Proactive Techniques for Correct and Predictable Internet Routing

Nicholas Greer Feamster

Proactive Techniques for Correct and Predictable Internet Routing

by

Nicholas Greer Feamster

M.Eng., Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001
S.B., Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2000

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

Massachusetts Institute of Technology

February 2006

Author .....
Department of Electrical Engineering and Computer Science
September 30, 2005

Certified by .....
Hari Balakrishnan
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Proactive Techniques for Correct and Predictable Internet Routing

by

Nicholas Greer Feamster

Submitted to the Department of Electrical Engineering and Computer Science
on September 30, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

The Internet is composed of thousands of autonomous, competing networks that exchange reachability information using an interdomain routing protocol. Network operators must continually reconfigure the routing protocols to realize various economic and performance goals. Unfortunately, there is no systematic way to predict how the configuration will affect the behavior of the routing protocol or to determine whether the routing protocol will operate correctly at all. This dissertation develops techniques to reason about the dynamic behavior of Internet routing, based on *static* analysis of the router configurations, before the protocol ever runs on a live network.

Interdomain routing offers each independent network tremendous flexibility in configuring the routing protocols to accomplish various economic and performance tasks. Routing configurations are complex, and writing them is similar to writing a distributed program; the (unavoidable) consequence of configuration complexity is the potential for incorrect and unpredictable behavior. These mistakes and unintended interactions lead to routing faults, which disrupt end-to-end connectivity. Network operators writing configurations make mistakes; they may also specify policies that interact in unexpected ways with policies in other networks. To avoid disrupting network connectivity and degrading performance, operators would benefit from being able to determine the effects of configuration changes *before deploying them on a live network*; unfortunately, the status quo provides them no opportunity to do so. This dissertation develops the techniques to achieve this goal of proactively ensuring correct and predictable Internet routing.

The first challenge in guaranteeing correct and predictable behavior from a routing protocol is defining a specification for correct behavior. We identify three important aspects of correctness—path visibility, route validity, and safety—and develop *proactive* techniques for guaranteeing that these properties hold. Path visibility states that the protocol disseminates information about paths in the topology; route validity says that this information actually corresponds to those paths; safety says that the protocol ultimately converges to a stable outcome, implying that routing updates actually correspond to topological changes.

Armed with this correctness specification, we tackle the second challenge: analyzing routing protocol configurations that may be distributed across hundreds of routers. We develop techniques to check whether a routing protocol satisfies the correctness specification *within a single independently operated network*. We find that much of the specification can be checked with static configuration analysis alone. We present examples of real-world routing faults and propose a systematic framework to classify, detect, correct, and prevent them. We describe the design and implementation of *rcc* ("router configuration checker"), a tool that uses static configuration analysis to enable network operators to debug configurations before deploying them in an operational network. We have used *rcc* to detect faults

in 17 different networks, including several nationwide Internet service providers (ISPs). To date, *rcc* has been downloaded by over seventy network operators.

A critical aspect of guaranteeing correct and predictable Internet routing is ensuring that the interactions of the configurations *across multiple networks* do not violate the correctness specification. Guaranteeing safety is challenging because each network sets its policies independently, and these policies may conflict. Using a formal model of today's Internet routing protocol, we derive conditions to guarantee that unintended policy interactions will never cause the routing protocol to oscillate.

This dissertation also takes steps to make Internet routing more predictable. We present algorithms that help network operators predict how a set of distributed router configurations within a single network will affect the flow of traffic through that network. We describe a tool based on these algorithms that exploits the unique characteristics of routing data to reduce computational overhead. Using data from a large ISP, we show that this tool correctly computes BGP routing decisions and has a running time that is acceptable for many tasks, such as traffic engineering and capacity planning.

Thesis Supervisor: Hari Balakrishnan
Title: Associate Professor of Computer Science and Engineering

To Mom, Dad, and Gram

The crowd makes the ballgame.
- Ty Cobb

Acknowledgments

My advisor, Hari Balakrishnan, once said that one secret to success is to surround yourself with people who are smarter than you are. This task must be incredibly difficult for him. I am extremely fortunate to have crossed paths with Hari and to have had the opportunity to work with him so closely for many years. Hari has guided me with amazing expertise. His sharp intellect and ability to quickly grasp the important and interesting aspects of a problem are matched only by his fantastic taste in research problems. Hari opened (and closed) all of the right doors for me; he gave me the opportunity to succeed at every corner, and he continually encouraged me to consider the broader impact of my research while allowing me the freedom to pursue problems I found most interesting.

Jennifer Rexford has been another fantastic mentor. She shares my insatiable passion for details and practical problems. Her patience, willingness to consider new ideas, and ability to explain complicated concepts make her a fun person to work with and should serve as a model for every advisor. Jennifer introduced me to Internet routing, and many of her papers provided inspiration for several chapters in this dissertation. She has also been a teacher, a useful sounding board, and a continual source of sound advice.

I owe a great debt to the other two members of my committee, David Clark and Frans Kaashoek. David read this dissertation with vigilance, and my discussions with him, both during the writing process and throughout my graduate career, helped me strengthen many of the conceptual aspects of this dissertation. Frans provided invaluable guidance in clarifying the presentation of many of the ideas in this dissertation. His enthusiasm for both the work in this dissertation and other projects we worked on together have kept me optimistic and excited about my research. When I'd think some of my projects were fruitless or boring, Frans always provided the boost I needed. On the squash court, he continually reminds me about the virtues of humility.

I am lucky to have Ramesh Johari and David Andersen as colleagues. I've had great fun working with both of them and learning from their different research styles. Ramesh's penchant for precision—both in math and in English—and his tenacity in problem solving make him a pleasure to work with. I have great respect for him as a researcher, a mentor, and a friend. Dave's energy, fun-loving nature, and willingness to put up with my continual questions while always questioning my own research was one of the most valuable assets throughout my graduate career. Dave was my officemate and housemate, but he is

also a great collaborator and a terrific friend. Some of my best times in graduate school were spent cycling through Massachusetts and talking research with Dave.

One of the great things about MIT is that you don't have to try very hard to surround yourself with people who are smarter than you are. My research and career benefited from the influence of many faculty members, including Rod Brooks, John Guttag, David Karger, Daniel Jackson, Dina Katabi, Barbara Liskov, Sam Madden, Robert Morris, and Ron Rivest. It has been a joy to share my graduate career with Magdalena Balazinska; she is a tremendous colleague and friend. Thanks to her, I remembered to eat lunch most of the time. I have learned a lot from exchanging ideas with Alex Snoeren, who was also a tremendous support during my job search. Both Michel Goraczko and Dorothy Curtis were helpful and patient with my technical questions and emergencies. I have also had fantastic officemates. Jaeyeon Jung and Mythili Vutukuru have been fun to work with; I hope we continue working together. Michael Walfish's clarity of expression—not to mention his enthusiasm for a good argument—made G982 an exciting place to work. Discussions with Jaeyeon, Mythili, and Mike have helped me refine my research ideas. Sheila Marian made my life around the lab smooth and pleasant; I will miss discussing the latest Red Sox gossip with her. I also thank the members of the NMS and PDOS groups for listening to my many thoughts and practice talks.

I am fortunate to have wonderful colleagues outside of MIT. In particular, I would like to thank Randy Bush, Tim Griffin, Albert Greenberg, Jacobus van der Merwe, Aman Shaikh, Lixin Gao, kc claffy, Avi Freedman, Morley Mao, Olivier Bonaventure, Richard Mortier, and the network operators who have used *rcc*, for taking a deep interest in my work and providing feedback and support. I thank Joan Feigenbaum and Anthony Joseph for their advice and insights during my job search. I've had helpful discussions (and great times) with Aditya Akella, Anukool Lakhina, Ratul Mahajan, Joel Sommers, Lakshmi Subramanian, and Renata Teixeira. Susie Wee and John Apostolopoulos introduced me to the rewards of research and have offered useful professional advice.

Mike Freedman, Sean Montgomery, and Ajay Kulkarni have continually supported me for nearly ten years. Anukool Lakhina has been an indispensable friend during the writing of this dissertation. Greg Harfst, Claire Monteleoni, Jamie Fine, Jennifer Bowen, Jonathan Hall, Jeremy Stribling, Kelly Carleton, Carson Reynolds, and Steven Richman have been companions in both celebration and commiseration. Dave Andersen, Thomer Gil, Alex Yip, Sam Madden, Dye-Zone Chen, and the members of the MIT Cycling Club were all great cycling partners. Sanmay Das, Tony Ezzat, and José Rafael Galvanis gave me many memories on the squash courts, and Doug DeCouto made swimming less dull.

The research in this dissertation was funded through an NSF Graduate Research Fellowship, the NSF under Cooperative Agreement ANI-0225660, the Defense Advanced Research Projects Agency (DARPA) and the Space and Naval Warfare Systems Center, San Diego, under contract N66001-00-1-8933, and a Cisco URP grant. The writing of this dissertation was fueled with sustenance from the 1369 Coffeehouse and Mariposa Café in Cambridge, MA; and the Big Cup, the Saurin Park Café, and Grounded in New York City.

My parents, Carolyn and Scott Feamster, have always encouraged me to push the frontier of knowledge. Both my parents and my late grandmother, Elizabeth Huey, have been a constant source of love and inspiration, without which this accomplishment would have been impossible. In return, I dedicate this dissertation to them. My success is also theirs.

Bibliographic Notes

An early version of some material from Chapter 3 appears in a paper co-authored with Hari Balakrishnan [29]. Material from Chapter 4 appears in a paper co-authored with Hari Balakrishnan [30]. Material from Chapter 5 appears in papers co-authored with Jennifer Rexford and Jared Winick [36, 37]. Material from Chapter 6 appears in a paper co-authored with Ramesh Johari and Hari Balakrishnan [33]. The original design for the Routing Control Platform, which is briefly discussed in Chapter 7, appears in a paper co-authored with Hari Balakrishnan, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe [31]. Ramesh Johari's dissertation provided formatting and typography inspiration.

Contents

List of Figures

List of Tables

If you don't know where you're going, you might not get there.
- Yogi Berra

CHAPTER 1
Introduction

The past fifteen years have seen a migration from the government-operated NSFNet to an agglomeration of commercial networks that communicate with one another to constitute what we commonly refer to as "the Internet". This data network requires the cooperation of tens of thousands of independently operated networks that are nonetheless competing with one another for each other's customers. In the midst of this changeable, federated landscape, we as users expect to be able to reliably communicate with other users of the network at any time.

Reliable communication between nodes in a network fundamentally depends on *routing*, the process by which some participant discovers paths to other network destinations. In the same way that a human might use routing to plan a course (*e.g.*, using the "driving directions" feature of an online mapping service to discover a path from Boston to New York), computers on the Internet rely on routing to discover paths between each other. There are many reasons why traffic on the Internet may not reach its intended destination: parts of the network infrastructure, network systems, and end hosts can all fail, for example. Even if all infrastructure and services operate correctly, though, two endpoints cannot communicate if they cannot discover a path between them.

Today's Internet routing infrastructure is unacceptably fragile. Among its shortcomings, it converges slowly [78] (and sometimes not at all [56]); it is often misconfigured [85]; it is hard to control and predict [32]; and it has weak security properties [93]. This fragility causes communication on the Internet to be unreliable and unpredictable. A major contributing factor to this fragility is Internet routing's configurability. Internet routing configuration enables competing networks both to implement policies reflecting complex business arrangements and to tune routing protocols to maintain good performance under highly dynamic conditions. The behavior of Internet routing is determined almost entirely by the set of configurations distributed across the routers in the network. In this sense, it is rather accurate to think of Internet routing as a massive distributed computation, and the routing configuration as a complex, distributed program written in a variety of low-level languages running on a heterogeneous set of platforms. This dissertation develops techniques towards making Internet routing more correct and predictable; much of the dissertation focuses on how to make configuration less of a harbinger of incorrect and

unpredictable behavior.

Given that there are so many ways for Internet routing to go wrong, guaranteeing correct and predictable behavior is a daunting task. Each new problem seems to merit a point solution that adds complexity to the routing protocol, making the infrastructure more complex, unpredictable, and unwieldy. Worse yet, network operators, protocol designers, and researchers have adopted a *reactive* approach to reasoning about Internet routing. The state-of-the-art for configuring Internet routing typically involves logging configuration changes and rolling back to a previous version when a problem arises. The lack of a formal reasoning framework means that configuring routers is time-consuming, ad hoc, and error-prone, and it is becoming more so as with the unceasing addition of new point fixes and "features".

This trend is unsettling, especially as the Internet matures and increasingly becomes a mission-critical part of our communication infrastructure. Rather than proposing point solutions to the myriad problems in Internet routing, this dissertation takes the opposite approach: we work top-down, first defining a high-level correctness specification for Internet routing and subsequently developing techniques to ensure that the routing protocol satisfies this specification. Using the correctness specification as a guide, this dissertation develops techniques that improve the correctness and predictability of the Internet routing system. We focus on this problem in the context of non-malicious network operations. We develop techniques that help operators and protocol designers reason about the behavior of today's Internet routing system. Further, we propose architectural and protocol changes that make the routing protocol itself less likely to behave incorrectly. This dissertation proposes two such tools based on *proactive* analysis of static routing configurations. One of these tools, called *rcc* ("router configuration checker"), uses the correctness specification and its constraints to derive a set of constraints that it checks directly against the routing configuration. The second tool, which we call a *routing sandbox*, allows a network operator to determine how traffic will flow through the network and quickly evaluate the effects of configuration changes on the flow of traffic.

While the techniques we present have proved effective for improving the correctness and predictability of today's Internet routing system, we firmly believe that the design of the routing infrastructure should address correctness and predictability as first-order concerns. The tools and techniques presented in this dissertation would likely have been far easier had the routing infrastructure been designed with these goals in mind in the first place. Consequently, this dissertation also proposes several architectural and protocol changes towards improving the robustness and manageability of the system. In general, we believe that the correctness specification in this dissertation provides a sound framework for considering such design changes.

We begin with a high-level overview of Internet routing to provide some context. In Section 1.2, we discuss how routing configuration can be used to control the behavior of the routing protocol; we also describe how mistakes and unintended interactions in routing configuration can induce catastrophic routing failures. After discussing the challenges in guaranteeing correct and predictable Internet routing (Section 1.3), we explain how *proactive* techniques for analyzing routing configuration can guarantee correctness and improve predictability of Internet routing (Section 1.4), summarize the major contributions of this dissertation (Section 1.5) and offer some take-away lessons (Section 1.6) that

offer insights that we hope will prove useful when thinking about further improvements to Internet routing. Section 1.7 concludes the chapter with a guide for reading the rest of this dissertation.

1.1 Internet Routing Overview

Although it is common to think of "the Internet" as a single, monolithic network, it is actually composed of tens of thousands of independently operated networks, commonly called *Autonomous Systems* (ASes). Figure 1-1 shows an example of how traffic from a cable modem user may traverse multiple ASes en route to machines at MIT. Internet traffic is forwarded from source to destination through a sequence of *routers* in one or more ASes.

Each one of these ASes typically has independent (and often conflicting) economic and performance goals, yet these ASes must cooperate by exchanging routing information to achieve global connectivity (*e.g.*, to allow a home user who buys service from his cable modem provider to communicate with hosts that purchase connectivity from other ASes). The current routing protocol on the Internet is the Border Gateway Protocol (BGP) [118]. As shown in Figure 1-2, ASes achieve global reachability by establishing BGP sessions between neighboring "border" routers. Each AS has may have anywhere from a single router to hundreds of routers. Each of these routers maintains a *routing table*, which contains one or more routes for each destination. Each router selects a single best route to each destination. Routing on the Internet is *destination-based*; that is, a router selects the next hop (*i.e.*, router) for which to forward traffic solely based on the destination IP address of each packet. The destination for a route is represented in terms of an IP *prefix*, which specifies a group of IP addresses that share a common number of bits.

Figure 1-3 shows example routing table entries for the set of destinations represented by the IP prefix `18.0.0.0/8`, which represents the 2^{24} IP addresses that share the first 24 bits, `18.*`. Although each router may learn multiple candidate routes to a prefix, (*i.e.*, the routing table shown in Figure 1-3 has two possible routes for the same destination), each router ultimately selects a *single* best route for each prefix (in the routing table, the route that the router ultimately selects is indicated by the ">". The *next hop* attribute is the IP address that the router must forward traffic towards to send traffic along this route. The router may learn how to reach this next hop IP address in one of several ways: a "static" route may be hardcoded, the router might learn a route via a routing protocol that

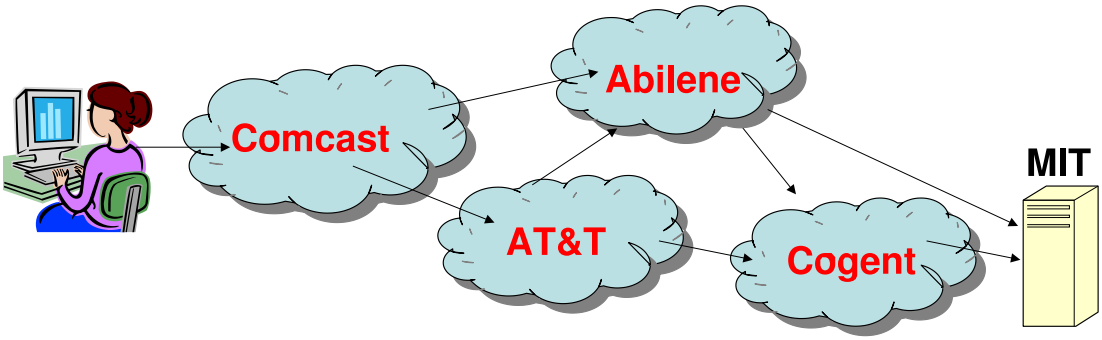


Figure 1-1: A typical Internet path may traverse multiple "Autonomous Systems".

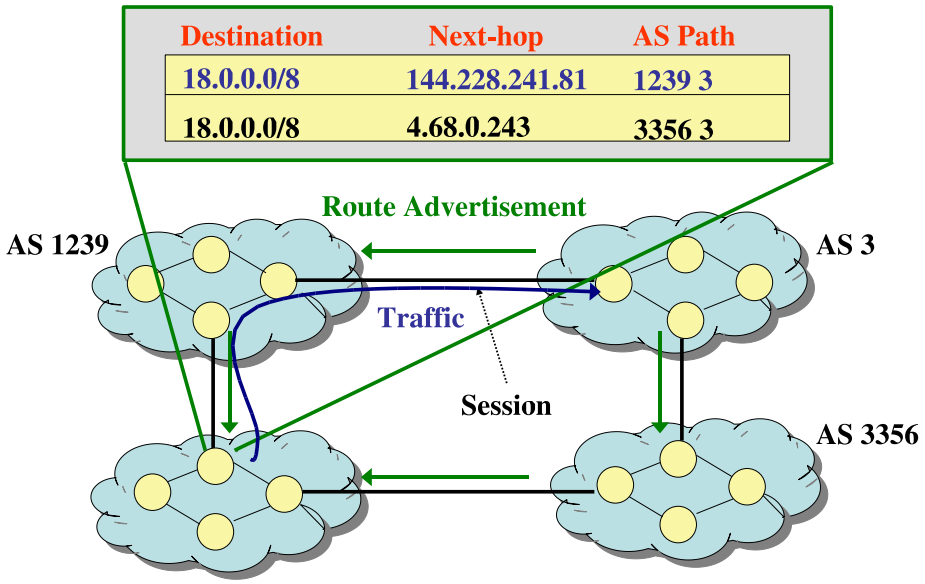


Figure 1-2: ASes exchange routing information over BGP *sessions* between routers. A router may learn multiple routes to the destination but ultimately selects a *single* best route. Traffic flows in the opposite direction of route advertisements.

```
   Network          Next Hop            Path
*> 18.0.0.0/8       144.228.241.81      1239 3
*  18.0.0.0/8       12.0.1.63           7018 3356 3
```

Figure 1-3: BGP routing table entry for prefix 18.0.0.0/8 as it might appear on a router. Each BGP route has attributes in addition to the next hop IP address and AS path, which we will discuss later.

is run inside the AS (*e.g.*, OSPF), and so forth. The *path* attribute refers to the AS path: the sequence of ASes that the route advertisement traversed en route to this router. A BGP route has several additional route attributes that are not pictured; we will describe these additional attributes in more detail in Section 2.2.1.

Internet routing requires neighboring ASes to exchange routing information, but it also requires each of these ASes to run an internal routing protocol ("Interior Gateway Protocol", or *IGP*) to establish reachability information about destinations within the same AS. For example, in the routing table excerpt shown in Figure 1-3, the router knows that to forward traffic to any destination in `18.*`, it must send the traffic towards the next hop `144.228.241.81`. For "border" routers, this next hop is typically the address of an interface of the router in a neighboring autonomous system and is an immediate next hop. Routers within an AS, however, must use the IGP to discover the outgoing interface over which to send traffic towards this next hop, which may be multiple hops away. This dissertation focuses on BGP but does not address the operation of internal routing protocols. Other work provides more detailed treatment of IGPs [41, 123, 124].

1.2 Configuration: The Achilles' Heel of Internet Routing

Analyzing the behavior of any routing protocol is inherently difficult, but Internet routing presents a unique set of challenges because it must be highly configurable to support the complex economic and performance goals that each independently operated network is attempting to satisfy. The standards document for BGP [118] specifies the message format but intentionally leaves unspecified many details, including the criteria for selecting the route to a destination given multiple alternatives. Instead, these details are left to its *configuration*. In this section, we explain how configuration affects routing protocol behavior; we then present some examples that demonstrate how configuration mistakes can induce catastrophic routing failures.

1.2.1 How Configuration Affects Routing Protocol Behavior

Internet routing configuration provides a network operator remarkable latitude in controlling how the protocol behaves. In particular, Internet routing configuration allows an operator to control the routing protocol in the following ways:

- **Which ASes to carry traffic for.** Depending on the business relationships that an AS has established with other ASes, it may arrange to carry traffic to a destination for some of those ASes but not others [47]. Routing configuration controls which routes an AS advertises to each of its neighbors, implicitly controlling which neighbors can send traffic over the AS en route to a destination.

- **How traffic enters and leaves the AS.** An AS typically has multiple links over which it can send traffic to a destination: some of these links are internal (*i.e.*, they are between two routers in the same AS) and others are external (*i.e.*, they are between routers in neighboring ASes). Changes in traffic demands may cause any of these links to become congested. In response, a network operator may change the routing configuration to shift portions of the traffic load to a different set of links [36].

- **How routers within an AS learn routes to external destinations.** Each independently operated network comprises tens to hundreds of routers. Ultimately, every router in the AS must learn the routes to external destinations, but, initially, only the AS's "border" routers learn these routes. Routing configuration controls how the routes propagate from an AS's border routers to the rest of the routers in the AS.

Changing traffic demands and business relationships, planned maintenance, and equipment failures may all change traffic patterns through the AS, but routing protocols do not automatically adapt to these changing conditions. As a result, network operators must constantly tune the behavior of the routing protocols in their ASes to control how traffic flows through them.

