineering, require the addition of new objects to the model. 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.

4.4 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. Sometimes it can also be advantageous to include some of this derived information in a network model. These extensions could draw on information from a variety of sources:

- **Router hostnames:** 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 [15], based on the remote AS of the BGP session. The classification may have implications on the desired BGP import and export policies [16, 17]. 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 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 can be used to define additional error and consistency checks. These conventions include default global settings, access-control lists, and BGP import/export policies that should appear in every router configuration file.

4.5 Higher-Level Checks

An operational network may have high-level policies that result in additional error checks. For example, certain checks could be customized based on the type of the router or link. A BGP session with a customer may employ different import and export policies than a session with a peer [17]. These BGP policies can be codified in separate input files, and checked against each BGP session in the router configuration files. Similarly, conventions for iBGP configuration can be expressed in a separate input file. For example, an AS may use route reflectors to reduce the overhead of distributing BGP information inside the core. An AS may have policies for configuring the route reflectors (e.g., in a full mesh) and the route-reflector clients (e.g., every client communicates with at least two route reflectors). In fact, the full desired route reflector hierarchy may be specified in an external file and checked against the actual configuration. This approach has the side benefit of encouraging network operators to keep the specification of the intended architecture up-to-date.

As part of checking for configuration mistakes, the tool creates a snapshot of the current state of the backbone by populating the network model. Incorporating this snapshot into other tools facilitates a wider range of error checks. In particular, an accurate model of intradomain routing [3] enables verification of assertions about the physical and logical topology of the network. For example, an operator may want to verify that every layer-three link resides on at least one path. In an OSPF network, this can be tested by computing the shortest path(s) between each pair of routers based on the OSPF weights and area assignments. Similarly, an operator could verify that a BGP route reflector is close (in the OSPF sense) to each of its route-reflector clients. Going one step further, the intradomain routing model can be combined with information about customer traffic subscriptions for guaranteed services, such as IP telephony. By applying the routing model to the current network configuration, the operator could ensure that each link has sufficient bandwidth resources to satisfy the service-level agreements (SLAs) for the high-priority traffic. Applying these higher-level checks can help network operators handle the growing challenge of managing a large IP backbone network.

5 The Tool

In collaboration with AT&T IP Services, we have developed a tool that populates the network model and detects possible configuration errors for the AT&T 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 large service providers. The tool is a Perl script that parses configuration files in IOS format. Running the tool on the configuration files for the operational network requires a few minutes on a Sun Enterprise server. In this section, we describe its design and operation and present sample error messages that illustrate what kind of configuration problems 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, the tool reads through the files to create a table that stores every line, as shown in Figure 6. Then, for each router, it 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 5 has three interface entries. For each router and each section, the tool stores a list of entries. For each entry, it 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. The value associated with each attribute is stored in a hash table. For example, “router1|POS2/0|ip address” would map to “10.126.10.2 255.255.255.252”.

The tool generates an error message upon encountering an unfamiliar global setting or section name. After identifying section boundaries, it parses and checks the global settings, and checks if all desired global variables have been specified. As part of processing each global setting, it assigns the associated parameter in the network model. This process repeats across all of the routers. Next, each of the sections is parsed, one at a time, across all of the routers. The processing order derives from the dependencies identified in

```
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 6: Processing order of the tool

Section 3—controllers, access lists, interfaces, other 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 the tool 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. The tool flags an error if the interface entry refers to an access-control list that does not exist. 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, the tool has already parsed all of the interface sections across all of the routers. As such, it already knows which interfaces have IP addresses in this prefix. If two or more interfaces fall within this prefix, the tool can infer the existence of a backbone link in a particular OSPF area. Otherwise, it can flag an error. If another router's configuration file has an OSPF "network" statement that applies to the same IP prefix, the tool can check that the two statements assign the same OSPF area.

In the process of parsing a section, the tool 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 the interface entries within the interface section are parsed. For each interface entry, the tool considers the IP prefix associated with the interface's IP address. For the first occurrence of a prefix, it creates a new link object. The link object is associated with the IP prefix and with the IP address of the particular interface. If the tool 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, the tool performs a longest-prefix match to associate each statement with the appropriate keyword. Following our philosophy of customization, each section may have one or more input files that specify additional information, policies, or default values.

After processing all of the sections, the tool 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, the script 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, the tool keeps track of which statements have been applied one or more times. As the final stage of processing, it generates warnings for statements that were never applied.

5.2 Error-Checking Examples

The tool's error checks generate a variety of error messages, as shown by the sample output in Figure 7. The first example illustrates how it flags unknown entries. In the second example, the tool flags the fact that BGP community 1000 is referenced but not defined. Both of these errors relate to mistakes within a single router configuration file. The third, fourth, and fifth examples illustrate OSPF problems—a backbone

```
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 7: Examples of error messages from the tool

```
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: customer needs in and out ACL; (customer MoneyBags)
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 8: Examples of error messages from the tool for policy violations

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 from 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.

The tool performs a wide variety of consistency checks; Figure 8 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 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.