1.2.2 Problem: Configuration Mistakes Cause Routing Failures

The cost of Internet routing's configurability is a high degree of complexity. The unfortunate consequence of this complexity is that the potential for incorrect behavior is enormous.

The consequences of incorrect behavior can be staggering. The past few years alone have seen several high-profile examples of Internet routing configuration problems:

- In 1997, a small ISP in Florida configured its routing in a way that caused all of the Internet's traffic to be routed through it [121].
- In 2001, Microsoft brought down its Web servers with a routing misconfiguration; it took nearly a day to diagnose the problem [90].
- In 2002, Worldcom took down more than 20% of its nationwide "backbone" in the United States with a routing configuration problem [143].
- In 2004, Level3 incurred a widespread outage due to a router configuration problem [81].
- In 2005, an ISP in Bolivia caused a major outage when it announced the IP prefix for AT&T's United States backbone network (*i.e.*, `12.0.0.0/8`). [82].

Major news outlets report only the most catastrophic routing failures; in fact, mistakes in routing configuration are continually causing reasonably serious routing failures. In the introduction to Chapter 4, we present the results of our informal study of the mailing list archives of the North American Network Operators Group (NANOG) [96]. In this study, we find that upwards of two-thirds of the routing failures reported on this mailing list can be attributed to problems with routing configuration. Network operators are continually misconfiguring routing protocols in ways that cause such problems as loops, "blackholes" (where a router simply drops traffic en route to some destination because it does not have a route for it), routing instability, and so forth.

1.3 Challenges

While guaranteeing correct and predictable behavior poses challenges for any routing protocol, Internet routing presents several unique challenges. First, Internet routing has more configurable facets than traditional routing protocols, many of which can be misconfigured or otherwise cause the routing protocol to behave unpredictably. In order to analyze the correctness of the routing protocol, we must first define a specification for correct behavior. Second, the sheer size of the distributed router configuration, as well as the fact that the configurations have dependencies across routers, can give rise to erroneous or unpredictable behavior. Third, because each network operator configures his AS independently of others, the policies defined in one AS may conflict with those in neighboring ASes.

1.3.1 Defining a Correctness Specification

As described in Section 1.2.1, Internet routing configuration affords a network operator much flexibility in defining how the protocol operates. The Cisco configuration language has more than 1,000 different commands, and a network of 500 routers may have upwards of one million lines of configuration distributed across the AS [19]. Given the many ways

in which an operator can affect protocol behavior, determining the correctness of the configuration is a daunting task without a high-level specification for "correct" behavior. Developing such a specification involves distilling the high-level properties that the protocol should satisfy from the protocol's mechanistic detail.

Defining a correctness specification for Internet routing is complicated by the fact that the protocol's correctness is in part based on whether it achieves a network operator's economic and performance goals. Unfortunately, these high-level policies are encoded in terms of mechanistic configuration commands distributed across hundreds of routers—that is, the specification of the *intended* behavior doesn't even exist in the first place, which makes it difficult to determine whether the routing configuration indeed induces the intended behavior.

One motivation for developing a correctness specification for Internet routing is that the protocol not only "does the right thing" when it satisfies the specification, but a routing protocol that satisfies the specification is also more predictable. For example, network operators often would like to predict the effects of a configuration change on the behavior of the routing protocol without testing that change on a live network or running a complex simulator. Predicting how the protocol will behave first requires making assumptions about its behavior to simplify route prediction, but precisely determining the constraints that are required to simplify route prediction for such a complex protocol is challenging.

1.3.2 Analyzing Complex Configuration

In designing tools that can help network operators reason about the correctness of Internet routing, we must also design ways to manage routing configuration's staggering complexity. First, we must represent this distributed router configuration in a format that is easy to analyze. Second, we must tackle the engineering problem of parsing the various routing configuration languages from different vendors and translating the configurations into this format.

Determining how a configuration change will affect routing is difficult in practice. Not only do ASes contain a large number of routers, but the route that each router ultimately selects for each destination depends on many factors, including the routes that the AS's "border" routers learn from neighboring ASes, the routing topology within the AS (*i.e.*, both the internal topology, and the internal BGP topology that controls how routes are disseminated within the AS). As we discuss in more detail in Chapter 5, various complicating protocol artifacts prevent informally reasoning about the route that each router selects. Thus, network operators need tools and systematic techniques to assist them in predicting how a particular routing configuration will affect the flow of traffic through the AS.

1.3.3 Providing Global Guarantees with Only Local Information

While end-to-end connectivity between Internet hosts fundamentally depends on the *global* behavior of the routing system, no single party has a global view of the Internet routing system. A network operator may configure the protocol in a way that interacts in unexpected ways with the configurations in other ASes. For example, the interactions of routing policies in neighboring ASes can cause the routing protocol to oscillate.

One goal of our work is to explore how we can achieve assurances about the global behavior of the Internet routing system, while still preserving the *autonomy* of each AS.

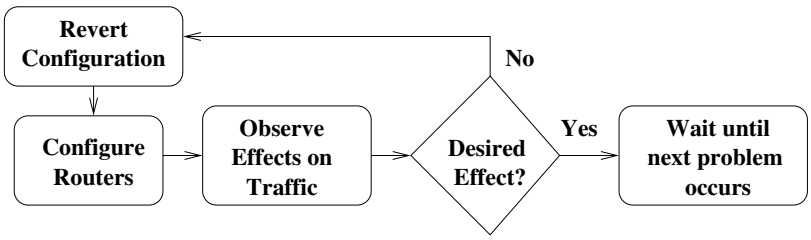


Figure 1-4: The state-of-the-art in network configuration management: *reactive*, "stimulus-response" mode of operation.

That is, each network operator should retain the ability to independently configure his own AS, but should also be able to gain some assurances about the global behavior of the routing system, assuming every AS abides by a similar set of rules.

■ 1.4 The Role of Proactive, Static Configuration Analysis

Prior to our work, the state-of-the-art in managing Internet routing protocols was *reactive*: the primary way for network operators to determine the effects of a particular routing configuration (*i.e.*, what effects that configuration will have on the flow of traffic, or whether the configuration is even correct in the first place) was to deploy that configuration on a live network, observe the resulting behavior, and revert the configuration to a previous version in the event that the configuration did not produce the desired effects (see Figure 1-4). This mode of operation has two shortcomings: First, testing configuration on a live network can cause unnecessary downtime or poor network performance (and, hence, angry customers!). Second, the undesirable effects that result from a particular configuration may not be immediately apparent when the configuration is deployed; a failure may only be triggered by the presence or absence of certain routes.

This dissertation posits that *proactive* techniques for analyzing routing configuration can both prevent a large class of routing failures and help operators predict and analyze routing protocol behavior. Changing the workflow from Figure 1-4 to include a step that proactively detects problems with routing configuration is critical for improving the correctness and predictability of Internet routing. A remarkable result of our work is that proactive analysis techniques (*i.e.*, those that analyze the static routing configuration offline, before it is deployed on a live network) are useful in detecting configuration faults and efficiently and accurately predicting the routing protocol's behavior. In particular, proactive analysis techniques provide the following benefits over the status quo:

1. **Offline.** A reactive mode of configuration is time consuming and can lead to unnecessary performance degradation. The complexity of Internet routing makes it essentially impossible to compute back-of-the-envelope estimates of the effects of configuration changes. Proactive techniques for determining the correctness and the effects of Internet routing configuration can help network operators evaluate the effects of a routing configuration before it is deployed on a live network.
2. **Accurate.** Network simulators (*e.g.*, ns [7], SSFNet [127]) help operators understand dynamic routing protocol behavior, but simulation models network behavior

in terms of its protocol dynamics over some certain period of time. The *outcome* of the simulator will ultimately be the same as that predicted by static techniques, and the dynamics (which are nondeterministic) may not necessarily correspond to those in the actual network anyhow. Existing simulators do not capture all of the relevant protocol interactions that may affect correctness, nor do they explain *why* a particular configuration is incorrect. Because incorrect behavior sometimes depends on a particular sequence of route advertisements or may be nondeterministic, correct behavior in a simulator cannot guarantee correct behavior on a live network. In this dissertation, we show that techniques based on direct analysis of static configuration files can test for necessary or sufficient correctness conditions and predict the outcome of the routing protocol without having to simulate the protocol dynamics.

3. **Efficient.** Because network operators cannot use "back of the envelope" calculations to determine what a particular routing configuration will do, they must often experiment with many possibilities before arriving at an acceptable solution. As we will show in Chapters 4 and 5, static analysis techniques can assist network operators in efficiently determining the correctness and effects of incremental changes to a routing configuration.

Analyzing the static router configurations of a single AS proves surprisingly effective at improving the correctness and predictability of Internet routing. An important open question that is *not* addressed in this dissertation involves exploring the classes of faults that *cannot* be detected with static analysis alone and how analysis of routing *dynamics* might complement static configuration analysis for fault detection and diagnosis.

■ 1.5 Contributions

We now briefly overview the major contributions of this dissertation before describing each in more detail. A central contribution of this dissertation is a formal correctness specification for Internet routing. We use this specification to derive tests for configuration faults and as the groundwork for efficient offline analysis of Internet routing. Using this specification as a guide, we present the design and implementation of two systems that use proactive configuration analysis techniques to improve the correctness and predictability of Internet routing. The first, *rcc* ("router configuration checker"), detects faults in routing configuration. *rcc* helps network operators eradicate configuration faults before they cause catastrophic routing failures on a running network. The second, the *routing sandbox*, efficiently computes the routes that each router within a single AS ultimately select, using only a static snapshot of the routing configuration and network state.

Because Internet routing inherently involves interaction between multiple independently operated, competing ASes, some aspects of the correctness specification are difficult to verify by only looking at the configuration of a single AS in isolation. In particular, it turns out to be difficult to determine whether the routing protocol will converge to a stable route assignment (a property defined as *safety* in previous work [135]). Traditional routing protocols typically satisfy safety, but BGP's *policy-based* nature means that the policies of neighboring ASes can interact to create oscillations. This dissertation provides the first necessary conditions for guaranteeing the stability of a policy-based routing protocol. The rest of this section discusses these contributions in more detail.

1.5.1 A Correctness Specification for Internet Routing

We define three high-level properties that a routing protocol should satisfy. Essentially, all three aspects of this correctness specification reflect the following underlying principle: *a routing protocol should propagate information that accurately reflects the properties of the underlying network topology*. The three properties are as follows:

- **Route validity.** If an endpoint learns a route to a destination, then that route should correspond to some path. An example of a route validity violation would be a persistent forwarding loop: a router learns a route to some destination (*i.e.*, a destination, and the next-hop IP address to which traffic should be sent for that route), but when it actually attempts to send traffic along that route, it is caught in a forwarding loop and never reaches the destination.

- **Path visibility.** If there exists a sequence of IP-level hops (*i.e.*, a *path*) from an endpoint to a destination, then that endpoint should learn at least one route to that destination. An example of a path visibility violation would be a case where all routers inside a fully-connected AS did not learn a route to the destination when at least one of those routers learned the route.

- **Safety.** This property requires that the routing protocol ultimately assigns a route to each node in the Internet graph such that no node has a more preferred available route to the destination that it would rather use. Safety is important not only because a routing protocol that persistently oscillates can cause lost and reordered packets, but also because it is incredibly difficult to diagnose problems when the routing protocol is changing independently of the underlying topology. A violation of safety would be an oscillation caused by conflicting policies from different ASes (rather than due to topological changes), as described in previous work [56, 135].

Chapter 3 formalizes each of these properties, as well as well as other concepts (*e.g.*, path, route, etc.). Path visibility and route validity are violated primarily because of configuration complexity, and safety is violated because the configurations of one AS may interact in unexpected ways with configurations of other ASes.

Although this dissertation applies this correctness specification to Internet routing, these properties should prove valuable for analyzing any routing protocol. Applying these properties to routing in other areas (*e.g.*, wireless or sensor networks) is beyond the scope of our work, but is ripe for exploration.

1.5.2 Systems for Proactive Fault Detection and Route Prediction

This dissertation recognizes a critical missing piece in the *workflow* of configuring today's networks: the step at which network operators evaluate the effects of a routing configuration before deploying and running it on a live network.

This dissertation presents two complementary systems that advance the state-of-the art in fault detection and traffic engineering, respectively. Figure 1-5 shows how these two systems change the workflow of configuring routers. Both of these new systems analyze the *static* router configuration files. We first describe *rcc*, a fault detection system for router configurations. We then describe the *routing sandbox*, a tool that predicts how traffic will

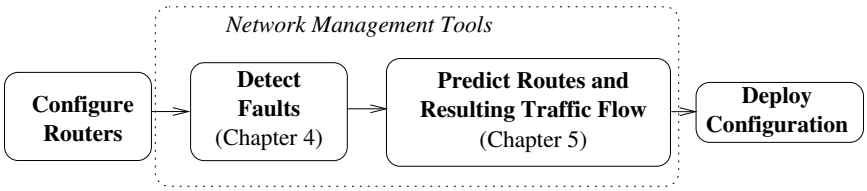


Figure 1-5: This dissertation develops two tools for fault detection and route prediction. These tools should be used to analyze the behavior of routing configuration before it is deployed on a live network.

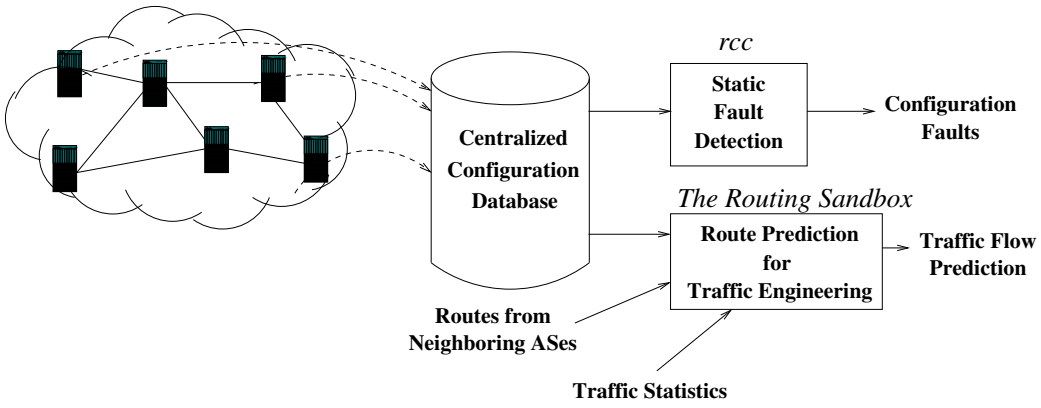


Figure 1-6: Workflow for fault detection and route prediction tasks described in this dissertation. Both fault detection and route prediction tasks rely on *offline* analysis of the distributed router configurations, which are first collected into a centralized database.

flow through an AS, given only a static snapshot of the router configurations. Together, these contributions assist a network operator in detecting faulty routing configurations and determine the effects of a configuration on traffic flow.

Because the techniques we present predict the behavior of the routing protocol before it even runs, the techniques we present directly analyze of the static configuration files. Figure 1-6 illustrates this process. The configurations from the routers within an AS are first collected from the routers and stored in a central database. Fault detection is performed by executing queries directly against this repository (Chapter 4 provides more details). Route prediction for traffic engineering (Chapter 5) is performed in a similar fashion, but also requires additional inputs: computing the routes that each router selects requires knowing which routes each router learns from neighboring ASes in the first place. Once the route that each router ultimately selects is computed, an operator could use traffic statistics to determine the utilization on each link in the network, but this dissertation does not address this problem.

rcc: Proactive fault detection for Internet routing configuration

Chapter 4 presents the design and implementation of *rcc*, the "router configuration checker", a tool that uses static configuration analysis to detect faults in the routing configurations within a single AS. We designed *rcc* starting with the correctness specification as a guide and determining the constraints on the aspects of configuration (described in more detail in Section 2.3.1) that must hold to guarantee that the higher level correctness

properties are satisfied. *rcc* analyzes the set of router configurations from a single AS and determines whether they could induce violations of the correctness specification.

rcc has been downloaded by over seventy network operators, and we have personally used *rcc* to study faults in the routing configurations of 17 real-world ASes. Chapter 5 presents algorithms and a tool that helps network operators predict how a particular routing configuration will affect the flow of traffic through the AS. In conjunction with *rcc*, this tool allows a network operator to answer critical questions about routing configuration (*i.e.*, whether the configuration will cause catastrophic problems, and how it will affect the flow of traffic through the AS) *before* the configuration is actually deployed on a live network.

Our work on *rcc* demonstrates that static configuration analysis can detect a significant number of configuration faults that could cause the routing protocol to violate correctness. Surprisingly, *rcc* also discovered many such faults in *deployed* routing configurations, even those from large, well-known Internet service providers. *rcc* even found configuration faults in ASes that use "automated" configuration techniques: of course, automated configuration systems will have difficulty generating correct configuration if they do not receive correct input in the first place. Many of the configuration faults that *rcc* detects in practice suggest that configuring a network of routers without introducing inconsistencies across router configurations is surprisingly difficult.

The Routing Sandbox: Proactive route prediction for network engineering

Chapter 5 describes algorithms that compute the effects of a configuration change offline, given only a static snapshot of the routing configurations and the available routes. These algorithms can then be combined with information about traffic demands to help network operators determine how a particular routing configuration will affect the flow of traffic through the AS. Rather than simulating complex protocol dynamics to determine the effects of configuration on route selection, we model the *outcome* of BGP's route selection process. *We exploit the properties of the correctness specification to simplify this process:* assuming that *rcc* has already verified that the configuration satisfies route validity, path visibility, and safety makes it possible to model AS-wide route selection without simulating the dynamics of the routing protocols.

Route prediction becomes increasingly difficult in ASes that enable two protocol "features": the Multiple Exit Discriminator (MED) attribute and route reflection, which are described in Sections 2.2.1 and 2.3.1, respectively. We design algorithms for predicting BGP route selection for ASes that have any combination of these two features enabled. We also perform a running-time analysis of each of these algorithms, which provides insight into how each of these features adds complexity to Internet routing.

We have implemented these algorithms in a tool called the *routing sandbox* that allows a network operator to *quickly* evaluate the effects of incremental configuration changes. This tool exploits several unique aspects of the system inputs to optimize the computation of routes for all destinations for every router in the AS. We envision that the routing sandbox could be used as an "inner loop" to other tools that iteratively search through a large parameter space to find optimal settings [146].

1.5.3 Conditions for Safety of the Global Routing System

Configuration allows an operator to control both the route that each router selects to a destination (*i.e.*, *ranking*) and which routes each router readvertises to neighboring routers (*i.e.*, *filtering*). One way to guarantee safety is to restrict some aspects of the configuration's flexibility. In Chapter 6, we derive necessary and sufficient conditions on the policies of each AS that guarantee safety if each AS independently follows these constraints. Specifically, we explore how each AS must restrict rankings to guarantee that the global routing system satisfies safety, assuming that no AS wants to share its rankings with any other AS, and no AS's rankings should be constrained by the rankings of another AS (that is, each AS should retain *autonomy*).

We show that any protocol that does not restrict the business arrangements of how ASes exchange routes with one another must impose strong restrictions on how ASes are allowed to express preferences over candidate routes for a destination. Initially, this finding may sound rather grim, because shortest paths routing may not provide network operators sufficient flexibility to achieve their economic and performance goals. On the contrary, the results we present in this dissertation should be viewed as a first cut towards designing routing protocols that are guaranteed to satisfy safety, regardless of how they are configured. Today, network operators have no way to reason about the stability of the routing protocol, so they are left to *ad hoc* methods for determining whether routing updates correspond to unintended interactions. A routing protocol that conforms to the guidelines we outline in Chapter 6 guarantees that changes in the routing protocol always reflect changes in the underlying topology, thereby facilitating troubleshooting. Furthermore, designing a protocol that is guaranteed to satisfy safety on a fast timescale allows conflicts of business policy to be resolved *outside* the protocol, rather than being reflected as oscillations within the protocol itself.

1.6 Lessons Learned

This dissertation provides the following important lessons that can aid the networking community as it considers proposals for evolving the Internet routing infrastructure.

1.6.1 Static configuration analysis detects many faults

rcc detected configuration faults in the routing configurations of all 17 ASes we analyzed and more than a thousand faults overall. It may seem surprising that *rcc* was able to detect configuration faults in *deployed* routing configurations. In fact, this finding demonstrates that there are many potentially catastrophic configuration faults that do not immediately cause routing failures when they are deployed.

The fact that static analysis can find important configuration faults without overwhelming network operators with a vast quantity of false positives is also somewhat remarkable. *rcc* does *not* operate with a specification of the routing protocol's intended behavior. Rather, it operates solely on the routing configurations that *implement* an operator's intent with low-level mechanisms. Ideally, *rcc* would check the router configurations against a high-level specification of intended behavior. It is noteworthy that *rcc* can provide a useful tool to network operators in the absence of such a specification.

Configuration languages may ultimately evolve to make certain types of configuration faults less likely, but static configuration analysis will remain a crucial step in the workflow of network operations. We believe that the more mechanistic aspects of routing configuration will ultimately be supplanted with high-level policy specification. For example, preventing routes that were learned from one neighboring AS from being advertised to another today requires configuring low-level, mechanistic operations on every router that exchanges routes with either of those ASes. Such a simple policy would better be expressed in a specification language and "compiled" to the statements that implement the mechanisms on the routers themselves. However, because Internet routing must always afford a network operator flexibility in controlling the behavior of the protocol, static configuration analysis will be invaluable for detecting faults and evaluating routing protocol behavior, *regardless of the configuration language*.

■ 1.6.2 Distributed routing configuration leads to errors

Our study of configuration faults in Chapter 4 (Section 4.5) demonstrates that most configuration faults result from the fact that routing protocol configuration is *distributed* across the routers in the AS. This approach naturally causes operators to make mistakes because it is more natural to think of the AS *as a whole* as implementing some certain task (*e.g.*, controlling traffic flow, implementing contractual arrangements, etc.) rather than reasoning about what each router must do to implement such a task. A better approach may be to allow operators to configure the AS as a monolithic entity from a single location.

Of course, a crucial open research challenge upon which this goal depends is what that centralized language should look like, and how the routers themselves should implement the directives in that language. Recent work in constraint satisfaction for network configuration presents a possible starting point for a centralized configuration language that satisfies high-level specification [97].

A logically centralized routing infrastructure could act as a catalyst for such a centralized configuration language. For example, a network operator could configure the AS from the RCP, which could either (1) compile this high-level specification into low-level router configurations, and *push the configuration* to the individual routers or (2) select the routes on behalf of each router and *push the routes* themselves to the routers.

■ 1.6.3 Safety + Autonomy ⇒ Tight restrictions on expressiveness

The (strict) conditions we derive in Chapter 6 for guaranteeing safety suggest several possible ways for evolving the Internet routing infrastructure. One possibility is to relax the autonomy requirement, by allowing groups of ASes to share certain properties about their rankings with one another (although likely not the rankings themselves). Recent work has begun exploring this possibility by recognizing that some ASes may have rankings that are more expressive as long as others are not and designing ways to guarantee these global properties without requiring ASes to divulge sensitive information about rankings [83]. One area for future work to determine the information that must be shared (and with what other ASes it must be shared) to detect and *resolve* safety violations, and, in general, to study the tradeoffs between safety and the autonomy and privacy of an ASes rankings.

Another possibility for evolving the Internet routing system is to restrict expressiveness so that rankings must be consistent with shortest paths routing, but allow each AS

to control the weights on edges incident to itself. Such a routing protocol would always satisfy safety (even assuming ASes are allowed to filter routes arbitrarily), and any policy disputes could then be resolved with a negotiation protocol that operates independently of the routing protocol, where routing updates would only reflect actual changes in the network topology. We explore this possibility and others in detail in Chapter 6 (Section 6.6).

1.6.4 Protocol design should consider correctness and predictability

This dissertation focuses on improving the correctness and predictability of the current Internet routing system, but a major lesson from our work is that many of the tools and techniques that we develop in the coming chapters could have been much simpler had the protocol been designed with correctness and predictability in mind in the first place.

To this end, this dissertation proposes several minor modifications to the Internet routing protocol that would have simplified the task of achieving correct and predictable behavior. These modifications are minor in the sense that they can be implemented with no modifications to routers or routing protocol specifications; on the other hand, they are significant because they eliminate the artifacts that result from the two most troublesome aspects of BGP: route reflection and the "multiple exit discriminator" (MED) attribute. In summary, we will see that these two artifacts are responsible for much of the undesirable behavior in Internet routing, ranging from persistent oscillation to network partitions.

One logical conclusion that can be drawn from the work in this dissertation is that, rather than trying to *infer* the protocol's behavior, the Internet routing system should provide more direct *control* over route selection. This insight is central to the Routing Control Platform (RCP) proposal, which we describe briefly in Section 7.3.4. RCP takes as input the routes that an AS learns from neighboring ASes and the network configuration, and computes routes on behalf of each router in the AS. In some sense, RCP can be viewed as the logical extension to the routing sandbox: RCP takes roughly the same inputs as the sandbox, but rather than simply computing the routes that each router would select, RCP actually controls route selection.

1.7 How to Read This Dissertation

The problems with today's Internet routing infrastructure suggest one of two attitudes:

1. Accept the Internet routing architecture "as is" and retrofit correctness and predictability by providing tools and techniques that make network operations less prone to faults and more predictable.

2. Adapt the routing architecture to make incorrect behavior less likely in the first place.

Various parts of this dissertation cater to each of these philosophies. The former philosophy can have more immediate impact and in fact can provide "bottom up" insight regarding what aspects of the routing architecture are most problematic. Chapters 4 and 5 adopt this philosophy by providing tools and algorithms that have helped network operators *today*. *rcc* has been downloaded by over seventy network operators and has successfully detected faults in the configurations of many large backbone Internet Service Providers. Additionally, the faults that *rcc* uncovered in our analysis of 17 real-world ASes, as well as

the various aspects of BGP that contribute complexity to the algorithms in Chapter 5, have helped us identify the aspects of the routing architecture that beg for improvement.

Chapter 6 explores possibilities for improving routing stability that will most likely require fundamental changes to the Internet routing architecture because the conditions for stability would require changing the configuration "knobs" that are exposed to network operators. The problems examined in this chapter use restrictions on *static* configuration of the routing protocol to guarantee stable dynamics.

This dissertation caters both to the theoretician and the practitioner. Chapter 3 presents a correctness specification for Internet routing that could appeal to both parties. Those most interested in practical applications should focus primarily on Chapters 4 and 5; Chapter 6 has fewer immediate practical applications, but will be of interest to those interested in fundamental results on routing stability and safety. At the end of Chapters 4, 5, and 6, we explore possibilities for evolving the Internet infrastructure to make the problems we solve easier in the future; these sections should also have broad appeal.

Don't look back. Something might be gaining on you.
- Leroy "Satchel" Paige

CHAPTER 2
Background and Related Work

In this chapter, we provide an overview of how routing on the Internet works today, as well as prior work on improving the correctness and predictability of Internet routing. We begin in Section 2.1 with an overview of today's Internet routing infrastructure: we describe the high-level organization of the Internet (*i.e.*, as a federation of thousands of independently operated networks that exchange reachability information) and proceed to describe in detail the routing protocols that these networks use to achieve global reachability. Section 2.2 describes how these independently operated networks exchange routing information with one another using the Border Gateway Protocol (BGP) [118, 119], and Section 2.3 both explains at a high-level how configuration controls BGP's operation and presents a brief example that describes Cisco's router configuration language syntax [19, 76]. Section 2.4 presents an overview of previous studies related to the correctness and predictability of Internet routing. Readers who are already familiar with today's Internet routing protocols and infrastructure (in particular, BGP) may wish to skip directly to Section 2.4.

■ 2.1 Internet Structure and Operation

Tens of thousands of independently operated networks connect to each other to form the larger network that we know as "the Internet". These networks are called *Autonomous Systems* (ASes), and they cooperate with one another to provide global connectivity. Nevertheless, these networks are also in *competition* with one another. Each one of these ASes contains many routers. The routers inside of one of these networks run an internal routing protocol called an *interior gateway protocol* (IGP) that allows them to discover routes to other destinations within the same AS, including the AS's *border routers*—those routers that connect to neighboring ASes. Examples of IGPs are Open Shortest Paths First (OSPF) [91], Intermediate System-Intermediate System (IS-IS) [104], and Routing Information Protocol (RIP) [67].

The Internet is composed of many different types of ASes, from universities to corporations to regional Internet Service Providers (ISPs) to nationwide ISPs. Smaller ASes (*e.g.*, universities, corporations, etc.) typically purchase Internet connectivity from ISPs. Smaller

regional ISPs, in turn, purchase connectivity from larger ISPs with "backbone" networks.

The different types of ASes lead to different types of business relationships, and, hence, different policies for exchanging and selecting routes. Although we will expound on these business relationships later in this chapter, it is reasonable to think of these business relationships in terms of two types: *customer-provider* and *peering*. Customer-provider relationships involve one AS (the "customer") paying another (the "provider") in exchange for carrying its traffic to some portion of the Internet's destinations (often, every destination outside of its own network) [45]. In today's Internet routing, *a route advertisement is an implicit agreement for carrying traffic*. The process of carrying traffic between two different ASes is called "providing transit". In these relationships, the customer pays the provider for transit, regardless of the direction in which the traffic is flowing.

In *peering* relationships, two ASes agree to trade traffic to various destinations at no cost. Typically, a pair of ASes will recognize that it is more cost-effective to directly exchange traffic to (some or all of) one another's customers, rather paying to send the traffic through one or more provider ASes. For those interested in a more detailed treatment of the business aspects of peering, Norton provides an excellent overview of peering and the decision parameters that ASes must consider for deciding whether to peer [99].

2.2 Internet Routing: The Border Gateway Protocol

In this section, we describe the operation of the Border Gateway Protocol, version 4 (BGP) [118, 119]. The first two sections describe the basic operation of the protocol—the protocol state machine, the format of routing messages, and the propagation of routing updates—all of which is defined in the protocol standard [118]. A noteworthy aspect of BGP is that many of the features that determine the behavior of the global routing system are *not* standardized. Later in this section, we discuss two important non-standard aspects of Internet routing: the route selection process and configuration languages.

To ensure reliable delivery of routing messages, all BGP sessions exchange information using the Transmission Control Protocol (TCP) [109] (the same transport protocol used by common Internet applications that require reliable message delivery, such as email and the Web). Like TCP, BGP also has a protocol state machine. Because BGP's state machine is primarily concerned with enabling two routers to establish a communication channel with one another and is unconcerned with the routing messages themselves (all routing messages are exchanged in the "`ESTABLISHED`" state), we forgo further description of BGP's state machine. For a detailed description of BGP's finite state machine (including how timers can affect the transition between protocol states, and when these timers are reset), see the protocol standard and related documents [6, 118, 128].

2.2.1 Route Propagation: Announcements and Withdrawals

To understand how routers exchange routes using BGP, it is important to keep in mind several defining features. First, BGP is a *path vector* protocol. In a path vector protocol, routing updates contain the sequence of ASes that the routing advertisement traversed (*i.e.*, the *AS path*). BGP includes AS path information to avoid the "counting to infinity" problem that exists in traditional distance vector protocols [64]. The AS path allows an AS that learns a route to determine whether or not it has already heard the route by checking

Route Attribute	Description
Next Hop	IP Address of the next-hop router along the path to the destination. On eBGP sessions, the next hop is set to the IP address of the border router. On iBGP sessions, the next hop is not modified.
Multiple-Exit Discriminator (MED)	Used for comparing two or more routes from the same neighboring AS. That neighboring AS can set the MED values to indicate which router it prefers to receive traffic for that destination. *By default, not comparable among routes from different ASes.*
Local Preference	This attribute is the first criteria used to select routes. It is not attached on routes learned via eBGP sessions, but typically assigned by the import policy of these sessions; preserved on iBGP sessions.

Table 2-1: Commonly used BGP route attributes.

to see whether its own AS is contained in the path.[1] Destinations are represented as IP *prefixes*, as described in Section 1.1. A BGP route announcement has several associated route attributes in addition to the AS path, many of which are obsolete. The most relevant BGP route attributes are summarized in Table 2-1.

Second, BGP maintains state about the routing topology: routers do not periodically "refresh" routing reachability information; rather, routing messages reflect only *changes* in this information. These changes in information are reported with two types of routing updates: *announcements* and *withdrawals*. To announce that it can reach a destination (or to change the existing route to a destination), a BGP-speaking router sends an announcement for that destination to a neighboring router. If a destination is no longer reachable, a router sends a withdrawal message to the neighboring router. That neighboring router may be located either in a neighboring AS or in the same AS. BGP sessions between routers in the same AS are called *internal BGP (iBGP)* sessions, and those between routers in different ASes are called *external BGP (eBGP)* sessions. The goal of eBGP is to allow ASes to exchange reachability to destinations in each other's networks; the goal of iBGP is to ensure that *every* router within an AS learns at least one route to every destination.

2.2.2 Route Selection (And How Operators Can Control It)

Any given router may learn multiple routes to a destination (*i.e.*, IP prefix), but must ultimately select a single best route along which to forward traffic to that destination. The route selection process determines which route each router selects. The original standards document does not specify the route selection process [118], but the route selection process has since become a *de facto* standard [17, 119].

[1] Contrary to what many believe, the AS path is *not* intended to indicate the sequence of ASes that traffic will traverse en route to the destination, although the AS path and this sequence of ASes often match [88].

Step	Criterion	How Configuration Can Manipulate This Step
1	Highest local preference	AS's import policy
2	Lowest AS path length	Neighboring AS can "prepend" additional hops
3	Lowest origin type	Obsolete
4	Lowest MED (with *same* next-hop AS)	Neighboring AS's export policy
5	eBGP-learned over iBGP-learned	—
6	Lowest IGP path cost to egress router	AS's IGP topology
7	Lowest router ID of BGP speaker	—

Table 2-2: Steps in the BGP route selection process.

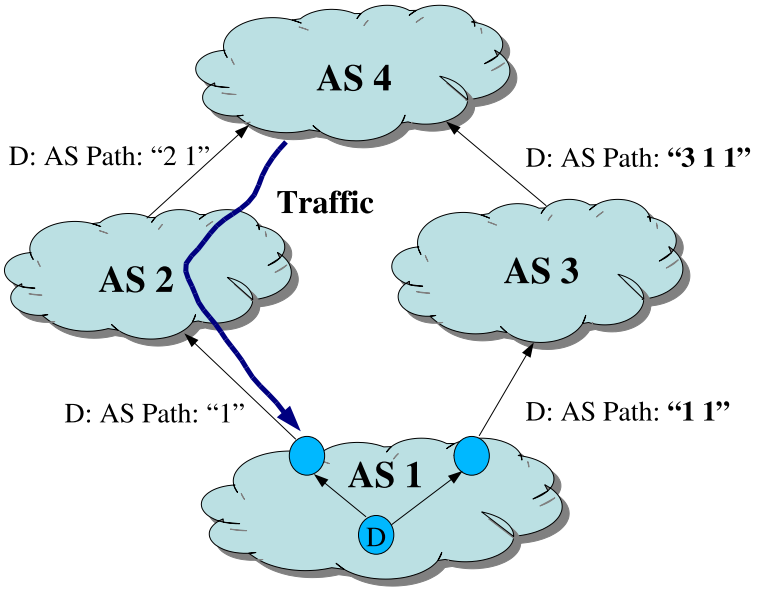


Figure 2-1: Operators sometimes use *AS path prepending* to try to control inbound traffic.

Table 2-2 summarizes the route selection process, as well as how network operators can use routing configuration to try to influence which route each router selects at each step of the process. First, given multiple routes to the same destination (*i.e.*, IP prefix), a router will select the route with the highest "local preference" value. As this attribute is not set by the receiving AS's import policy and is the first step in the decision process, it provides the operator direct influence over which route the router ultimately selects to this destination. Operators typically use the local preference attribute to implement the types of policies described in Table 2-3.

Among multiple routes to a destination with equal local preference values, a router will select the route with the shortest AS path length, which is simply the number of AS-level hops in the AS path. This criterion is a crude approximation for selecting a shortest path to the destination. In practice, a path with the fewest number of ASes does not correspond to the path with the lowest latency, or to the path with the fewest number of IP-level hops. However, network operators do use routing configuration to try to influence route selection using a technique called *AS path prepending*, which artificially increases the length of a route's AS path by adding the same AS number to the path multiple times. Figure 2-1

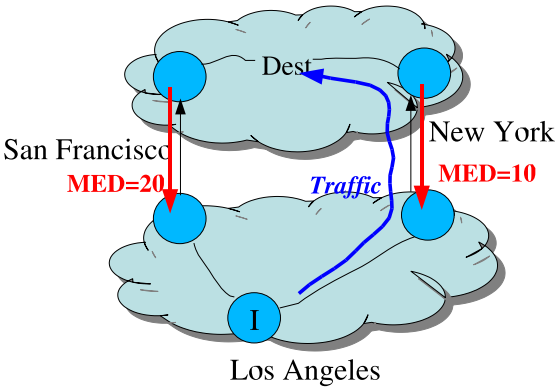


Figure 2-2: A neighboring AS can advertise routes to a destination with different MED values at different locations to control the exit point that routers in a neighboring AS uses to send traffic for that destination. When I learns both routes, it will select the route learned via the router in New York (and, thus, it will send traffic that way).

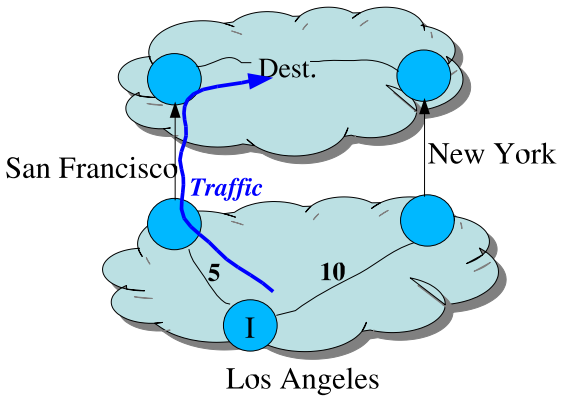


Figure 2-3: If router I learns two or more routes that are equally good up through Step 5 in Table 2-2, it will select the route whose next hop has a shortest IGP path cost. IGP path costs are shown as numbered edges; these path costs often correspond to geographic distance.

illustrates this practice: AS 1 wants inbound traffic for D to arrive via AS 2, rather than via AS 3. To express this preference, it prepends an additional "1" to the AS path for its route to D when it advertises the route to AS 3. If an upstream AS, say AS 4, learns two routes to D—*i.e.*, one via AS 2 and the other via AS 3—it will prefer the route with the route via AS 2 because it has a shorter AS path length (assuming it does not assign a higher local preference to routes learned from AS 3). Of course, this technique for controlling inbound traffic has limited utility in practice, because AS 1 does not control the policies (*i.e.*, local preference values) of other ASes, and it also cannot easily determine the AS path lengths that an upstream might see for its route advertisements [48]. Despite the fact that prepending is widely used, the technique is largely ad hoc and of only limited utility [32, 112].

If a router has two routes to the same destination with equal local preference and AS path length, the router will then select the route with the lowest "origin type". Because this route attribute is deprecated, in practice, route selection then typically falls to selecting the route with the lowest multiple exit discriminator (MED) value. If a neighboring AS advertises a route to an AS at multiple locations, it may attach different MED values to these routes to indicate that it prefers a neighboring AS to use one exit point over another. For example, in Figure 2-2, AS 1 is indicating to AS 2 that it wishes to have traffic sent to destination d via the exit point in New York over the one in San Francisco by advertising the route to d with a larger MED value over the BGP session in San Francisco. By default, the MED value is only comparable among routes learned from the same neighboring AS, since different ASes may use different ranges of MED values to specify preferences over routes. As we will see later in this dissertation, the fact that the MED attribute is not comparable across all routes creates many problems for correctness and predictability.

If multiple routes remain after Step 4, a router will prefer a route that it learned via eBGP over one that it learned via iBGP. If multiple routes still remain, the router will pre-

fer the route whose next-hop IP address is "closest" in the internal routing topology (*i.e.*, the shortest IGP path). This step allows a network to achieve what is commonly referred to today as "hot potato routing": the process by which an AS tries to offload traffic to neighboring ASes as quickly as possible.[2] Figure 2-3 illustrates this mechanism. A network operator could conceivably control interdomain traffic by adjusting edge weights in the IGP. Unfortunately, updates in the IGP topology can cause unexpected and unwanted shifts in BGP routes, potentially affecting large volumes of traffic [130].

If multiple routes remain after the IGP tiebreak, the routers may break ties in a number of ways. This final tiebreak is usually based on the "router ID" of the router that advertised the route, although other tiebreaking mechanisms are sometimes used, such as selecting the "most stable" route (*i.e.*, the one that has been advertised for the longest period of time).

A key problem operators face is determining which route each router in the AS will select. It might seem that this process might be as simple as taking the set of eBGP-learned routes for a destination and applying the process in Table 2-2 at each router. In fact, predicting the outcome of this process is not so simple: Chapter 5 is dedicated to solving this problem.

■ 2.2.3 Putting It Together: How Traffic Gets from Here to There

We now describe how IGP, iBGP, and eBGP act in concert to establish routes between various endpoints. One can think of a route in terms of two distinct phases: (1) the route to some exit (or "egress") router in that AS (or to the ultimate destination, if the destination is located in the same AS) and (2) the route from the egress point to the appropriate next-hop AS.

An example of the route to a destination in a router's routing table is shown in Figure 1-3: each destination prefix has a next-hop IP address to which to send traffic. If the destination is in a different AS and the router is not an egress router, then that next hop is typically the IP address of an egress router. The BGP route selection process determines which egress router each router sends traffic to: each router in the AS may learn a route for some destination from one or more egress routers, but ultimately selects only one of those routes. The AS's IGP is then responsible for determining the route from that router to the egress router named by that next-hop IP address.

If, on the other hand, the destination is in a remote AS and the router is an egress router, the router will either select a route with a next hop that is in a neighboring AS, or it will select a route learned from a different egress router and rely on the AS's IGP to forward traffic to the egress router with that next-hop IP address.

[2] "Hot potato routing" was initially coined by Paul Baran for *all* routing techniques where nodes would forward messages as quickly as possible [62]. This notion stood in contrast to quintessential "store and forward" networks like the telegraph. In these systems, the electrical signal would dissipate after some distance. As such, relay nodes would transcribe the message in Morse code, and an operator would then re-feed the ticker to the relay node, which would regenerate the electrical signal.

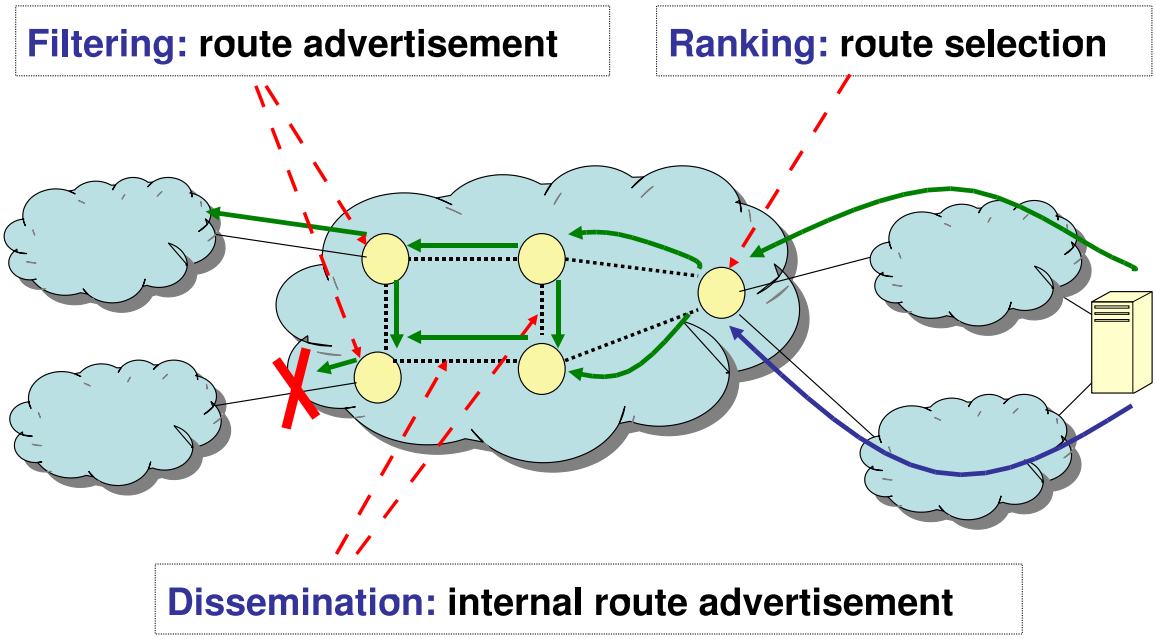


Figure 2-4: BGP configuration semantics.

2.3 Internet Routing Configuration

Internet routing's true complexity lies in the fact that so many aspects of the routing protocol's operation are manipulable with configuration. As discussed in Chapter 1 (Section 1.2), and as we will see throughout this dissertation, many of the problems faced by the Internet routing system result from the protocol's configurability.

This section provides background on Internet routing configuration. Internet routing configuration languages typically have thousands of distinct commands [19]; the reader will be pleased to learn that we will not survey all of them here. Rather, we first classify the *semantics* of routing configuration into three main operations: ranking, filtering, and dissemination. We then provide a brief example of Cisco router configuration *syntax* to demonstrate how a network operator can implement these operations in practice.