```
node|11|Backbone router|Los Angeles, CA|router1|10.126.236.3|-118.2|34.0
node|12|Backbone router|New York, NY|router21|10.126.236.94|-74.0|41.5
node|13|Peer router|PEER1|PEER1||-10.0|11.0
link|3|Core link|11|12|10.126.212.2|10.126.212.3|POS2/0|POS1/1|
    2.4Gbits|2.4Gbits|5|5|1|to router in NY|to router in LA
link|4|Access link|11|3127|10.1.2.117||Serial1/0/0:1||56Kbits|56Kbits|
    |||link to customer MoneyBags|
```

Figure 9: Instantiation of a network model generated by the tool

5.3 Applications of the Network Model in NetScope

In addition to generating error reports, the tool can export parts of an extended network model. For example, the script produces a concise representation of the network topology and OSPF configuration, as shown by the sample output in Figure 9. The first two fields identify the object type (e.g., node or link) and an identifier (integer). The third field identifies the type (e.g., backbone router or access link). For nodes the fourth, fifth, and sixth fields specify the city, router name, and loopback IP address, and the last two fields identify the geographic location of the city in longitude and latitude. The router names and loopback addresses are derived from the configuration files; the other information is derived in the customization process from the router name. For links, the fourth through eleventh fields specify the adjacent routers via their identifiers, the interface IP addresses, the card names, and their capacities. For example, the interface IP address for core link 3 is 10.126.212.2 in one direction and 10.126.212.1 in the other. The twelfth through fourteenth field consist of the OSPF weight (for each direction) and the OSPF area. Edge links have empty fields regarding the remote end point and the OSPF configuration. The remaining fields contain the up/down status and the description fields of the two interfaces of the link. Other output types are defined for other parts of the network model.

This instantiation of the network model is one of the key inputs to the NetScope toolkit for IP traffic engineering [3]. NetScope combines the configuration data with traffic measurements to support a variety of analysis and visualization applications:

- **Visualization of traffic statistics:** The configuration data provides NetScope with a network-wide view of the backbone topology, including the geographical location of each of the routers. NetScope displays the topology on a map with the links and routers colored and sized based on traffic measurement data. SNMP polling provides a periodic view of router and link utilization statistics, identified by router name and SNMP index numbers. These statistics can be associated with the appropriate routers and interfaces in the network model. Other traffic statistics, such as active measurements of delay, throughput, and loss [18], can be associated with the underlying topology in a similar way.

- **Visualizing and optimizing routes:** NetScope allows the network operator to identify which path a packet follows through the network from an ingress link to an egress link. The configuration data provides a snapshot of the topology and the intradomain routing configuration, including the OSPF weight for each core interface and the OSPF area for each core link. NetScope uses this information to compute the shortest path(s) between each ingress and egress point. The operator can experiment with changes in the topology and the weights to determine how the paths would change.

- **Computing and visualizing traffic demands:** Understanding the flow of traffic through the backbone depends on detailed measurements at various points in the network. The measurement data may be available from packet monitors or from flow-level measurements at the routers. Constructing a network-wide view of the traffic requires associating the measurement data with the appropriate interfaces and routers in the backbone [4]. This association is provided by the configuration data. Then, NetScope identifies how the traffic flows through the backbone, based on the network topology and routing configuration. This enables the network operator to identify the traffic responsible for congested links. In addition, the operator can invoke an optimization module to tune the OSPF weights to the prevailing traffic [5].

NetScope constructs a network-wide view of the backbone by combining the topology and configuration data with traffic measurements and a model of intradomain routing. NetScope models and displays the network-wide implications of local changes in topology, configuration, and traffic. This provides a powerful and extensible platform for "what-if" traffic-engineering investigations in a simulated environment. This would not be possible without an accurate, abstract representation of the network topology and configuration.

6 Conclusions

Parsing and analyzing routing configuration files is an effective way to check for configuration mistakes and to provide input data for network management tools. Our methodology capitalizes on the dependencies within and across configuration files to derive the network model and identify configuration errors. We discussed the operational requirements for collecting configuration data, evolving the network model, and customizing the error checks. The paper presented a case study of a network model for intradomain traffic engineering and showed how to check the actual configuration of the network with regards to its consistency. Our approach has been realized in a prototype tool that has been applied in the AT&T IP Backbone. This work is part of a larger effort to develop a suite of tools to help network operators manage the growing complexity of large IP networks.

Acknowledgments

We are indebted to Jay Borkenhagen for his depth of understanding of router configuration and his willingness to share his expertise with us. Thanks also to Han Nguyen, Mike Langdon, Kevin McClarren 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 the prototype configuration-checking tool. 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. Thanks also to the anonymous referees for their detailed feedback and constructive suggestions.

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] AT&T IP Backbone, `http://www.ipservices.att.com/backbone`.

[3] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford, “NetScope: Traffic engineering for IP networks,” *IEEE Network Magazine*, pp. 11–19, March 2000.

[4] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, “Deriving traffic demands for operational IP networks: Methodology and experience,” *IEEE/ACM Trans. Networking*, vol. 9, June 2001.

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

[6] 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.

[7] F. Baker, “Requirements for IP Version 4 Routers.” Request for Comments 1812, June 1995.

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

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

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

[11] 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.

[12] Cisco, “Cisco IOS Configuration Fundamentals,” 1998. Documentation from the Cisco IOS reference Library.

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

[14] R. Caceres *et al.*, “Measurement and analysis of IP network usage and behavior,” *IEEE Communication Magazine*, May 2000.

[15] G. Huston, “Interconnection, peering, and settlements,” in *Proc. INET*, June 1999.

[16] C. Alaettinoglu, “Scalable router configuration for the Internet,” in *Proc. IEEE IC3N*, October 1996.

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

[18] AT&T IP Network Performance, `http://ipnetwork.bgtmo.ip.att.net`.