2.3.1 Semantics: Ranking, Filtering, Dissemination

Internet routing configuration allows the routing protocol to achieve two important goals:

1. *Policy.* Internet routing must be flexible enough to implement complex business relationships. Internet routing configuration provides network operators the ability to encode these policies in router configurations.
2. *Scalability.* The Internet must scale to a large number of hosts, routers, and ASes. To achieve this scalability, Internet routing configuration allows an operator to specify various ways for the routing protocol to aggressively aggregate routing information (*e.g.*, using route reflection).

The rest of this section describes how routing configuration's three main operations—ranking, filtering, and dissemination—facilitate the expression of complex business poli-

Type of neighboring AS	**Ranking**	**Filtering**
Customer	Most preferred	Advertise to all other ASes
Peer	Less preferred than routes through customer, more preferred than routes through provider	Advertise to customer ASes
Provider	Least preferred	Advertise to customer ASes

Table 2-3: Common business relationships and practices between ASes on the Internet today. Although this table summarizes the conventional wisdom of how ASes commonly interact today, Section 6.1 describes several violations of these practices.

cies and provide options for achieving scalability. We also briefly explain how each of these operations can affect the correctness and predictability of Internet routing.

Policy: Ranking and Filtering

The bilateral business relationships established between ASes (as described in Section 2.1) imply that the *policy* afforded by Internet routing configuration should provide a network operator two degrees of control: (1) which route each router in the AS should prefer, given multiple routes to a destination (*ranking*); (2) which routes should be advertised to which neighboring ASes (*filtering*). Table 2-3 summarizes these common practices for both ranking and filtering. Although these practices constitute the conventional wisdom for how ASes operate, Section 6.1 presents examples of some common deviations from these practices. We now explain these practices in more detail in this section.

Because a customer pays a provider per unit traffic regardless of the direction the traffic is flowing, it is to an AS's advantage to select routes to destinations via its customer ASes, given the option. Similarly, an AS would typically prefer to send traffic through one of its peers (which it can do at no additional cost) versus sending traffic via one of its providers (which will charge it for the service of carrying that traffic).

A router's configuration can prevent a certain route from being accepted on inbound or readvertised on outbound. Configuring filtering is complicated because global behavior depends on the configuration of individual routers. An AS will typically advertise its entire set of routes to its customers (who it will gladly charge to carry traffic to any of those destinations) but will only advertise to one of its peers the routes that it learned from one of its customers. On the other hand, an AS will not advertise routes that it learns from one if its providers to another one of its providers: doing so would cause that AS to provide transit between two of its providers and pay *both* of its providers to boot!

Scalability: Dissemination

A router's configuration controls the *dissemination* of routes within the AS and between neighboring ASes by allowing each router to establish BGP sessions with neighboring routers. Router configuration allows an the router to establish two types of BGP sessions: those to routers in its own AS (iBGP) and those to routers in other ASes (eBGP). A small AS with only two or three routers may have only 10 or 20 BGP sessions, but large backbone networks may have more than 10,000 BGP sessions, more than half of which are iBGP

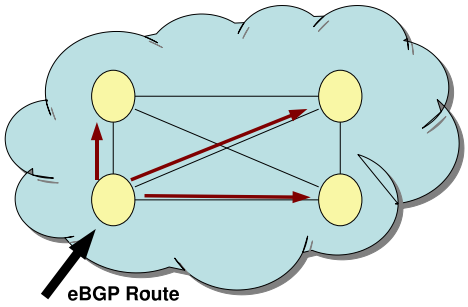
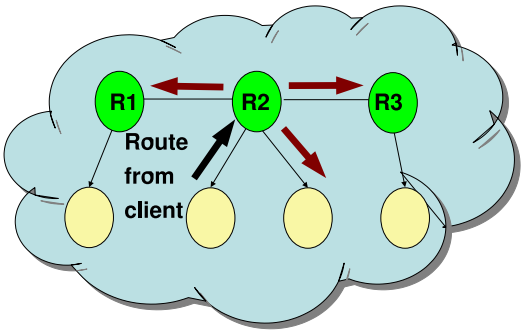
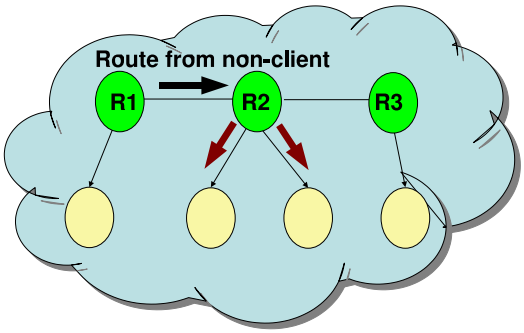


Figure 2-5: Small ASes establish a clique (or "full mesh") of iBGP sessions. Each circle represents a router within an AS. Only eBGP-learned routes are readvertised over iBGP sessions.



(a) Routes learned from clients are readvertised over all iBGP sessions.



(b) Routes learned from non-clients are readvertised to clients only.

Figure 2-6: Larger ASes commonly use route reflectors, which advertise some iBGP-learned routes as described above. Directed edges between routers represent iBGP sessions from route reflectors to clients (*e.g.*, router $R2$ is a route reflector with two clients). As in Figure 2-5, all routers readvertise eBGP-learned routes over all iBGP sessions.

sessions.

BGP messages propagate differently depending on whether the update is propagating over an eBGP session or an iBGP session. An eBGP session is typically a *point-to-point* session: that is, the IP addresses of the routers on either end of the session are directly connected with one another and are typically on the same local area network. There are, of course, exceptions to this practice (*i.e.*, "multi-hop eBGP" [24]), but directly connected eBGP sessions is normal operating procedure. In the case where an eBGP session is point-to-point, the next-hop attribute for the BGP route is guaranteed to be reachable, as is the other end of the point-to-point connection. A router will advertise a route over an eBGP session regardless of whether that route was originally learned via eBGP or iBGP.

On the other hand, an iBGP session may exist between two routers that are *not* directly connected, and it may be the case that the next-hop IP address for a route learned via iBGP is more than one IP-level hop away. In fact, as the next-hop IP address of the route is typically one of the border routers for the AS, this next hop may not even correspond to the router on the other end of the iBGP session, but may be several *iBGP* hops away.

In iBGP, the routers thus rely on the AS's internal routing protocol (*i.e.*, its IGP) to both (1) establish connectivity between the two endpoints of the BGP session and (2) establish the route to the next-hop IP address named in the route attribute.

The session-level iBGP topology determines how BGP routes propagate through the network. By default, a router will only readvertise a route on an iBGP session if it learned that route via an eBGP session. This constraint requires that every router in an AS have an iBGP session with every router that learns routes via eBGP (typically every other router), as shown in Figure 2-5; *i.e.*, those routers must form a clique (the networking community commonly refers to this configuration as a "full mesh" iBGP topology).

ASes with a small number of routers are often configured in a full mesh iBGP topology, but a fully meshed iBGP topology requires $O(n^2)$ sessions for an AS with n eBGP-speaking routers, which does not scale well. Two alternatives have been proposed to solve these problems: route reflection [5] and confederations [131]. To improve scalability, larger networks typically use *route reflectors*. A route reflector selects a single best route and announces that route to all of its "clients". A route reflector is defined by the fact that it has *client* routers, and it readvertises its iBGP-learned routes to some other routers in the same AS according to the following rules: (1) if a route reflector learns a route via eBGP or via iBGP from one of its clients, it readvertises that route over all of its sessions to its clients; (2) if it learns the route via iBGP from a router that is not one of its clients, it readvertises the route to its client routers *but not over any other iBGP sessions*. Figure 2-6 shows an example route reflector hierarchy and how routes propagate from various iBGP sessions.

Configuring an iBGP topology correctly is relatively difficult; we discuss iBGP misconfiguration in more detail in Section 4.2. Incorrect iBGP topology configuration can create many types of incorrect behavior, including persistent forwarding loops and oscillations [61]. Route reflection causes problems with correctness because not all route reflector topologies are guaranteed to propagate a route learned via an eBGP session (*i.e.*, not all iBGP topologies satisfy path visibility). We describe this problem in more detail in Chapter 4.

Route reflectors also complicate predictability because they prevent each router from learning every BGP route. Thus, the BGP route that each router ultimately selects may not be the same route that it would have selected had it learned every BGP route for that destination (as it would have in a full mesh iBGP topology). This property makes route prediction more difficult because the routes that some routers ultimately select depend on the routes that other routers in the AS select. Efficiently computing the route that each router selects thus requires determining these dependencies and "visiting" the routers within the AS in the correct order. Chapter 5 describes in more detail how route reflection complicates predicting route selection within an AS.

Dissemination primarily concerns flexibility in iBGP configuration, but the configuration may also *manipulate* route attributes when disseminating routes for one of the following reasons: (1) controlling how a router ranks candidate routes, (2) controlling the "next hop" IP address for the advertised route, and (3) "tagging" a route to control how the ranking and filtering functions on other routers treat it.

```
router bgp 7018
   neighbor 192.0.2.10 remote-as 65000
   neighbor 192.0.2.10 route-map IMPORT in

   neighbor 192.0.2.20 remote-as 7018
   neighbor 192.0.2.20 route-reflector-client
!
route-map IMPORT permit 1
   match ip address 199
   set local-preference 80
!
route-map IMPORT permit 2
   match as-path 99
   set local-preference 110
!
route-map IMPORT permit 3
   set community 7018:1000
!
ip as-path access-list 99 permit ^65000$
access-list 199 permit ip host 192.0.2.0 host 255.255.255.0
access-list 199 permit ip host 10.0.0.0 host 255.0.0.0
```

Figure 2-7: Example of a Cisco router configuration.

■ 2.3.2 Syntax: Routing Configuration Languages

Figure 2-7 shows an example of how ranking, filtering, and dissemination are encoded into a router configuration for a single router. The example shows an excerpt from a Cisco router configuration; each router vendor has a different configuration language, although many are similar to Cisco's. The first clause indicates that this router is located in AS 7018. BGP sessions to neighboring routers are indicated with the `neighbor` statements. The router has a BGP session with IP address `192.0.2.10` in AS 65000.

The second `neighbor` statement specifies that the "inbound route map" (*i.e.*, import policy) called `IMPORT` should be applied to the route advertisements learned on this BGP session. This policy has two clauses that implement the import policy. The first clause assigns a local preference value of 80 for advertised routes to `192.0.2.0/24` and `10.0.0.0/8`, as defined in access-list 199. The second clause assigns a local preference of 110 to routes with an AS path of 65000 (*i.e.*, a one-hop path to AS 65000). All remaining routes are assigned the default local preference value, 100, and tagged with a "community" value of `7018:1000`. By itself, the community value has no meaning; it is nothing more than a label. Some other router's configuration, however, may have an import or export policy that takes some action (*e.g.*, filtering the route, changing its local preference value, etc.) based on this value. Setting the community attribute on one route and acting on that community on another introduces dependencies across routers that can be difficult to debug.

This router also has a BGP session to a router with the IP address `192.168.2.20`. The `remote-as` command indicates that this router is in AS 7018—the same as the

router of this AS, as specified with the `router bgp` statement—which implicitly configures this session as an iBGP session. The next line of the configuration indicates that `192.168.2.20` is a route reflector client. No additional configuration (beyond simply setting up a regular iBGP session) is required on the client router to establish that it is a client.

Figure 2-7 shows an excerpt of a Cisco configuration, but a noteworthy aspect of BGP configuration is that *the configuration language is not standardized*. As a result, an AS may contain routers from many different vendors (*e.g.*, Cisco, Juniper, Avici, etc.). Although all routers can exchange routes using the standard BGP message format, their configuration languages are often different. This heterogeneity makes the static configuration analysis problems described in Chapters 4 and 5 even more challenging.

2.4 Related Work

While Internet routing achieves its policy and scalability goals fairly well, the high degree of configurability that allows these goals to be met also presents challenges both for *correctness* (*i.e.*, preventing mistakes and unintended interactions) and for *predictability* (*i.e.*, determining offline how the protocol will behave in practice). This section surveys previous approaches to addressing these two challenges and how they relate to the work in this dissertation.

2.4.1 Correctness

Operator-induced configuration faults are perhaps the single biggest threat to the correct operation of Internet routing today. After surveying previous studies on the effects of configuration faults (and resulting routing instability) on end-to-end performance, we survey previous work on configuration management tools, which help operators audit configuration changes and detect faults.

How Routing Problems Affect Connectivity and Performance (Why Correctness Matters)

Many researchers have studied both the effects of misconfiguration and routing instability on network downtime and end-to-end performance. This section highlights the results of some previous studies, which provide supporting evidence for the adverse effects of routing instability on end-to-end performance. Table 2-4 summarizes these previous findings in terms of three categories: those that study the effects of configuration faults on end-to-end performance, those that study the effects of routing instability on end-to-end performance, and those that study the effects of various routing protocol artifacts on routing stability and convergence (both of which indirectly affect end-to-end performance).

Mahajan *et al.* studied the effects of BGP misconfiguration on connectivity disruptions [85]. This work studied short-lived BGP misconfiguration by analyzing transient, globally visible BGP announcements from an edge network. They defined a "misconfiguration" as a transient BGP announcement that was followed by a withdrawal within a small amount of time (suggesting that the operator observed and fixed the problem). They found that many misconfigurations are caused by faulty route origination and incorrect

Year	Author	Analysis Technique	Major Results
		How Configuration Faults Affect End-to-End Performance	
2002	Mahajan *et al.* [85]	Measurement/Email survey	30% of all configuration "slips" that cause short-lived BGP announcements disrupt connectivity.
2002	Griffin *et al.* [61]	Theoretical analysis	Some iBGP configurations can cause protocol oscillations and persistent forwarding loops.
		How Routing Instability Affects End-to-End Performance	
1997	Paxson [106]	End-to-end measurement	Routing-induced path failures for 0.21-0.5% of end-to-end observations.
2001	Labovitz *et al.* [78]	Fault injection	Up to 30% packet loss during periods of routing instability
2003	Feamster *et al.* [28]	End-to-end measurement	50% of all end-to-end path failures correlate with BGP instability.
2005	Bush *et al.* [12]	Fault injection	85% of BGP instability events cause loss periods of 15 seconds or longer.
		How Protocol Artifacts Affect End-to-End Performance	
2001	Labovitz *et al.* [78]	Measurement/Analysis	Some failures take up to 15 minutes to converge after failover. Some routers may explore $O(n!)$ paths, where n is maximum AS path length.
2001	Griffin *et al.* [55]	Simulation	Advertisement timer settings significantly affect convergence time.
2002	Mao *et al.* [86]	Simulation	Route flap damping can slow convergence by several orders of magnitude.

Table 2-4: The results of previous empirical studies of the effects of routing faults and protocol artifacts on routing convergence or end-to-end performance.

filtering. *rcc* (Chapter 4) can help operators find these faults; it can also detect faults that are difficult to quickly locate and correct. *rcc* also helps operators detect the types of misconfigurations found by Mahajan *et al.* [85] *before* deployment.

Griffin *et al.* examine two aspects of *iBGP* correctness that may affect the end-to-end delivery of traffic: non-convergence and "deflections", whereby packets do not follow their intended path [61]. This work does not observe how often incorrect iBGP routing occurs in practice.

Previous work has gathered evidence to suggest that routing instability and configuration faults have serious ramifications for end-to-end connectivity. Paxson studied the

end-to-end properties of Internet paths by performing two separate experiments between 37 hosts distributed across the Internet; each path was probed with `traceroute` approximately once every day or two (in a second experiment, a fraction of the paths were probed approximately once every two hours) [106]. Each experiment was conducted over the course of approximately 6 weeks. The first experiment was in 1994, and the second was in 1995. In this work, Paxson studied both general routing pathologies (*e.g.*, asymmetric paths, erroneous routing, route flapping), and also studied the times during which paths were unreachable due to failures of the routing infrastructure. Paxson found that paths were unavailable for 0.21% of the time in his first sample and 0.5% of the time during his second experiment.

In 2002-2003, Feamster *et al.* performed a similar study on the RON testbed [3] over the course of 13 months. This work extended Paxson's study by incorporating both more frequent active probes (each of approximately 900 geographically and topologically diverse paths was probed at least once every 90 seconds), which triggered `traceroute` probes upon detecting a reachability failure. The BGP routing information was collected at sites that were co-located with the measurement hosts [28]. About half of the end-to-end failures observed coincided with some BGP routing instability. This study considered only correlation between BGP routing instability with end-to-end path failures; an interesting future direction would be to determine how many of the path failures were *caused* by routing instability (*i.e.*, cases where the routing protocol actually disrupted communication) versus those that were simply reflected by instability.

Labovitz *et al.* observed that BGP undergoes a process called "path exploration", a process by which routers select (and propagate) alternate routes upon learning a route withdrawal [78]. They showed that, in theory, during convergence, a router may explore $O(n!)$ alternate routes, where n is the maximum AS path length to the destination. To study this phenomenon in practice, they injected routing faults into the running network and measured the duration of the convergence process, finding that convergence may take as long as 15 minutes when a route is withdrawn [78]. Bush *et al.* performed a similar study that artificially injected routing updates into the network from "BGP beacons" [87]; this experiment created instability on a small set of paths, on which they then perform more targeted observations to study the properties of end-to-end paths during these periods of instability. While they observe that many periods of prolonged loss do not correlate with routing instability, they also find that most episodes of routing instability cause periods of prolonged loss.

Several other researchers have examined the effects of various protocol artifacts on convergence time. Mao *et al.* observed that damping routes that oscillate can cause significant delays in convergence; in pathological topologies, BGP may take roughly an hour to converge after a single route withdrawal [86]. Other work has simulated the effects of BGP's timer settings on convergence time [55]. This dissertation does not address correctness problems that occur during the convergence process.

Configuration Management Tools: Helping Operators Cope with Complexity

Many network operators use configuration management tools such as "rancid" [113], which periodically archive and manage versions of router configurations. When a network problem coincides with the configuration change that caused it, these tools can help oper-

	Traffic eng.	Failure Analysis	Static Fault Detection	Monitoring	Inventory	Revision Mgmt.	"Automated" Configuration
Fault Detection and Traffic Engineering							
NetSys/IPAT [140]		•	•				•
OpNet SP Guru [102]		•					
Cariden MATE [15]	•						
Change Management							
Intelliden R-Series [72]					•	•	
Redcell [114]						•	
VoyenceControl [115]						•	
Opsware NAS [103]						•	
Tripwire [132]						•	
Route Analytics							
HP RAMS [69]				•			
RouteDynamics [73]				•			
Route Explorer [105]				•			

Table 2-5: Existing configuration management tools, which generally fall into three categories: fault detection and traffic engineering, change management, and route analytics. The fault detection tools are most related to *rcc*, and the traffic engineering tools are most related to the model and tool in Chapter 5. Change management tools help an operator audit configuration changes and revert to a previous version of the configuration when faults are discovered. Route analytics products rely on analyzing protocol *dynamics* to detect faults.

ators revert to an older configuration. Unfortunately, a configuration change may induce a fault that becomes active later, and these tools do not detect whether the configuration has these types of faults in the first place.

Some tools analyze network configuration and highlight rudimentary configuration errors. One such tool was Cisco's Netsys-Agent, which was decommissioned in November 2000 and evolved into a product called IP Analysis Tools (IPAT), supported by the Wide Area Network Design Laboratory [140]. IPAT periodically collects the configurations from the network's devices and helps network operators diagnose "connectivity issues". The product also allows network operators to evaluate "what-if" scenarios (*i.e.*, how a particular configuration change will affect network connectivity and topology), verify that a network configuration satisfies connectivity requirements (*e.g.*, that two nodes in the network can reach each other), and evaluate various failure scenarios. IPAT also provides graphical interfaces to network operators that assist them in viewing router configurations and routing tables.

OpNet's NetDoctor product analyzes the configurations of many routing protocols, including OSPF, IS-IS, RIP, MPLS, and BGP [100]. A white paper on NetDoctor provides examples of the types of fault detection that the tool performs, such as: checking that two interfaces on the opposite ends of an OSPF edge are in the same area, checking consistency of BGP "hold timer" values, checking for redundant access control lists, enabling system logging, blocking ICMP and telnet, etc. [101] NetDoctor also appears to allow operators to check their configurations against best common practice (an issue we also tackle in Section 4.3). Although the checks performed by NetDoctor are similar in spirit to those performed by *rcc* (Chapter 4), the work we present in this dissertation focuses more on

verifying *network-wide* properties of routing, rather than simply performing checks on the consistency of the configuration of a single router (or pair of routers)—NetDoctor does not implement these types of checks.

Intelliden provides a product called R-Series, which helps network operators keep track of device inventory and changes to the network configuration [72]. The product provides: (1) a device modeling application, which translates the differing configuration languages for a device (*i.e.*, depending on vendor, type, model, and operating system) into a single, independent XML representation, (2) a system for managing version histories of the network configuration, and (3) a framework for managing device inventory. Intelliden's R-Series does not provide any automated fault detection or debugging support; it is primarily a tool to help network operators manage the complexity that results from heterogeneous devices and multiple network operators who can make changes to the configuration.

Several other similar products exist to assist network operators with auditing configuration changes: Redcell allows a network operator to update the network configuration from a centralized location and also performs version control and automated backup [114]. Voyence's VoyenceControl allows an operator to configure network devices from a centralized server and performs some validation of configuration before deployment [115]. Opsware's Network Automation System [103], and Tripwire [132] also monitor configuration changes to network devices.

Other commercial tools do not directly analyze the configuration, but rather monitor routing dynamics and network performance to assist operators in finding problems (including possible configuration faults). Hewlett Packard (HP) offers a Route Analytics Management System (RAMS) [69], which participates in the routing protocol with the routers in the network, actively detects problems related to OSPF, IS-IS, BGP, and EIGRP, and reports these problems to the network operator. RAMS also correlates routing information with other performance data to help network operators diagnose the underlying cause of a problem. HP's OpenView product line has many other tools designed to help operators with network and application management [68].

Ipsum Networks (now defunct) developed a product called RouteDynamics that actively participated in a network's IP routing protocols and monitored routing activity to predict and analyze potential routing problems [73]. The product primarily focuses on helping operators both detect routing instability and identify the cause of the instability. The product also allows network operators to view and "play back" historical routing data, as many of the routing instabilities that affect network performance are transient.

Most network management tools detect rudimentary errors and track changes to the network configuration for auditing purposes, but they do not help a network operator determine whether a network configuration will actually achieve the intended behavior. A deficiency in today's router configuration languages is that there is no high-level language with which a network operator can specify policies. The lack of such a language not only makes configuring the network more complex, but it also makes deducing faults more difficult: any fault detection technique must infer the operator's intent solely from the low-level configuration. Recent work proposes a "service grammar" for BGP, which includes a requirements language with which operators can specify higher-level requirements against which the system can be checked [110]. While developing such a grammar would ease the task of both specifying and validating routing configuration, the proposed

grammar is still rather low-level: for example, it requires operators to specify requirements such as the existence of a BGP session between pairs of routers, which routers are route reflectors for which other routers, etc. Such a grammar could have just as many errors as the configuration itself and still does not specify the intended behavior of the network at a high enough level (*e.g.*, load balance, backup links, etc.).

Model Checking and "Automated" Configuration

Model checking has been successful in verifying the correctness of programs [49] and other network protocols [8, 63, 95]. Unfortunately, model checking is not ideal for verifying all aspects of BGP configuration because it depends heavily on exhausting the state-space within an appropriately defined environment [94]. The behavior of an AS's BGP configuration depends on routes that arrive from other ASes, some of which, such as backup paths, cannot be known in advance [27]. On the other hand, model checkers may ultimately be appropriate for *generating* BGP configuration: Recent work has proposed using a model checker to specify the parameters of a high-level routing policy for a virtual private network and running these parameters through a model checker [97]. The output of the model checker—a solution that satisfies all of the specified constraints—are the configuration fragments themselves. It is conceivable that such an approach could be used to generate BGP configuration as well.

A trend called "automated configuration" refers to techniques that allow a network operator to configure the network using templates and graphical interfaces, rather than by typing configuration commands on individual devices. Another goal of these automated configuration projects is to assist network operators in handling device heterogeneity by providing a standard interface through which all devices in the network can be configured.

Recent work has proposed automation tools that build an inventory of both intradomain routing and session-level interdomain routing configuration [41] and automate enterprise network configuration [14]. These tools detect router and session-level syntax errors only (*e.g.*, undefined filters), a subset of the faults that *rcc* detects. *rcc* is the first tool to check *network-wide properties* using a vendor-independent configuration representation and the first tool that bases its tests on a high-level specification of routing protocol correctness.

Automated configuration entails not only building an inventory of the network-wide configuration, but also enabling an operator to configure many heterogeneous network devices through a common interface. Cisco is currently leading a research initiative in this area [20]. The IETF "Netconf" working group is also actively developing a standard API over which network devices can be managed via remote procedure calls; the group is also defining a standard XML schema with which a network operator can send queries about the network device [98]. More recent work has proposed a technique for automating interdomain routing configuration by representing the network configuration in a database, configuring the network from the database itself, and automatically generating low-level routing configuration from the specification in the database [84]. Some existing firewall and virtual private network (VPN) technologies, such as products from Reef Point [116], also allow a network operator to configure network appliances from a centralized database. *rcc* also normalizes the network-wide configuration and stores it in a centralized database. In this sense, it could be seen as an element of a configura-

tion automation system that configures the network from a central location, checks the configuration for errors, and subsequently pushes that configuration to the actual devices distributed across the network.

Correctness Problems that Span Multiple ASes

The behavior of Internet routing depends on configuration that spans many independently operated networks and administrative boundaries. Internet routing configuration allows each AS to express policies independently of every other AS. Unfortunately, the policies of one AS may interact with those of another in unexpected (and unintended) ways. One serious undesirable consequence of this interaction is that the protocol may oscillate—that is, it may continue to send cycles of routing updates that do not reflect changes in the underlying topology. These oscillations (which we referred to as violations of *safety* in Chapter 1) can cause problems for both performance and debugging. Chapter 6 addresses how to guarantee safety across multiple ASes; Section 6.1 discusses related work on this topic.

■ 2.4.2 Predictability

Previous work presented an IGP emulator that helps network operators optimize link weights for intradomain traffic engineering [39]. Cariden's Multiprotocol Automation and Traffic Engineering (MATE) framework parses IGP routing configuration, estimates traffic demands, performs IGP simulation and metric optimization, and helps network operators with capacity and changeover planning, in addition to everyday traffic engineering tasks [15]. These tools incorporate traffic demand data to model how routing changes will affect traffic flow, but they do not model changes to BGP routing policies or the effects of iBGP on path selection. There has also been much focus on modeling BGP *convergence* [47, 56, 78, 125] but the work in Chapter 5 is the first to model BGP *route selection*.

OpNet's SP Guru [102] helps network operators perform traffic engineering and failure analysis within a single domain. Such tools have similar applications to the model of *interdomain* routing that we present in Chapter 5. SP Guru can model the network, given the routing configuration as input, to provide a "virtual network environment", where a network operator can evaluate potential changes to routing in the network. One of the tool's stated goals is to assist network operators in evaluating various types of configuration changes before deploying a configuration on a live network. SP Guru also integrates with NetDoctor [100] (described earlier in this section).

Network simulators (*e.g.*, ns [7], C-BGP [16], SSFNet [127]) help operators understand dynamic routing protocol behavior, but simulation represents network behavior in terms of message passing and protocol dynamics over a certain period of time. In contrast, network engineers usually just need to know the outcome of the path-selection process and not the low-level timing details. Furthermore, existing simulators do not capture some of the relevant protocol interactions that can affect the outcome of the route selection process. Simulating every detailed interaction is difficult without a higher level model of BGP in the first place.

Recent work proposes efficient techniques for large-scale parameter optimization for various network protocols, including the tuning of the local preference attribute in BGP [146]. This work is complementary to ours—the proposed search techniques could

use the routing sandbox (Chapter 5) as the "inner loop". These techniques currently use simulators such as C-BGP [16] or SSFNet [127], but the techniques only depend on the outcome of BGP path selection (not on dynamics) and would likely benefit from having an efficient, accurate emulation tool as an inner loop. An accurate model of network-wide BGP route selection can also improve the accuracy of simulators. Recent work has built a system that takes the BGP and IGP configurations from a single AS, as well as an estimate of traffic demands, and searches for the appropriate configuration changes that should be made to achieve a certain traffic engineering goal while limiting the amount of "churn" (*i.e.*, route changes) [133].

The BGP model in Chapter 5 applies several previous theoretical results in new ways. The constraints for iBGP configuration that we present in Section 5.3 are motivated by previously derived sufficient conditions for iBGP to guarantee that the routing protocols converge to a stable assignment [60, 61]. This work specified these conditions to ensure correct routing behavior, but these constraints are also required to *model* BGP routing. The route prediction algorithm in Section 5.6.2 also uses results from previous work. We apply a constructive proof regarding stable inter-AS policy configurations [47] to iBGP configuration and used this proof as the basis for predicting how a network with route reflectors will select BGP routes.

In previous work, we explored practical traffic engineering techniques in BGP; we had assumed the existence of a BGP emulator for testing traffic engineering techniques offline [32]; Chapter 5 describes such a tool that uses algorithms that *accurately* and *efficiently* predict the outcome of the BGP route selection process in a single AS using only a snapshot of the network configuration and the eBGP-learned routes from neighboring domains, without simulating protocol dynamics [37]. The evaluation of this prototype demonstrates that our algorithm is accurate and efficient enough to be used in practice for many network engineering tasks.

■ 2.5 Summary

This chapter has provided the requisite background material on Internet routing for reading the rest of this dissertation. In the process of introducing the basic operation of today's Internet routing protocol, BGP, we have drawn attention to the aspects of the protocol that, though they provide the necessary flexibility for network operations, introduce a significant amount of complexity. The rest of the dissertation focuses on proactive techniques for providing correctness and predictability guarantees in the face of this complexity.

Although this dissertation focuses on providing correctness and predictability using static configuration analysis within the context of *today's* routing protocol, the following chapters will draw out many important lessons for designing future Internet routing protocols and architectures. Chapter 7 draws on some of these lessons to propose a routing infrastructure called the Routing Control Platform (RCP) [13, 31], whose ultimate goal is to explicitly enforce various correctness guarantees.

Network operators are often their own worst enemies concerning Internet routing correctness and predictability, but malicious behavior also poses a major threat to these goals. This dissertation presents techniques for protecting network operators from themselves, but our next goal should be to protect it from adversaries. This imminent problem is not

our focus (it is difficult enough to provide correctness and predictability even when no parties are malicious!), but we discuss some open problems in Internet routing security in the final chapter (Section 7.3.5).

We are now armed with a basic understanding of Internet routing and the basic problems with guaranteeing correct and predictable behavior, but, despite all previous work, our understanding of the properties that constitute "correct" behavior are not sharp. The next chapter addresses this problem.

One of the beautiful things about baseball is that every once in a while you come into a situation where you want to, and where you have to, reach down and prove something.
- Nolan Ryan

CHAPTER 3
Correctness Specifications for Internet Routing

The flexibility offered by Internet routing configuration allows the protocol to scale well and enables operators to express a wide variety of policies, but it increases the complexity of the system. This complexity makes it difficult to reason about the behavior of Internet routing, leading to *ad hoc* fixes to observed problems that ultimately only worsen this complexity. Today, network operators (who continually tweak routing configuration) and protocol designers (who repeatedly propose "point" solutions to various problems) have no way of reasoning about whether their modifications to Internet routing will operate as intended. Worse yet, operators and designers do not even have a *specification* of properties that Internet routing should satisfy. This chapter seeks to remedy this situation by specifying correctness properties that any routing protocol should satisfy.

We introduce three properties to classify the behavior of a routing protocol. We briefly describe these properties below and explain why they are critical for correct routing.

1. *Route validity* states that if a router has a route to a destination, then a usable path corresponding to that route exists in the underlying topology. If route validity is violated, then end users could experience a failure of end-to-end connectivity, because routers could forward packets along non-existent paths.
2. *Path visibility* states that if there is a usable path between two nodes, then the routing protocol will propagate information about that path. A failure of path visibility could disrupt end-to-end communication by preventing two connected nodes from learning routes between one another.
3. *Safety* states that the routing protocol converges to a stable route assignment, regardless of the order in which routing messages are exchanged. A routing protocol that violates safety will induce persistent route oscillations, causing routing changes that are unrelated to changes in topology or policy.

This chapter defines and investigates these properties and demonstrates how they can deepen our understanding of network routing. This correctness specification addresses *static* properties of network routing, *not* dynamic behavior (*i.e.*, its response to changing

inputs, convergence time, etc.). Internet routing, like any distributed protocol, may experience periods of transient incorrectness in response to changing inputs, but we are concerned with persistent misbehavior.

Chapter 4 presents an approach to detect when two of these properties (route validity and path visibility) are violated. Chapter 5 exploits these properties to predict which of many possible routes each router in the network will select. Chapter 6 deals with the challenges of guaranteeing safety, an inherently global property.

After we introduce some basic terminology in Section 3.1, we motivate and describe the correctness properties. Sections 3.2, 3.3, and 3.4 describe route validity, path visibility, and safety, respectively, and explain how various aspects of the operation and configuration of the Internet's routing protocols can cause each of these to be violated in practice.

3.1 Preliminaries: Paths, Routes, and Policy

Before introducing the correctness properties themselves, we first introduce some basic terminology for routing. We explain these terms in the context of Internet routing and BGP. We first define paths and routes in terms of a graph $G = (V, E)$, where the nodes in $V = \{v_1, \ldots, v_N\}$ correspond to IP-level nodes (*i.e.*, routers and end hosts) and the edges in E corresponds to IP-level links between those nodes.

3.1.1 Paths and Routes

We now define two basic terms—path and route—and explain how they are related.

Definition 3.1 (Path) *A* path *is a sequence of nodes* $P = (v_0, \ldots, v_n)$*, where* $v_i \in G$ *for all* $0 \leq i \leq n$*.*

The definition of a path does not constrain how the sequence of nodes is actually constructed. As such, a path might represent a sequence of directly connected IP-layer nodes or endpoints of a tunnel.[1] Note that deleting some nodes from a path still results in a path.

In contrast to a path, a route is *information* that allows nodes in G to construct paths to *destinations*. The *destination* d may refer either to a single node or a group of nodes (named, for example, by an IP prefix). The purpose of a routing protocol is to propagate routes for destinations. Collectively, the routes to d that the nodes in G ultimately select define the *path* from any node in G to that destination. All of our definitions presume that the handle for a destination, d, cannot be manipulated (*e.g.*, our definitions do not consider the effects of IP prefix aggregation [44, 117]).

Definition 3.2 (Route) *A* route *is a mapping* $(d \rightarrow v_i)$*, where* d *is a destination, and* $v_i \in G$ *is a node en route to the destination* d*.*

We say that $v_i \in d$ if the destination d includes v_i. A route $(d \rightarrow v_i)$ received by v_j indicates that, if v_j has a packet to send to some node at destination d, it can forward that packet to v_i, which in turn ought to have a route to d (whereupon this process repeats until the

[1] A *tunnel* is a sequence of nodes that all forward packets to some intermediate node (*i.e.*, the tunnel's "exit"), rather than the ultimate destination. A tunnel may be implemented by a variety of mechanisms, such as IP-in-IP encapsulation [26], Multiprotocol Label Switching (MPLS) [22, 92], etc.

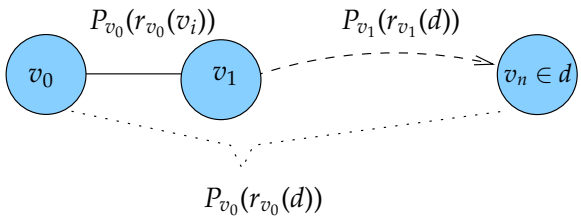


Figure 3-1: Illustration of an induced path from v_0 to d, collectively induced by the selection of a route to d at every node along the path. The node v_i may be immediately adjacent to v_0 (as shown), but it may also be several hops away.

data reaches d). One can think of a route $(d \rightarrow v_i)$ being used at node v_j as *inducing* a path, $(v_j, \ldots, v_i)$, where either v_j and v_i are directly connected or where the actual nodes along that path segment are determined by the connectivity between those nodes, as established by the IGP or using tunnels. We will formalize the notion of induced paths in Section 3.1.2.

Note that Definition 3.2 can apply to any routing protocol, not just to BGP. In an IGP, the node v_i is typically the router that is immediately connected at the IP layer. In BGP, however, (particularly in iBGP) the next hop may be several IP-layer hops away. In BGP, a node that receives a route $(d \rightarrow v)$ but is not directly connected to v must rely on the IGP for reachability to v.

■ 3.1.2 Induced Paths and Consistent Paths

We can think of paths as being *induced* by routes. That is, while there exist many sequences of nodes between any node v_0 and a node in some destination d, the path that traffic will actually take from v_0 en route to d is determined by the routes that the nodes in G select. In many routing protocols, including BGP, no single node has knowledge about the entire sequence of nodes that traffic traverses en route to d; rather, the nodes that the routes select collectively *induce* a path to d. We are interested in making statements about those induced paths. We now formalize the concept of induced paths and describe a special class of induced paths called *consistent paths*.

Definition 3.3 (Induced path) *Let $r_{v_j}(d)$ be the route that node v_j selects en route to d* (i.e., *it is a mapping $(d \rightarrow v_k)$ for some other $v_k \in G$. Then, the* path induced *by route $r_{v_0}(d) : (d \rightarrow v_i)$, $P_{v_0}(r_{v_0}(d))$, is:*

$$P_{v_0}(r_{v_0}(d)) = \begin{cases} \phi \text{ if } v_0 \text{ has no route to } d \\ v_0 \text{ if } v_0 \in d \\ (v_0, P_{v_1}(r_{v_1}(d))) \text{ otherwise} \end{cases}$$

where v_1 is defined according to $r_{v_0}(v_i) : (d \rightarrow v_1)$; that is, v_1 is the next-hop node in v_0's forwarding table for destination v_i.

Figure 3-1 illustrates an induced path and its constituent subpaths. The node v_i in the induced path may either be adjacent to v_0 in G, or it may be several hops away. When v_i is adjacent to v_0 in G, data traffic can reach v_i from v_0 via a direct IP link. When v_i is several hops away in G, however, v_0 must rely on intermediate nodes to forward traffic to v_i. In this case, the nodes between v_0 and v_i could use routes that induce paths that never

even traverse v_i. In other words, the path that is described by $(v_0, P_{v_0}(r_{v_0}(v_i)))$ may traverse an intermediate node whose induced path to d does not traverse v_i. To precisely classify the types of paths for which this inconsistency does not arise, we define the notion of a consistent path.

Definition 3.4 (Consistent path) *An induced path $P_{v_0}(r_{v_0}(d)) = (v_0, \ldots v_n)$ to d is consistent if, (1) for all $1 \leq i \leq n$, $P_{v_i}(r_{v_i}(d)) = (v_i, v_{i+1}, \ldots, v_n)$; (2) $P_{v_0}(r(v_0(d)))$ contains v_i, where $r_{v_0}(d) : d \rightarrow v_i$.*

Consistent paths are an important class of paths because inconsistent paths can sometimes give rise to forwarding loops. A *forwarding loop* is a special case of an inconsistent path where some intermediate node's induced path includes a node that has already appeared on the path.

When v_0 and v_i are not adjacent, ensuring that all intermediate nodes select a route $(d \rightarrow v_i)$ will guarantee that an induced path is consistent. If an intermediate node selects some route $(d \rightarrow v_j)$ where $v_i \neq v_j$, then the induced path to d may never traverse v_i. If, on the other hand, the intermediate node selects a route $(d \rightarrow v_i)$, then the induced path from that node to v_i will be a subpath of $P_{v_0}(r_{v_0}(v_i))$, assuming all nodes use the same function to induce paths (*e.g.*, if the induced path to v_i is based on shortest paths routing, as in an IGP).

We now briefly discuss paths and routes in the context of BGP. To illustrate the distinction between routes and paths, we examine their definitions within the context of BGP routing within a single AS. In this case, a *route* is of the form $(d \rightarrow v_i)$, where d is an IP prefix and v_i is the BGP "next hop" (a node that need not be directly connected at the IP layer). The *path* that traffic ultimately takes from some node v_j to the destination d, for which v_j has a route $(d \rightarrow v_i)$, depends on how connectivity is established between v_j and v_i. If v_j and v_i are in two different ASes, then they are typically directly connected. If they are in the same AS, however, it is common for v_i to be the IP address of an egress (or "border") router and for v_j to be several IP hops away. The induced path between v_j and v_i may be determined by a tunnel, by a shortest paths routing protocol, using static routes, etc.

If the induced path between v_j and v_i is not defined by a tunnel, then the nodes between v_j and v_i will use their own routes for forwarding data to d. In this case, the induced path to d is actually determined by "stitching together" these constituent induced paths. If all nodes between v_j and v_i select routes that indicate that traffic to d should be sent via v_i, then the induced path between v_j and v_i will be consistent. Otherwise, the path could be arbitrary; in fact, it might never traverse v_i.

Figure 3-2 shows an example that illustrates this distinction. In this example, all routers in the AS are clients of the route reflector, v_3; solid lines show the edges in the IGP graph, and all edges have a cost of 1. Suppose that v_3 learns two routes to d and selects the route that it receives from v_5. In this case, v_3 propagates that *route* (*i.e.*, $(d \rightarrow v_5)$) to all of its clients, as shown. Using that route, each node ultimately uses a different *path* to the egress router, v_5. For example, v_1's shortest IGP path to v_5 is v_1, v_3, v_5, whereas v_2's shortest path to v_5 is v_2, v_5. Even if a node, say v_1 selects a BGP route with the "next hop" v_5, there is no guarantee that the resulting *induced path* will traverse v_5. If an additional node, v_6, had been on the path between v_1 and v_3, and had instead selected a route $(d \rightarrow v_4)$, then v_1's path to d through the AS could have in fact been v_1, v_6, v_4.

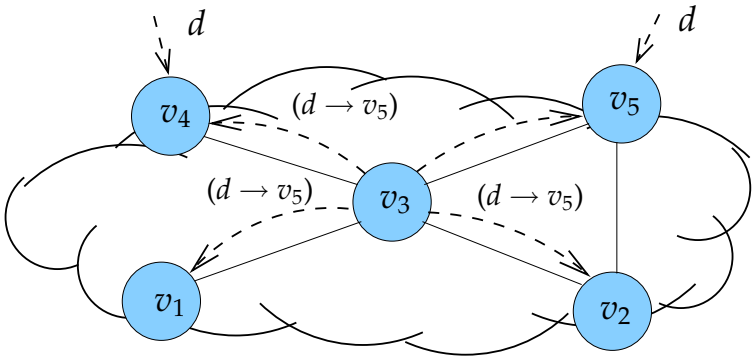


Figure 3-2: An example that demonstrates how BGP routes induce paths. Dashed lines are iBGP sessions from route reflectors to clients (*i.e.*, v_3 is a route reflector, and the rest of the routers are its clients); route reflector operation is summarized in Section 2.3.1. Dashed lines show propagation of routes. Solid lines show IGP links; in this example, all links have a cost of 1. The routes at each node *induce* paths over the IGP topology. For example, the induced path from v_2 to d is $(v_2, v_5, \ldots)$, the induced path from v_1 to d is $(v_1, v_3, v_5, \ldots)$, etc.

■ 3.1.3 Policy

A noteworthy aspect of Internet routing is that it is *policy-based*. The job of the routing protocol is not to propagate complete information about the topology, but to only propagate information about paths that comply with the various economic and policy goals of each AS. We must therefore qualify paths in the topology according to those that comply with such these policies and those that do not.

Definition 3.5 (Policy) *A* policy *is a function* $\mathcal{P}(s, v_{i-1}, v_i, v_{i+1}, d) \rightarrow (0, 1)$*, where* s *is a source,* v_{i-1}*,* v_i*, and* v_{i+1} *are nodes on a path* $(v_0, v_1, \ldots, v_n)$*,* d *is a destination, and* $\mathcal{P}$ *is defined as follows:*

$$\mathcal{P}(s, v_{i-1}, v_i, v_{i+1}, d) = \begin{cases} 1 \text{ if } i = 0 \text{ and } v_0 \text{ forwards packets from source } s \text{ destined for } d \\ 1 \text{ if } 0 < i < n \text{ and } v_i \text{ forwards packets with source } s \text{ from } v_{i-1} \\ \quad \text{destined for } d \text{ via } v_{i+1} \\ 1 \text{ if } i = n \text{ and } v_n \text{ forwards packets with source } s \text{ destined for } d \\ 0 \text{ otherwise} \end{cases}$$

The function $\mathcal{P}(v_{i-1}, v_i, v_{i+1}, d)$ is not expressive enough to capture all policies, but, as we will see, it is general enough to capture the policies that are commonly expressed in Internet routing. Other routing protocols may require more expressive policy functions. Our intent here is not to define a policy function that captures all policies, but rather to allow us to define a policy-conformant path in the context of Internet routing.

Definition 3.6 (Policy-conformant path) *A path* $(v_0, v_1, v_2, \ldots, v_n)$ *is policy-conformant for source* s *and destination* d *if* $\mathcal{P}(s, v_{i-1}, v_i, v_{i+1}, d) = 1$ *for all* $0 \leq i \leq n$*.*

For simplicity, we assume that paths for which the source, destination, and all nodes in between the source and destination are in the same AS are policy-conformant.

Although the policy function is defined at the level of nodes, it is in fact expressive enough to capture many AS-level policies that network operators commonly want to express. For example, suppose an operator wants to express that AS Y should not forward

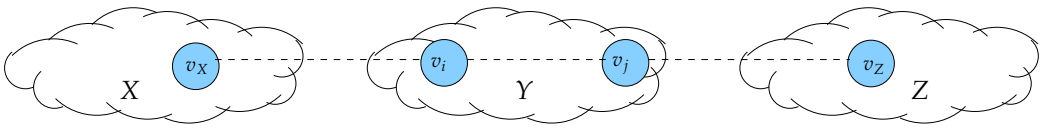


Figure 3-3: An example illustrating policy-conformant paths at the AS-level in BGP.

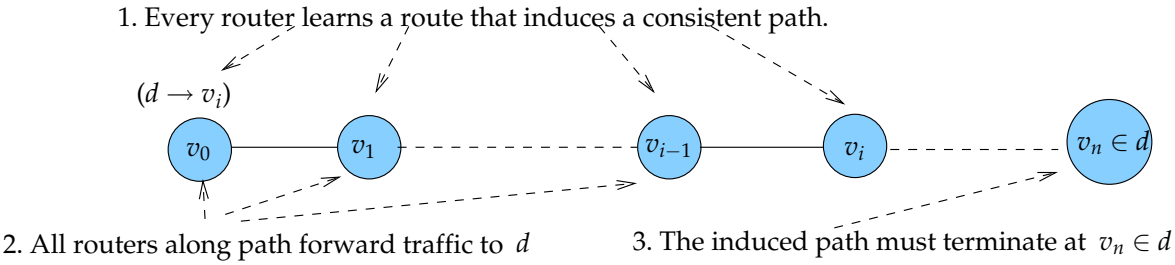


Figure 3-4: The conditions of route validity. A route is valid if it induces a consistent, policy-conformant path at every node along the path from v_0 to some $v_i \in d$.

traffic between two other ASes, X and Z, for some destination d, as pictured in Figure 3-3. Recall that a path with some nodes removed still constitutes a path. As such, it is possible to express this policy in terms of the nodes in ASes X and Z along the path. For example, in Figure 3-3, the policy can be expressed as $\mathcal{P}(s, v_X, v_i, v_Z, d) = 0$. In a more complicated scenario, if AS Y has multiple nodes that are adjacent to nodes in ASes X and Z, the AS-level policy would be expressed as an enumeration over node-level policies.

■ 3.2 Route Validity

In this section, we motivate and describe *route validity*. Informally, route validity says that any route that the routing protocol propagates should correspond to a usable path in the topology. Route validity concerns the properties of the paths induced by the routes that the routing protocol propagates.

Definition 3.7 (Route validity) *A route for a destination d is valid if, and only if, the path induced by the route (1) is consistent, (2) is policy-conformant for all sources that use the route, and (3) terminates at d. We say that a routing protocol satisfies route validity if the protocol propagates only valid routes for all destinations.*

Figure 3-4 illustrates the conditions of route validity. The first condition of route validity enforces consistency along the path between v_0 and the node v_i towards which v_0 sends traffic en route to d. Furthermore, the induced path from v_0 to v_i must be policy-conformant; that is, every node along the path $(v_0, \ldots, v_i)$ must forward traffic from its predecessor to its next hop en route to d. Verifying policy conformance is difficult for paths that traverse multiple ASes, because operators do not explicitly specify the policy function $\mathcal{P}$. The final condition says that the path induced by the route must actually terminate at some node $v_n \in d$.

Because a source v_0 and a destination d may be in different ASes, guaranteeing that BGP satisfies route validity is difficult in practice. Determining both the induced path to d and

determining whether that path is policy-conformant requires knowledge of the configurations of multiple ASes. Fortunately, the properties of route validity (*i.e.*, consistency and policy-conformance) are composable.

Observation 3.1 *Composing a path by concatenating two consistent, policy-conformant paths results in a new consistent, policy-conformant path.*

This observation implies that if the routing protocols in each AS en route to a destination induce only consistent and policy-conformant paths to some destination d, then BGP will only induce consistent, policy-conformant paths for that destination d. For the purposes of this chapter, we assume that all paths are policy-conformant, because detecting violations of policy are difficult to verify in practice; we will return to this issue in Section 4.3. We also assume that all eBGP sessions are point-to-point (*i.e.*, immediately connected at the IP layer), which is almost always the case in practice: service providers typically apply policies at AS boundaries, rather than on paths within an AS. Finally, we assume that the IGP already satisfies route validity; detecting route validity faults in internal routing protocols is beyond the scope of this dissertation.

Modulo policy-conformance, guaranteeing that BGP satisfies route validity boils down to ensuring that iBGP always induces consistent paths within each AS. Guaranteeing this property is the focus of the remainder of this section. We first focus on how to guarantee that "full mesh" iBGP configurations (and protocol modifications that would allow every router to the complete set of "best" eBGP-learned routes) always induce consistent paths; we then derive conditions on iBGP configurations that use route reflection that guarantee that iBGP only induces consistent paths.

■ 3.2.1 Case #1: Every router learns all "best" eBGP routes.

If different routers within an AS receive different sets of candidate routes for some destination d, then the routers along a path from v_0 to v_i may not ultimately select the route $(d \rightarrow v_i)$. It turns out that the default iBGP configuration, where every eBGP-speaking router has an iBGP session with every other eBGP-speaking router in the AS (*i.e.*, a "full mesh" iBGP configuration, as described in Section 2.3.1, Figure 2-5) satisfies route validity.

Theorem 3.1 *If (1) every router learns the BGP routes selected by the complete set of eBGP-speaking routers, and (2) iBGP-speaking routers do not modify route attributes* (i.e., *local preference, origin type, MED, or next hop), then all paths induced by iBGP (within the AS) will be consistent.*

Proof. If each router eventually learns of the route selected by every eBGP-speaking router, then there are two cases for any router v_0 in the AS: either (1) v_0 selects a route via a v_i in a neighboring AS, or (2) v_0 selects a route via v_i in the same AS, where v_i is an eBGP-speaking router and, hence, in turn selects a route such that v_{i+i} is in a neighboring AS. The first case corresponds to a point-to-point eBGP session; the second case corresponds to an iBGP session where the route's next hop v_i learned the route via a point-to-point eBGP session but may be multiple IP hops away. For the proof of this theorem, we are only concerned with the latter case; we rely on Observation 3.1 to ensure that iBGP induces only consistent paths in neighboring ASes.

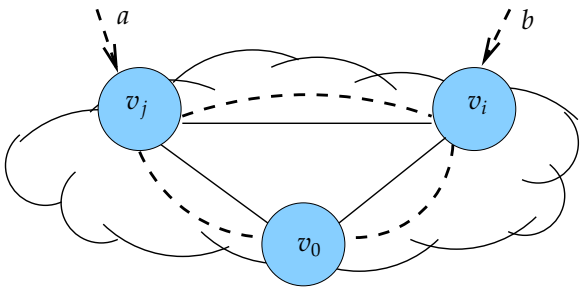


Figure 3-5: This figure illustrates the main idea of the proof of Theorem 3.1. Dashed lines represent iBGP sessions, and solid line represent IGP links. If routes a and b do not have equal local preference, AS path length, origin type, or MED, then v_0, v_i, and v_j will all select the same route. If these attributes are equal for both a and b, then v_0 selects either a or b depending on whether v_i or v_j has a shorter IGP path. If v_j selects route a and v_i selects route b, then v_0's shortest IGP path to the next hop corresponding to the chosen route must be direct.

To show that iBGP induces only consistent paths within the AS, we show that all routers on the path $(v_0, \ldots, v_i)$ select the route $(d \rightarrow v_i)$, for any $v_i \in G$. Because all eBGP-speaking routers have an iBGP session with every other router in the AS, all routers (and, in particular, all routers along the path $(v_0, \ldots, v_i)$) learn the same set of "best" routes. All of these routers would thus select a route with the highest local preference, shortest AS path length, lowest origin type, and lowest MED.

As a result, we know that all routers along the path $(v_0, \ldots, v_i)$ select *some* iBGP learned route with the shortest IGP path among candidate iBGP routes. Suppose that some router on this path, say v_j, selects a route other than $(d \rightarrow v_i)$, say $(d \rightarrow v_k)$, because $(v_j, \ldots, v_k)$ has a shorter path cost than $(v_j, \ldots, v_i)$. Then, the nodes in $(v_0, \ldots, v_k)$ *also* have a shorter IGP path cost than $(v_0, \ldots, v_i)$ and, hence, all nodes in $(v_0, \ldots, v_{k-1})$ would also select $(d \rightarrow v_k)$. ■

A full mesh iBGP configuration can guarantee the first condition of Theorem 3.1. Another approach to ensuring that every router learns the set of routes selected by the complete set of eBGP-speaking routers is to alter how route reflectors readvertise routes to their clients. By a similar argument as in the proof of Theorem 3.1, such a modification would cause iBGP to induce only consistent paths. Such a configuration not only guarantees consistent paths, but it also prevents certain types of persistent route oscillation (a pathology we examine in more detail in Section 3.4) [4]. Unfortunately, implementing such a configuration in practice requires modifying the large deployed base of BGP routers. Alternatively, an architecture such as either the Routing Control Platform (RCP) [13, 31] or the recent proposal for more versatile route reflectors [9] could implement this type of iBGP configuration.

■ 3.2.2 Case #2: Each router learns only some "best" eBGP routes

If full mesh iBGP were the only intra-AS BGP configuration, guaranteeing that iBGP satisfied route validity would be easy. Unfortunately, as discussed in Section 2.3.1, this technique does not scale to large ASes because it requires $O(|R|^2)$ iBGP sessions, where $|R|$ is

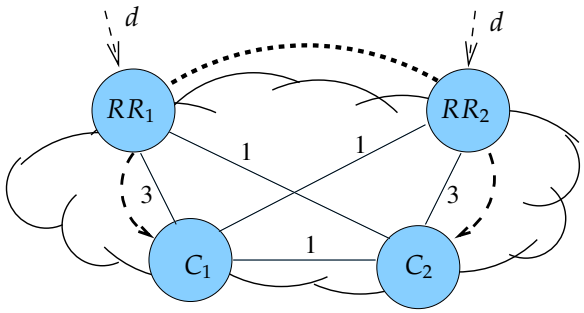


Figure 3-6: The interaction of IGP and route reflection in iBGP may cause route validity violations resulting in forwarding loops [23]. Note that this topology satisfies path visibility but not route validity. Dashed lines represent iBGP sessions; a directed edge indicates an iBGP sessions from a route reflector to its client.

the number of routers in the AS. Large ASes typically use a technique called route reflection, where a single route reflector selects a route on behalf of its client routers.

Guaranteeing route validity in an iBGP topology with route reflectors is not easy. Previous work has observed that the interactions between the IGP topology and an iBGP topology with route reflectors can give rise to route validity violations [23]. Figure 3-6 shows one such example. Route reflectors RR_1 and RR_2 both receive a route to destination d and have clients C_1 and C_2 respectively. Hence, C_1 may receive and select the route $(d \rightarrow RR_1)$, and C_2 may receive and select the route $(d \rightarrow RR_2)$. If the shortest IGP path (*i.e.*, the induced path) between A and RR_1 is via B, and the shortest IGP path between B and RR_2 is via A, then traffic en route to d that traverses either router A or B will be caught in a persistent *forwarding loop*: that is, traffic destined for d will never reach d but instead will repeatedly visit a cycle of two or more nodes. A forwarding loop is simply a special case of a route validity violation.

Our goal is to detect whether a configuration of route reflectors and clients induces only consistent paths with a simple algorithm that examines only the static iBGP and IGP topologies. One such sufficient condition that guarantees this property requires that the iBGP topology be *RR-IGP-Consistent*, defined as follows:

Definition 3.8 (RR-IGP-Consistent) *A route reflector configuration is* RR-IGP-Consistent *if, for all nodes, every shortest IGP path between that node and its possible egress nodes (*i.e.*, the set of eBGP-speaking routers) traverses that node's route reflector before any other node's route reflector and the egress node's route reflector before the egress node itself.*

In previous work, Dube suggested placing route reflectors on the shortest IGP path to their clients [23]. We now prove that this condition guarantees that the iBGP configuration will only induce consistent paths.

Theorem 3.2 *If an iBGP configuration is* RR-IGP-Consistent*, then all paths induced by iBGP are consistent.*

Proof. Suppose not. Then, there must exist a destination d and a path $P = (v_0, \ldots, v_n)$ for which some node v_j between v_0 and v_n selects a route $(d \rightarrow v_i)$, where $i \neq n$. There are two

cases: (1) v_j is on the path from v_0 to the route reflector of v_0; and (2) v_j is on the path from the route reflector of v_0 to v_n.

In the first case, if v_0 and v_j select different next hops then, by definition, they must be clients of different route reflectors. By definition, then, the iBGP topology is not *RR-IGP-Consistent*. The second case reduces to a similar argument as in Theorem 3.1: if v_j selects a route with a next hop other than v_n, then the route reflector of v_0 would have also learned that route and selected it (otherwise, v_j would not have been on the route reflector's shortest path to v_n, by the same argument as in Theorem 3.1). ■

Although this result is a helpful sufficient condition, it does not guarantee that route validity will be satisfied when arbitrary links fail, thus causing shortest IGP paths to change. Designing an *RR-IGP-Consistent* iBGP topology that is robust to link failures is a difficult task. Recent work has proposed using graph separators as a way of efficiently placing route reflectors in an iBGP topology to guarantee that route validity is satisfied [139].

■ 3.3 Path Visibility

Path visibility says that if there exists one or more policy-conformant paths between two nodes, then the routing protocol should propagate routes that induce at least one of those paths. Path visibility is an important property for a number of reasons. First, if a routing protocol satisfies this property, then every node is guaranteed to have the necessary information to reach all other nodes.

Definition 3.9 (Path visibility) *A routing protocol satisfies* path visibility *if, for all $v_0 \in V$ and for all destinations d, the existence of a policy-conformant path $P = (v_0, \ldots, v_n)$ implies that v_0 learns a valid route $(d \rightarrow v_j)$ for some $0 \leq j \leq n$.*

Path visibility states that if there is a policy-conformant path from v_0 to d, then v_0 should learn *at least one* valid route to d. Note that the definition does not require v_0 to learn all routes to d, nor does it require that v_0 learn the "best" route to d by any metric. Path visibility also does not require that the route that v_0 learns correspond to the actual path that traffic takes from v_0 to d.

By definition, path visibility violations result when some router fails to propagate usable routes. These failures in route propagation range from the mundane (*e.g.*, misconfigured filters that fail to install or advertise routes for a policy-conformant path) to the subtle (*e.g.*, errors in iBGP configuration).

Because of the way iBGP readvertises routes, an arbitrary iBGP configuration is not guaranteed to satisfy path visibility. In fact, even the very simple iBGP topology in Figure 3-7 does not satisfy path visibility: if the route reflector RR_1 (or its client, C_1) receives a route for some destination via an eBGP session, then neither RR_3 nor C_3 will receive a route to the destination, and vice versa.

Path visibility is important because it ensures that, if the network remains connected at lower layers, the routing protocol does not create any new network partitions. Path visibility also reduces the likelihood of suboptimal routing. For example, in Figure 3-7, even if all clients learned *some* route to the destination via eBGP, the clients would not be guaranteed to discover the *best* route to the destination (*e.g.*, if a client of the route reflector

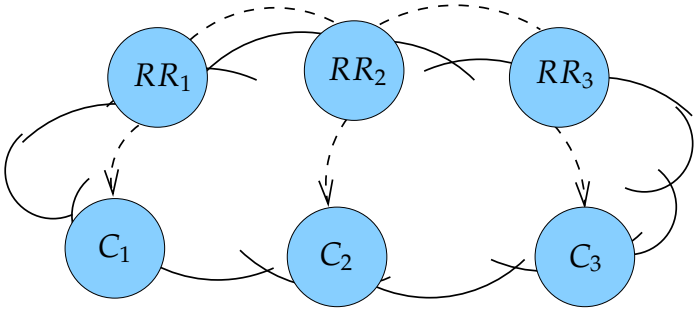


Figure 3-7: A simple iBGP topology that violates path visibility. Routes learned via eBGP at RR_1 or C_1 will not be propagated to RR_3 or C_3 (and vice versa).

on the far left learned a route with a shorter AS path, neither the route reflector on the far right nor its clients would learn it). As such, it is important that an AS's iBGP configuration satisfy path visibility. In the remainder of this section, we derive the constraints on the iBGP configuration that must be satisfied to guarantee path visibility. We first consider iBGP topologies that do not employ route reflection.

Theorem 3.3 *For an iBGP topology without route reflectors, satisfying path visibility requires a full mesh iBGP configuration.*

Proof. Consider a router v_i, which learns a route r for some destination d via eBGP, and a router v_0 within the same AS that does not have an iBGP session to v_i. Then, v_i will readvertise r to the routers to which it has iBGP sessions, but none of those routers will advertise the route to v_0, because they all learned the route via iBGP. ■

In large networks, a route reflector may itself be a client of another route reflector. Any router may also have "normal" (*i.e.*, peer) iBGP sessions with other routers. We use the set of reflector-client relationships between routers in an AS to define a graph I, where each router is a node and each session is either a directed or undirected edge: a client-reflector session is a directed edge from client to reflector, and peer iBGP sessions are undirected edges. We say that I is *acyclic* if I has no sequence of directed and undirected edges that form a cycle. In typical iBGP hierarchy designs, I is acyclic (previous work states that I should be acyclic to prevent protocol oscillations [59]—and it is a good design decision anyway—although we will see in Section 3.4 that this constraint is unnecessary). We now define the topological constraints on I to guarantee path visibility.

Theorem 3.4 *Suppose that the graph defined by an AS's iBGP relationships, I, is acyclic. Then, I does not have a signaling partition if, and only if, the eBGP-speaking routers that are not route reflector clients form a clique.*

Proof. Call the set of routers that are not reflector clients the "top layer" of I. If the top layer is not a clique, then there are two routers with no iBGP session between them, such that no route learned via eBGP at RR_i will ever be disseminated to RR_j, since no router

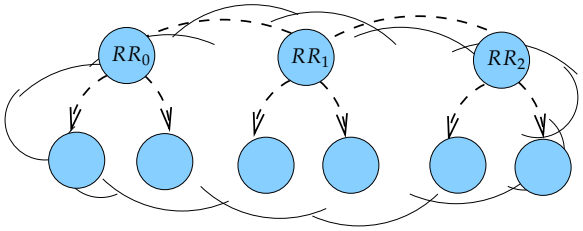


Figure 3-8: Illustrating the main idea of the proof of Theorem 3.4.

readvertises an iBGP-learned route (*e.g.*, RR_0 and RR_2 in Figure 3-8), and vice versa. Furthermore, no route that is learned via eBGP at any of RR_i's clients will be disseminated to RR_j or RR_j's clients, and vice versa.

Conversely, suppose the top layer is a clique. Observe that if a route reflector has a route to the destination, then all of its clients have a route as well. Thus, if every router in the top layer has a route, all routers in the AS will have a route. If any router in the top layer learns a route through eBGP, then all the top layer routers will hear of the route (because the top layer is a clique). Alternatively, if no router at the top layer hears an eBGP-learned route, but some other router in the AS does, then that route propagates up a chain of route reflectors (each client sends it to its reflector, and the reflector sends it on all its iBGP sessions) to the top layer, from there to all the other top-layer routers, and from there to the other routers in the AS. ■

The results in this section suggest that path visibility can be guaranteed by checking relatively simple constraints on the iBGP topology, which can be determined by analyzing the static configuration files alone. Although, in the long term, architectural changes could be made to guarantee that no configuration ever violates path visibility [9, 13, 31], relatively simple checks against routing configuration can guarantee path visibility today (as we will see in Chapter 4).

■ 3.4 Safety

Violations of safety can cause the routing protocol to continually send routing updates that do not reflect changes in the underlying topology. We provide an informal definition of safety and defer a formal definition to Chapter 6 (Definition 6.10).

Definition 3.10 (Safety) *A routing protocol satisfies* safety *if and only if, given no changes to available paths after time t_0, then, at some finite time $t_s > t_0$, each node $v \in G$ selects some route r and does not select a route $r' \neq r$ for any $t > t_s$.*

Safety is an important property because it guarantees that changes in *routes* (*i.e.*, routing update messages) correspond directly to changes in available *paths*. This invariant is important for several reasons. First, if the routing protocol causes routers to change routes unnecessarily (*i.e.*, when the paths are in fact stable), the protocol itself may cause performance degradations, such as lost or reordered packets. Second, if routing changes do not correspond to changes in the actual topology, then debugging the cause of an oscillation

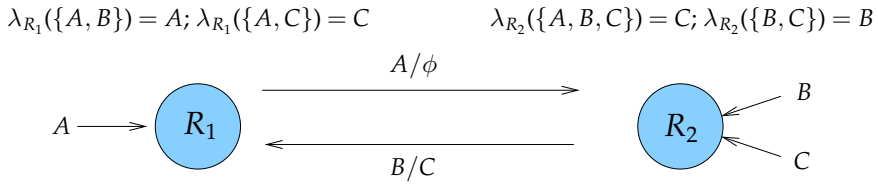


Figure 3-9: λ_{R_2} does not satisfy determinism. This violation can causes a safety violation.

becomes more difficult, because an operator cannot determine whether routing changes reflect problems with infrastructure (*e.g.*, flaky or failing equipment) or the routing protocol itself.

Safety problems arise for two reasons:

1. Conflicting route selections within the same AS, caused by interactions between BGP route attributes and the IGP (iBGP safety).
2. Conflicting rankings, caused by conflicting policies between ASes (eBGP safety).

In both cases, guaranteeing safety is hard. The rest of this section focuses on the constraints for guaranteeing safety in iBGP. Guaranteeing that eBGP satisfies safety requires either knowing the rankings of ASes across the *global* Internet (not a realistic requirement, because ASes typically insist on keeping their rankings private) or placing restrictions on how each AS can specify rankings and filters. This problem is the focus of Chapter 6.

Safety violations in iBGP occur because BGP's route selection process (as described in Table 2-2, Section 2.2.2) does not satisfy *determinism*. Determinism essentially says that the route each router ultimately selects should not depend on either (1) the presence or absence of routes that would not be selected in the first place or (2) the order in which messages arrive. We formally define determinism and explain why guaranteeing this property is difficult in practice.

Definition 3.11 (Selection function) *A selection function at router r, λ_r, takes as input a set of routes $R_d = \{r_1, \ldots, r_n\}$ for some destination d and produces a single route $r_i \in R_d$. The route r_i is often called the router's "best route" to d.*

Definition 3.12 (Determinism) *A routing protocol satisfies* determinism *for destination d if, for all routers r, if r has a set of routes R_d to d, $\lambda_r(R_d)$ satisfies the following two properties:*

1. *$\lambda_r(R_d) = \lambda_r(R'_d)$, where R'_d is any subset of R_d that contains $\lambda_r(R_d)$, and*
2. *$\lambda_r(R_d)$ does not depend on the order in which the routes in R_d arrived at router r.*

Determinism depends only on the selection function, λ_r, for all routers r. Thus, we may also discuss a single selection function, λ_r, in terms of whether it satisfies determinism.

Determinism is important for predictability; moreover, as the following observation shows, violations of determinism can induce safety violations, even when the selection function of only one router violates determinism.

Observation 3.2 *If the selection function of even one router, λ_r, violates determinism, then the routing protocol may also violate safety.*

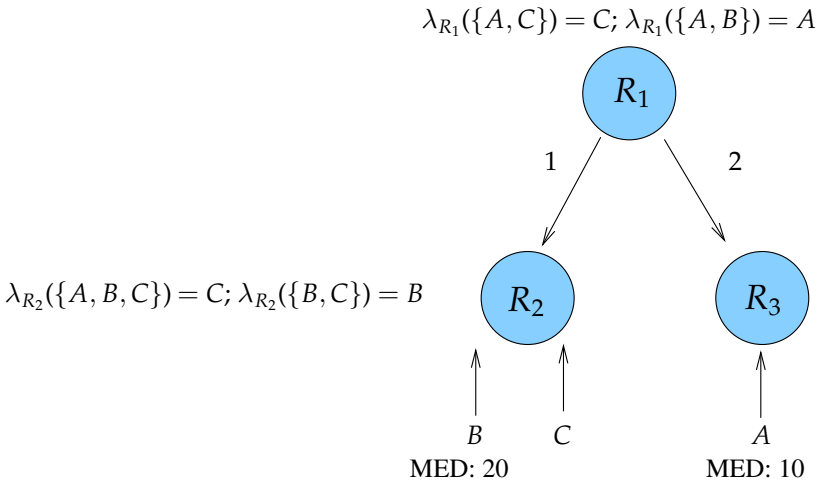


Figure 3-10: Instantiation of Figure 3-9 in a BGP configuration. Router 1 is a route reflector with two clients, R_2 and R_3. Costs on edges are IGP path costs. Router R_2 prefers route B over route C due to a tiebreak.

The following example illustrates this point.

Example 3.1 Consider Figure 3-9. Router R_1 selects route A given the choices $\{A, B\}$ and selects route C given choices $\{A, C\}$. This selection function satisfies determinism. On the other hand, router R_2's selection function violates determinism: $\lambda_{R_2}(\{A, B, C\}) = C$, but $\lambda_{R_2}(\{B, C\}) = B$. The interaction of the two selection functions creates the following oscillation:

1. Router R_1 receives only route A, selects it, and advertises this route to router R_2.
2. Router R_2 has received $\{A, B, C\}$, selects route C, and advertises it to router R_1.
3. Router R_1 has received $\{A, C\}$, selects route C, and sends a withdrawal (ϕ) for route A to router R_2.
4. Router R_2 selects B from the set $\{B, C\}$ and advertises it to router R_1, implicitly withdrawing route C.
5. Router R_1 now has to select a route from the set $\{A, B\}$, selects route A, and advertises it to router R_2.

This process repeats forever, violating safety. ■

■ 3.4.1 Determinism Violations in BGP: The MED Attribute

It turns out that the above scenario can occur in BGP, because the MED attribute causes each router's selection function to violate determinism. The addition of a third router, as shown in Figure 3-10, gives rise to the oscillation from the previous example. In this case, router R_1 is a route reflector for two clients: routers R_2 and R_3, with IGP costs as shown. Routes A and B are advertised by the same AS, and route A has a lower MED value (and, hence, is preferred to B). In this setup, the selection functions are exactly as described in the from Figure 3-9: when router R_2 learns $\{A, B, C\}$, route B is eliminated due to MED, and route C is selected because it is an eBGP-learned route. When router R_2 learns only $\{B, C\}$, on the other hand, it prefers route B over route C due to the router ID tiebreak. Similarly, router R_1 prefers route C over route A due to IGP, but router A over router B

due to MED. The routing system in this example oscillates in the same fashion as the one shown in Figure 3-9.

As the above example shows, the interaction between the MED attribute and route reflection can cause BGP to violate safety. Note that this example satisfies the guidelines that were specifically proposed to avoid these types of oscillations [89]. Even though not all safety violations are caused by violations of determinism, eliminating BGP's determinism problem can eliminate all oscillations that do not involve cyclic preferences over routes caused by setting the local preference attribute. Specifically, by making the MED attribute comparable across *all* routes, rather than just those from the same AS, each router's selection function can be made to satisfy determinism. We now formally show this result.

Lemma 3.1 *If a router's selection function compares the MED attribute across all routes to a destination (rather than just across those from the same neighboring AS), then its selection function satisfies determinism.*

Proof. We must show that if the router's selection function compares the MED attribute across all routes to a destination then: (1) the route it selects does not change when any route is removed from that set; and (2) the route it selects does not depend on the order in which the router receives them.

When a router compares the MED attribute across all routes to a destination, then all routes to a destination can be totally ordered. Specifically, all routes can be sorted by local preference. Each set of routes with equal local preference can be sorted from shortest AS path length to longest, and so forth. Thus, the set of routes to a destination can be totally ordered, and removing a route from that set that is not the most preferred in the total ordering will not change the most preferred route, since that route must have had a lower local preference, longer AS path length, higher origin type, lower MED, etc.

We must also show that the route a router r selects, $\lambda_r(R_d)$, does not depend on the order that r receives the routes in R_d. We know that the routes in R_d are totally ordered, which means that there is a preference relation between any two routes ρ_i and ρ_j that is consistent for *any* subset of R_d that contains both ρ_i and ρ_j. We also know that $\lambda_r(R_d)$ will ultimately select the route that is most preferred in that total ordering. Suppose that most preferred route is ρ_i. When ρ_i arrives, r will select ρ_i and continue to select it even after other routes arrive. Thus, if ρ_j arrives before ρ_i, then the router will not continue to select ρ_j after ρ_i arrives, since ρ_i is strictly better than ρ_j in the total ordering. Similarly, if ρ_j arrives after ρ_i, then r will continue to select ρ_i, since it is better than ρ_j in the total ordering. ■

We explore how comparing the MED attribute across all routes affects protocol operation, as well as how this might be done in practice, in Section 5.9.1. In short, the primary benefit of making the route selection function deterministic is that a set of routers *within a single AS* may violate safety if determinism is not satisfied. Although determinism prevents safety violations such as those shown in Figures 3-9 and 3-10, it does not prevent *all* violations of safety. For that, we require a stronger notion of determinism, which we call *egress determinism*.

Figure 3-11: The relationship between determinism, egress determinism, and safety.

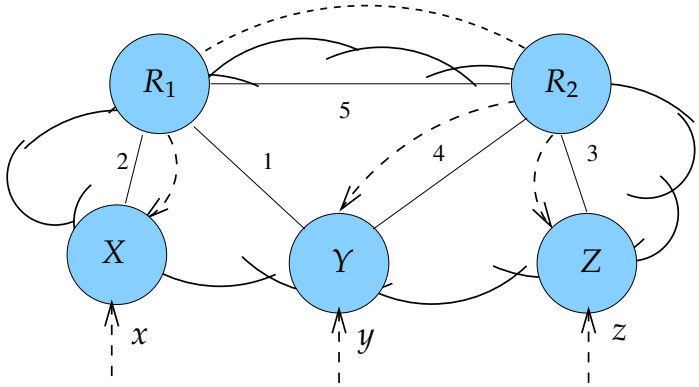


Figure 3-12: The interaction of IGP and iBGP can cause a violation of egress determinism. λ_{R_1} is equal to either x or y depending on whether $z \in E_d$.

■ 3.4.2 Egress Determinism Violations in BGP: Route Reflection

Even if determinism is satisfied, an AS's iBGP topology can still cause a routing protocol to violate safety. In particular, we can construct an oscillation that involves the interaction between an AS's route reflector topology and its IGP topology. To better understand this interaction, we first define the notion of *egress determinism*. Egress determinism is a stronger condition than determinism, as shown in Figure 3-11; essentially, it states that, given a set of routes learned at *any* egress router in the AS, a router's preference between any pair of those routes should not depend on either the order in which those routes arrive or the presence or absence of other routes. Egress determinism implies determinism, but it also states that every router's selection function should satisfy determinism for all routes learned at *any* router in the AS, not just those learned locally at that router.

Definition 3.13 (Egress Determinism) *Let E_d be the set of routes for destination d learned at* any *router in the AS. Then, a routing protocol satisfies* egress determinism *for destination d if $\lambda_r(E_d)$ satisfies the following two properties:*

1. *$\lambda_r(E_d) = \lambda_r(E'_d)$, where E'_d is any subset of E_d that contains $\lambda_r(E_d)$, and*
2. *$\lambda_r(E_d)$ does not depend on the order in which the routes in E_d arrived at router r.*

Note that egress determinism is a stronger condition than determinism because it states properties that λ_r must satisfy over the set of routes learned by all routers in the AS, not just the routes learned at r.

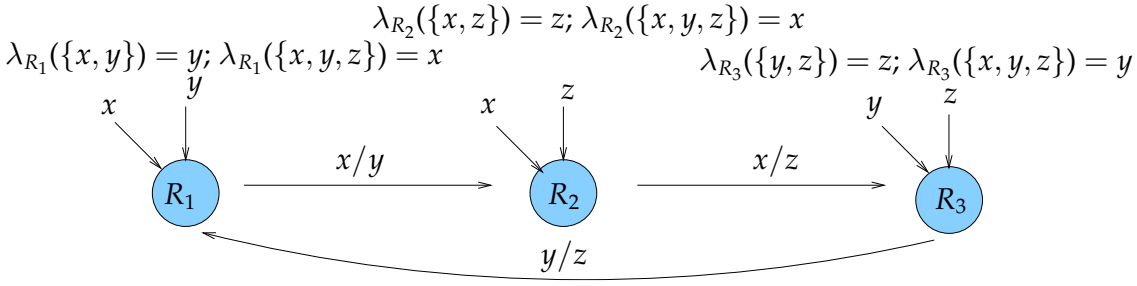


Figure 3-13: Violations of egress determinism can also cause the routing protocol to violate safety.

If every router in the AS always learned all routes in E_d, then violations of egress determinism would never cause oscillations: given a fixed set of routes E_d, every router would always see that set and select the same route. In an iBGP topology with route reflectors, however, most routers will see some subset of E_d, which means that violations of egress determinism may cause safety violations. Consider Figure 3-12: X is a route reflector client of R_1, and Y and Z are clients of route reflector R_2. Suppose that routers X, Y, and Z all learn routes for some destination d with equal local preference, AS path length, origin type, and MED attributes, causing routers within the pictured AS to resort to preferring eBGP routes over iBGP routes, and, that being equal, to prefer routes with the shortest IGP path cost. If $E_d = \{x, y, z\}$, then $\lambda_{R_1}(E_d) = x$: R_2 selects z due to its shorter IGP path cost to the next hop, and R_1, having learned x and z, selects route x. If, on the other hand, $E_d = \{x, y\}$, then $\lambda_{R_1}(E_d) = y$: R_2 selects y, and R_1, having learned both x and y, selects y due to the shorter IGP path cost. Thus, the first condition of egress determinism is violated.

Like determinism violations, egress determinism violations can cause the routing protocol to violate safety. Consider three routers whose selection functions violate egress determinism, as shown in Figure 3-13; R_1's selection function is identical to that in Figure 3-12. Each router prefers one route or the other depending on the presence or absence of a third route. In this case, there is no stable assignment of routes x, y, and z to routers R_1, R_2, and R_3. For example, if R_1 selects x, then R_2 selects z and R_3 selects y, prompting R_1 to select y, and so on. This very scenario can be realized in BGP today if three routers' route selection functions fail to satisfy egress determinism, as shown in Figure 3-14.

Lemma 3.2 *If an AS's iBGP topology is* RR-IGP-Consistent, *and every router's selection function satisfies determinism, then every router's selection function also satisfies egress determinism.*

Proof. Suppose that there exists some router R_1 and routes x, y, and z (not necessarily learned via eBGP at R_1) such that $\lambda_{R_1}(\{x, y, z\}) = y$, but $\lambda_{R_1}(\{x, y\}) = x$. First, because λ_{R_1} satisfies determinism, it must be the case that (1) R_1 learns x via iBGP and (2) some router R_2 in the iBGP topology withdraws the eBGP-learned route x from router R_1 upon learning the route z, thereby preventing router R_1 from receiving route x (if R_1 had learned x directly via eBGP, it would continue to select route x). Second, R_2 must have selected z over x because it had a shorter IGP path to the router from which it learned z—if x had been less desirable based on some other attribute (*e.g.*, AS path length), then r would have also selected the route z upon learning it from R_2, rather than selecting y instead. Third, because R_1 selects y after learning z from R_2, its IGP path to the egress router that learned

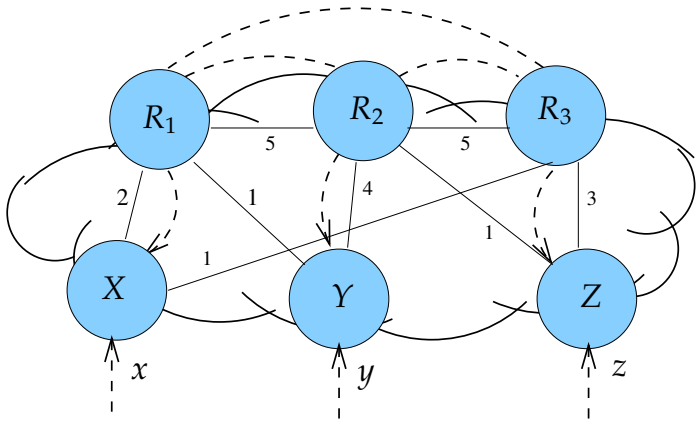


Figure 3-14: The interaction of IGP and iBGP can cause a violation of egress determinism that induces a safety violation. This figure shows the instantiation of Figure 3-13 in BGP. Previous work has also observed that violations of this type could occur [61] but did not observe that these could be constructed in general by composing egress determinism violations.

y must be shorter than its IGP path to the router that learned z, yet longer than its IGP path to the router that learned x. This relation among distances to egresses is only possible if the shortest path between R_1 and either the egress router that learned route x or z does not traverse R_2 (*i.e.*, the iBGP topology is not *RR-IGP-Consistent*). Figure 3-15 shows why this relation is not possible in an iBGP topology that is *RR-IGP-Consistent*. ■

We now state the conditions for iBGP to satisfy safety using our results involving determinism and egress determinism. Specifically, we show that if MED is compared across all routes (*i.e.*, every router's selection function satisfies determinism) and if the iBGP topology is *RR-IGP-Consistent* (*i.e.*, egress determinism is satisfied), then iBGP satisfies safety.

Theorem 3.5 *If every router's selection function compares MED attribute across all routes and the iBGP topology is* RR-IGP-Consistent*, then iBGP satisfies safety.*

Proof. The proof follows from Lemmas 3.1 and 3.2. If the conditions of the theorem hold, then BGP satisfies both determinism and egress determinism. It remains to show that no iBGP topology can violate safety if it satisfies both determinism and egress determinism.

We must show that, for any route that a router ultimately selects, other routers in the AS will not select routes that ultimately causes the original router to change the route it selects. For simplicity, we will consider an iBGP route reflector hierarchy with one level. The argument can be extended to a multiple-level hierarchy, and to a network with multiple top-level routers or multiple clients per route reflector, without loss of generality. Consider the possible propagation of BGP routes between routers C_1 and C_2 and their route reflectors R_1 and R_2, as shown in Figure 3-16. Suppose C_1 and C_2 initially select routes via eBGP; each will readvertise these routes to R_1 and R_2. Then, there are three cases:

1. *R_1 prefers the route through its client C_1.* In this case, R_1's selection of the route through C_1 will obviously not cause either R_1 or C_1 to switch paths.

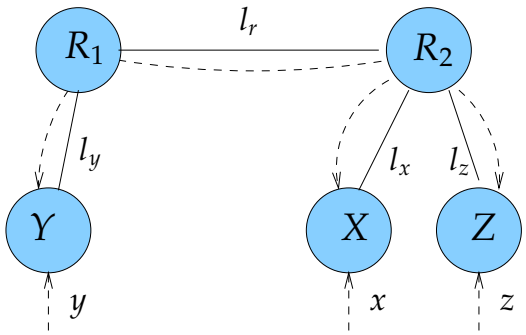


Figure 3-15: The main idea of the proof of Lemma 3.2. If R_2 prefers z over x, then $l_z \leq l_x$. If R_1 prefers x, given routes x, y, and z, then $l_y \geq l_r + l_x$. If R_2 is on the shortest IGP path between R_1 and both X and Z, it follows that $l_y \geq l_r + l_z$ (and, hence, R_1 would *not* have selected route y). Thus, for R_1 to prefer route y when it learns routes x and z, its shortest path to either egress router Y or Z must not traverse R_2. (The links labeled with l_x, l_y, and l_z are shown as single IGP links, but the argument generalizes to IGP paths.)

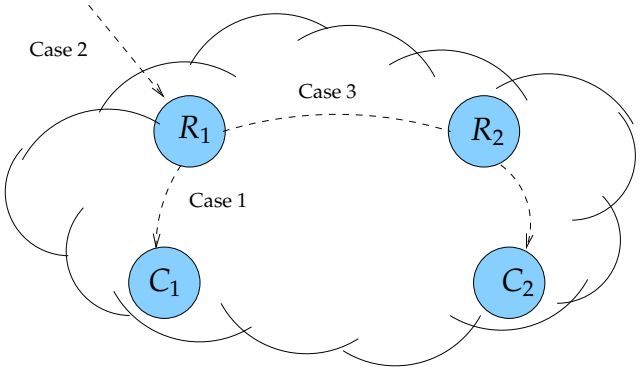


Figure 3-16: The three cases in the proof of Theorem 3.5.

2. *R_1 prefers its own eBGP-learned route (analogously for R_2).* In this case, C_1 learns a route via R_1. If it continues to prefer its own route, neither router will select a new route. If, on the other hand, it prefers the route through R_1, then it will withdraw its eBGP-learned route from R_1. However, since λ_{R_1} satisfies determinism, R_1 will continue to select its own eBGP-learned route when it receives this withdrawal.

3. *R_1 prefers a route via R_2 or C_2 (analogously for R_2).* A similar argument can be used to show that neither R_1 nor C_1 will select a new route once R_1 selects a route via C_1. For the third case, we must also show that R_1's withdrawal of either its own route or the route via C_1 from R_2 will cause neither R_2 nor C_2 to change routes. Since R_1 prefers a route via R_2 then R_2 selected its own route or the route via C_2; egress determinism guarantees that R_1's withdrawal will not cause either C_2 or R_2 to change its route selection.

■

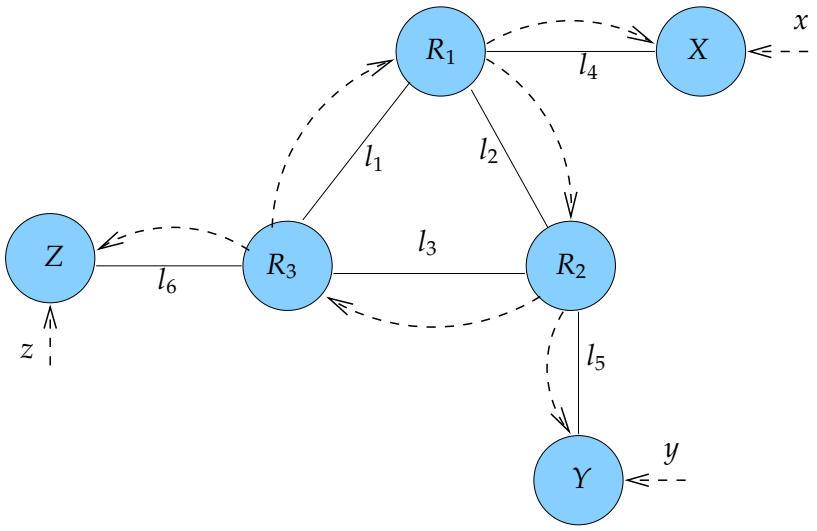


Figure 3-17: When iBGP violates safety but satisfies egress determinism, the only way a cyclic iBGP topology can violate safety is if the IGP allows negative edge weights. This example shows an iBGP hierarchy that includes only six routers, but it generalizes: R_i could be any cyclic relationship at the top of the hierarchy, X could be a path through a sequence of iBGP sessions to the egress router X (*e.g.*, a sequence of route reflector-client sessions) and l_4 could be the cost of that path, and so forth.

Our definitions have allowed us to derive sufficient conditions on safety that are significantly weaker (and therefore, the result is stronger) than in previous work [59]. In particular, *our results show that assuming that the relationships between route reflectors and their clients are acyclic is unnecessary* (although a cyclic topology may make an oscillation more likely if egress determinism is violated). It turns out that the only way for a cyclic iBGP topology to cause oscillations would be for either the iBGP topology to not be *RR-IGP-Consistent* or for some IGP edges to have negative edge weights.

To understand why cycles in the iBGP hierarchy do not cause problems if the topology is *RR-IGP-Consistent*, see Figure 3-17. In this example, if egress determinism is satisfied, then the only case where an oscillation might result is where R_1 prefers route y over route x, R_2 prefers route z over y, and R_3 prefers x over z. All other cases where oscillation might occur (*i.e.*, those caused by violations of egress determinism) require some shortest IGP path between a router and another egress to not traverse that router's route reflector. For safety to be violated in this example, routes x, y, and z must all have equal local preference, AS path length, origin type and MED (otherwise, all routers would select the most preferable route or routes). Presuming that all routes are equally good up to the step in route selection involving the IGP tiebreak, then the only way for such a situation to occur is if the following inequalities were satisfied:

$$\begin{aligned} l_1 + l_4 &< l_6 \\ l_2 + l_5 &< l_4 \\ l_3 + l_6 &< l_5 \end{aligned}$$

which implies that $l_1 + l_2 + l_3 < 0$, or that some IGP edge weights must be negative.

Theorem 3.5 is significant because the conditions on the iBGP topology that are required to guarantee safety are identical those for guaranteeing route validity, as stated in Theorem 3.2. Furthermore, because there are now known techniques for *generating* these configurations [139], our results are prescriptive, since this technique that was designed to generate iBGP topologies that guarantee route validity also happens to generate topologies that guarantee safety.

This section has described safety violations that result from protocol interactions *within a single AS*. However, safety is a somewhat unique property because it inherently depends on how different ASes are allowed to rank candidate routes to a destination (*i.e.*, Observation 3.1 is not satisfied). Whereas, in iBGP, a router's rankings are constrained by the underlying IGP topology, in eBGP, an AS's rankings can be arbitrary. Verifying a property that involves analyzing the interactions between the configurations of multiple ASes is challenging: because these ASes compete with one another, no single AS has knowledge of the configurations of other ASes. Chapter 6 explores how safety can be guaranteed *across multiple ASes* without requiring each AS to expose its configurations to other ASes.

■ 3.5 Summary

Detecting problems in Internet routing requires a precise specification of correct behavior and a framework for reasoning about whether that specification is satisfied. In this chapter, we presented a three-part correctness specification: route validity, path visibility, and safety. We explained why each of these properties is important for the fundamental operation of Internet routing and reasoned about how these properties may be satisfied or violated in the context of BGP.

In the subsequent chapters of this dissertation, we will use the specification constraints to guide the operations and design of correct Internet routing. In Chapter 4, we will explore how static configuration analysis can be used to guarantee that two of these properties—route validity and path visibility—hold. Theorems 3.1–3.4 suggest invariants on configuration that could be verified by detecting when the configuration violates certain invariants, and both Theorem 3.5 and the discussion at the end of Section 3.2.1 suggest possible protocol modifications. In Chapter 5, we will exploit the fact that, when the routing protocol satisfies certain aspects of this specification (specifically, path visibility, safety, and, in some cases, determinism), we can design algorithms that efficiently predict the routes that each router in an AS will ultimately select. Chapter 6 tackles the problem of guaranteeing safety in the context of eBGP, which is challenging since it inherently involves dependencies between the policies of multiple independent (and often competing) ASes.

Things could be worse.
Suppose your errors were counted and published every day,
like those of a baseball player.
- Unknown

CHAPTER 4
rcc: Detecting BGP Configuration Faults with Static Analysis

This chapter describes the design, implementation, and evaluation of ***rcc***, the router configuration checker, a tool that uses static analysis to detect faults in Border Gateway Protocol (BGP) configuration. By finding faults over a distributed set of router configurations, *rcc* enables network operators to test and debug configurations before deploying them in an operational network. This approach improves on the status quo of "stimulus-response" debugging where operators need to run configurations in an operational network before finding faults.

As described in Chapter 2 (Section 2.3), network operators use router configurations to provide reachability, express routing policy (*e.g.*, transit and peering relationships [99], inbound and outbound routes [6], etc.), configure primary and backup links [46], and perform traffic engineering across multiple links [37]. The complex process of configuring routers is exacerbated by the number of lines of code, by configuration being distributed across the routers in the network, by the absence of useful high-level primitives in today's configuration languages, by the diversity in vendor-specific configuration languages, and by the number of ways in which the same high-level functionality can be expressed in a configuration language.

Router configuration gives network operators the flexibility to control traffic and implement complex business relationships, but its complexity also means that router configurations are prone to faults [6, 85]. Configuration faults include invalid routes (including hijacked and leaked routes); contract violations [35]; unstable routes [78]; routing loops [23, 29]; and persistently oscillating routes [4, 58, 136]. As summarized in Table 2-4 (Section 2.4.1), BGP configuration faults can seriously affect end-to-end Internet connectivity, leading to lost packets, forwarding loops, and unintended paths between hosts or ASes.

To understand the extent to which this complex configuration is responsible for the types of failures that occur in practice, we studied the archives of the North American Network Operators Group (NANOG) mailing list, where network operators report operational problems, discuss operational issues, etc. [96]. Because the list has received about 75,000 emails over the course of ten years, we first clustered the emails by thread and

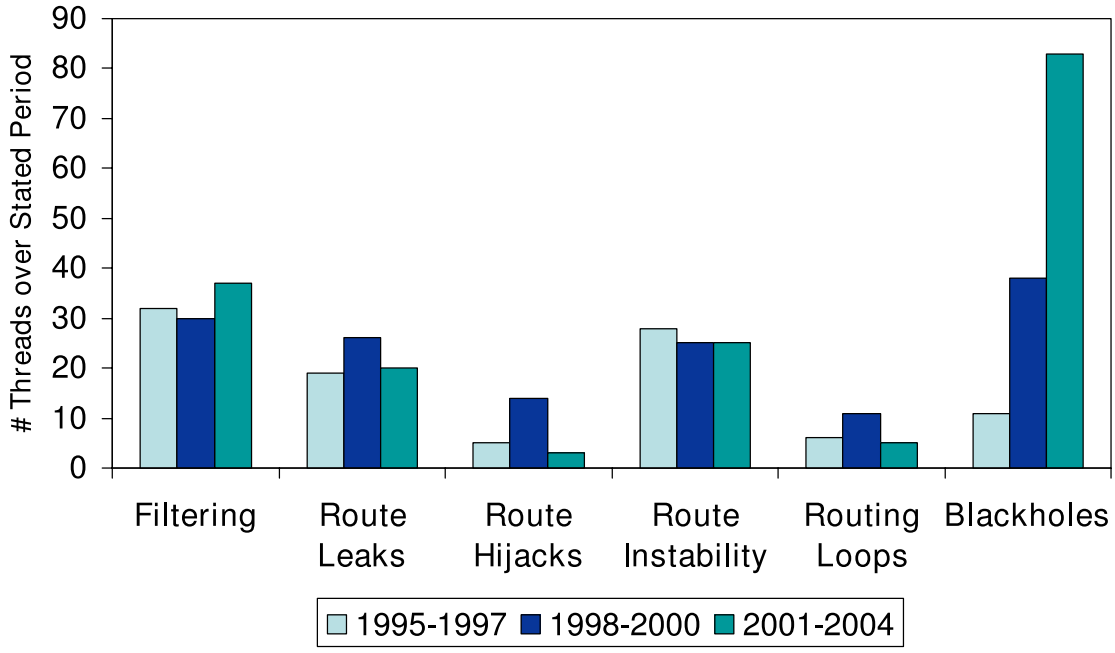


Figure 4-1: Number of threads discussing routing faults on the NANOG mailing list.

pruned threads based on a list of about fifteen keywords (*e.g.*, "BGP", "issue", "loop", "problem", "outage"). We then reviewed these threads and classified each of them into one or more of the categories shown in Figure 4-1. This informal study shows some clear trends. First, many routing problems are caused by configuration faults. Second, the same types of problems continually appear. Third, BGP configuration problems continually perplex even experienced network operators. A tool like *rcc* that can *proactively* detect configuration faults will clearly benefit network operators.

Remarkably, static configuration analysis can detect many of these configuration faults before the faulty configuration is ever deployed on a live network.

Detecting BGP configuration faults poses several challenges beyond simply defining a correctness specification. First, a high-level correctness specification, such as the one defined in Chapter 3, must be used to derive a set of constraints that can be tested against the actual configuration. Second, BGP configuration is distributed—analyzing routing configuration requires both synthesizing distributed configuration fragments and representing the configuration in a form that makes it easy to test constraints. This chapter tackles these challenges and makes the following two contributions:

1. We present the design and implementation of ***rcc***. *rcc* focuses on detecting faults that have the potential to cause *persistent* routing failures. *rcc* is not concerned with correctness during convergence (since any distributed protocol will have transient inconsistencies during convergence). *rcc*'s goal is to detect problems that may exist in the steady state, even when the protocol converges to some stable outcome.
2. We use *rcc* to explore the extent of real-world BGP configuration faults; this chap-

ter presents the first published analysis of BGP configuration faults in real-world ISPs. We have analyzed real-world, deployed configurations from 17 different ASes and detected more than 1,000 BGP configuration faults that had previously gone undetected by operators. These faults ranged from simple "single router" faults (*e.g.*, undefined variables) to complex, *network-wide* faults involving interactions between multiple routers. To date, *rcc* has been downloaded by over seventy network operators.

rcc is intended to be used *before* configurations are deployed, but we also used *rcc* to study the deployed configurations of live, operational networks. In these networks, *rcc* discovered many faults that could potentially cause failures. These include: (1) faults that could have caused network partitions due to errors in how external BGP information was being propagated to routers inside an AS, (2) faults that caused invalid routes to propagate inside an AS, and (3) faults in policy expression that caused routers to advertise routes (and hence potentially forward packets) in a manner inconsistent with the AS's desired policies. Our findings indicate that configuration faults that can cause serious failures are often not immediately apparent (*i.e.*, the failure that results from a configuration fault may only be triggered by a specific failure scenario or sequence of route advertisements). If *rcc* were used before BGP configuration was deployed, we expect that it would be able to detect faults that immediately caused routing failures as well.

Our analysis of real-world configurations suggests that most configuration faults stem from three main causes. First, the mechanisms for propagating routes within a network are overly complex. The main techniques used to propagate routes scalably within a network (*e.g.*, "route reflection with clusters") are easily misconfigured. Second, many configuration faults arise because configuration is distributed across routers: even simple policy specifications require configuration fragments on multiple routers in a network. Third, configuring policy often involves low-level mechanisms (*e.g.*, "route maps", "community lists", etc.) that should be hidden from network operators.

The rest of this chapter proceeds as follows. Section 4.1 describes the design of *rcc*. Sections 4.2 and 4.3 highlight some of *rcc*'s path visibility and route validity tests. Section 4.4 describes implementation details. Section 4.5 presents configuration faults that *rcc* discovered in 17 operational networks. Section 4.6 summarizes the take-away lessons from this study, and Section 4.7 concludes.

■ 4.1 *rcc* Design

rcc analyzes both single-router and network-wide properties of BGP configuration and outputs a list of configuration faults. *rcc* checks that the BGP configuration satisfies a set of constraints, which are based on a correctness specification. Figure 4-2 illustrates *rcc*'s high-level architecture.

We envision that *rcc* has three classes of users: those that wish to run *rcc* with no modifications, those that wish to add new constraints concerning the existing specification, and those that wish to augment the high-level specification. *rcc*'s modular design allows users to specify other constraints without changing the system internals. Some users may wish to extend the high-level specification to include other aspects of correctness (*e.g.*, safety [61]) and map those high-level specifications to constraints on the configuration.

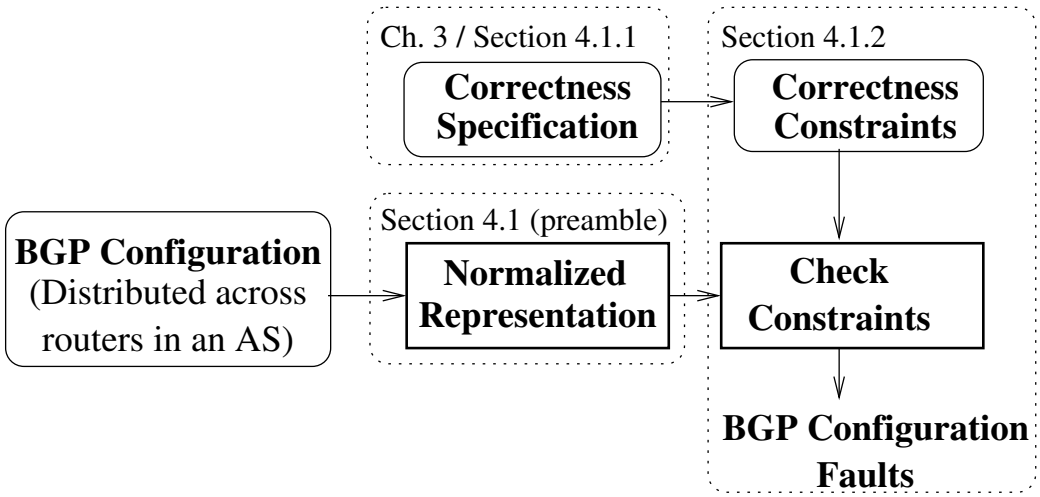


Figure 4-2: Overview of *rcc*.

Table	Description
global options	router, various global options (*e.g.*, router ID)
sessions	router, neighbor IP address, eBGP/iBGP, pointers to policy, options (*e.g.*, route-reflector client)
prefixes	router, prefix originated by this router
import/export filters	normalized representation of filter: IP range, mask range, permit or deny
import/export policies	normalized representation of policies
loopback address(es)	router, loopback IP address(es)
interfaces	router, interface IP address(es)
static routes	static routes for prefixes
	Derived or External Information
undefined references	policies and filters that a BGP configuration referenced but did not define
bogon prefixes	prefixes that should always be filtered on eBGP sessions [21]

Table 4-1: Normalized configuration representation.

In this section, we describe how *rcc* generates a normalized representation of the configuration that facilitates constraint checking. As described in Section 4.1.1, we use the correctness specification from Chapter 3 as a guide for deriving actual correctness constraints that *rcc* can check against the normalized configuration. Section 4.1.2 explains this process.

rcc implements the normalized representation as a set of relational database tables. This approach allows constraints to be expressed independently of router configuration languages. As configuration languages evolve and new ones emerge, only the parser must be modified. It also facilitates testing network-wide properties, since all of the information related to the network's BGP configuration can be summarized in a handful of tables. A relational structure is natural because many sessions share common attributes (*e.g.*, all sessions to the same neighboring AS often have the same policies), and many policies have common clauses (*e.g.*, all eBGP sessions may have a filter that is defined in exactly the same way). Table 4-1 summarizes these tables; Section 4.4.1 details how *rcc* populates them.

4.1.1 Applying the Correctness Specification to BGP Configuration

rcc's correctness specification uses the properties from Chapter 3 as a starting point. *rcc* checks two aspects of the correctness specification outlined in Chapter 3: *path visibility* and *route validity*. *rcc* finds path visibility and route validity violations in *BGP configuration only*. To make general statements about path visibility and route validity, *rcc* assumes that the internal routing protocol (*i.e.*, interior gateway protocol, or "IGP") used to establish routes between any two routers within a AS is operating correctly. BGP requires the IGP to operate correctly because iBGP sessions may traverse multiple IGP hops and because the "next hop" for iBGP-learned routes is typically several IGP hops away.

rcc detects faults that cause *persistent* failures. Both Chapter 3 and previous work (*e.g.*, [61]) have studied conditions on the relationships between iBGP and IGP configuration that must be satisfied to guarantee that iBGP converges; *rcc* does not yet parse IGP configuration, so it does not check for violations of these constraints. The correctness specifications and constraints assume that, given stable inputs, the routing protocol *eventually* converges to some steady state behavior.

Currently, *rcc* only detects faults in the BGP configuration of a *single AS* (a network operator typically does not have access to the BGP configuration from other ASes). Fortunately, because an AS's BGP configuration explicitly controls both dissemination and filtering, many configuration faults, including partitions, route leaks, etc., can be detected by analyzing the BGP configurations of set of the routers in a single AS.

4.1.2 Deriving Correctness Constraints and Detecting Faults

Deriving constraints on the configuration itself that guarantee that the correctness specification is satisfied is challenging. We reason about how the aspects of configuration from Section 2.3.1 affect each correctness property and derive appropriate constraints for each of these aspects. Table 4-2 summarizes the correctness constraints that *rcc* checks, which follow from determining which aspects of configuration affect each aspect of the correctness specification (from Section 4.1.1). These constraints are an attempt to map the path visibility and route validity specifications to constraints on BGP configuration that can be checked against the actual configuration.

Ideally, operators would run *rcc* to detect configuration faults *before* they are deployed. Some of *rcc*'s constraints detect faults that would most likely become active immediately upon deployment. For example, a router that is advertising routes with an incorrect next-hop attribute will immediately prevent other routers that use those routes from forwarding packets to those destinations. In this case, *rcc* can help a network operator diagnose configuration faults and prevent them from introducing failures on the live network.

Many of the constraints in Table 4-2 concern faults that could remain undetected even after the configuration has been deployed because they remain masked until some sequence of messages triggers them. In these cases, *rcc* can help operators find faults that could result in a serious failure. Section 4.2 describes one such path visibility fault involving dissemination in iBGP in further detail. In other cases, checking constraints implies some knowledge of high-level policy (recall that route validity, as defined in Definition 3.7, concerns a path that conforms to some high-level policy). In the absence of a high-level policy specification language, *rcc* must make inferences about a network operator's intentions.

Problem	Possible Active Fault
	Path Visibility
Dissemination Problems	
Signaling partition: - of route reflectors - within a RR "cluster" - in a "full mesh"	Router may learn a suboptimal route or none at all.
Routers with duplicate: - loopback address - cluster ID	Routers may incorrectly drop routes.
iBGP configured on one end	Routers won't exchange routes.
iBGP not to loopback	iBGP session fails when one interface fails.
	Route Validity
Filtering Problems	
transit between peers	Network carries traffic "for free".
inconsistent export to peer	Violation of contract.
inconsistent import	Possible unintentional "cold potato" routing.
eBGP session: - w/no filters - w/undef. filter - w/undef. policy filter: - w/missing prefix policy: - w/undef. AS path - w/undef. community - w/undef. filter	• leaked internal routes • re-advertising bogus routes • accepting bogus routes from neighbors • unintentional transit between peers
Dissemination Problems	
prepending with bogus AS	AS path is no longer valid.
originating unroutable dest.	Creates a blackhole.
incorrect next-hop	Other routers may be unable to reach the routes for a next-hop that is not in the IGP.
	Determinism
Ranking Problems	
nondeterministic MED age-based tiebreaking	Route selection depends on message order.

Table 4-2: BGP configuration problems that *rcc* detects and their potentially active faults.

Section 4.3 describes several route validity faults where *rcc* must make such inferences.

■ 4.1.3 Completeness and Soundness

rcc's constraints are neither complete nor sound; that is, they may not find all problematic configurations, and they may complain about harmless deviations from best common practice. However, practical static analysis techniques for program analysis are typically neither complete nor sound, either [94]. Figure 4-3 shows the relationships between classes of configuration faults and the class of faults that *rcc* detects. *Latent faults* are faults that are not actively causing any problems but nonetheless violate the correctness constraints. A subset of latent faults are *potentially active* faults, for which there is at least one input sequence that is certain to trigger the fault. For example, an import policy that references an undefined filter on a BGP session to a neighboring AS is a potentially active fault, which

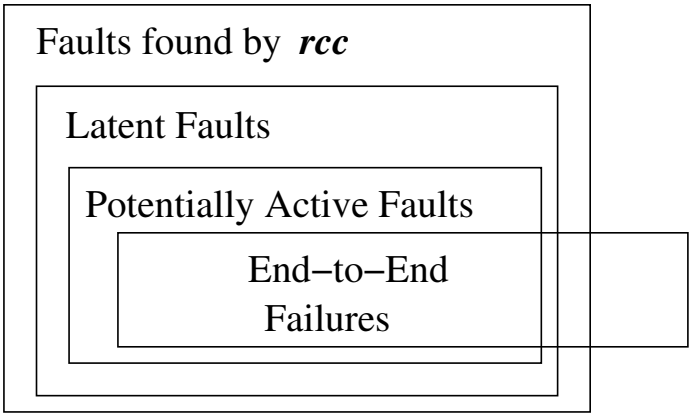


Figure 4-3: Relationships between faults and failures.

will be triggered when that neighboring AS advertises a route that ought to have been filtered. When deployed, a potentially active fault will become *active* if the corresponding input sequence occurs. An active fault constitutes a routing failure for that AS.

Some active faults may ultimately appear as *end-to-end failures*. For example, if an AS advertises an invalid route (*e.g.*, a route for a prefix that it does not own) to a neighboring AS whose import policy references an undefined filter, then some end hosts may not be able to reach destinations within that prefix. Note that a potentially active fault may not always result in an end-to-end failure if no path between the sources and destinations traverses the routers in the faulty AS.

rcc detects a subset of latent (and hence, potentially active) faults. In addition, *rcc* may also report some false positives: faults that violate the constraints but are *benign* (*i.e.*, the violations would never cause a failure). Ideally, *rcc* would detect fewer benign faults by testing the BGP configuration against an abstract specification. Unfortunately, producing such a specification requires additional work from operators, and operators may well write incorrect specifications. One of *rcc*'s advantages is that it provides useful information about configuration faults without requiring any additional work on the part of operators.

Our previous work [29] presented three properties in addition to path visibility and route validity: information flow control (this property checks if routes "leak" in violation of policy), determinism (whether a router's preference for routes depends on the presence or absence of other routes), and safety (whether the protocol converges) [56]. Our definitions of route validity (Definition 3.7), policy (Definition 3.5), and policy-conformant paths (Definition 3.6) incorporate the notion of information flow control. With a couple of exceptions (see Table 4-2), *rcc* does not check for faults related to determinism and safety. Many aspects of determinism depend on the route selection process that are inherent in today's practices (*e.g.*, the fact that MED is only comparable across routes received from the same neighboring AS) and cannot be effectively checked using static analysis. Safety is a property of the global routing system that, in practice, requires access to configurations from multiple ASes to check. In Chapter 6, we derive constraints that guarantee safety with access to configurations of only a single AS and find that these conditions are quite restrictive.

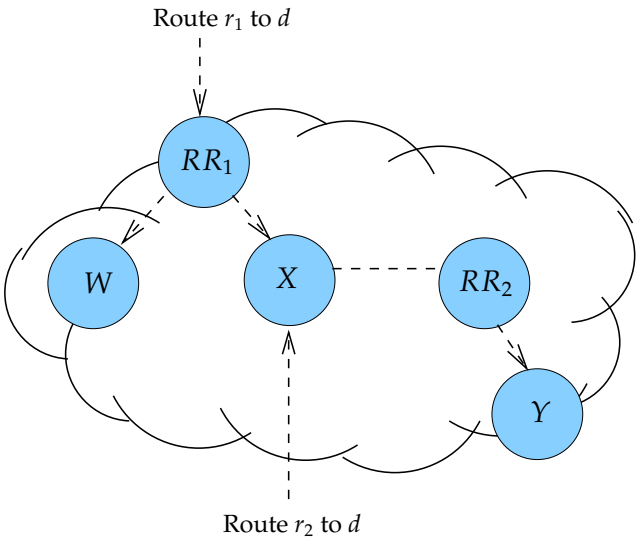


Figure 4-4: In this iBGP configuration, route r_2 will be distributed to all the routers in the AS, but r_1 will not. RR_2 and Y will not learn of r_1, leading to a network partition that won't be resolved unless another route to the destination appears from elsewhere in the AS.

■ 4.2 Path Visibility Faults

Recall that *path visibility* specifies that every router that has a usable path to a destination learns at least one valid route to that destination (Definition 3.9). It is an important property because it ensures that, if the network remains connected at lower layers, the routing protocol does not create any network partitions. Table 4-2 shows many conditions that *rcc* checks related to path visibility; in this section, we focus on iBGP configuration faults that can violate path visibility and explain how *rcc* detects these faults.

Ensuring path visibility in a "full mesh" iBGP topology (as described in Section 2.3.1) is reasonably straightforward; *rcc* checks that every router in the AS has an iBGP session with every other eBGP-speaking router. If this condition is satisfied, every router in the AS will learn all eBGP-learned routes.

A route reflector may itself be a client of another route reflector. Any router may also have iBGP sessions with other routers. Recall from Section 3.3 that we use the set of reflector-client relationships between routers in an AS to define a graph $\mathcal{I}$, where each router is a node and each session is either a directed or undirected edge: a client-reflector session is a directed edge from reflector to client, and other iBGP sessions are undirected edges. An edge exists if and only if (1) the configuration of each router endpoint specifies the loopback address of the other endpoint[1] and (2) both routers agree on session options (*e.g.*, MD5 authentication parameters). $\mathcal{I}$ should also not have partitions at lower layers. We say that $\mathcal{I}$ is *acyclic* if $\mathcal{I}$ has no sequence of directed and undirected edges that form a cycle.

[1]If a router establishes an iBGP session with a router's *loopback* address, then the iBGP session will remain active as long as that router is reachable via *any* IGP path between the two routers. If a router establishes an iBGP session with an interface address of another router, however, the iBGP session will go down if that interface fails, even if an IGP path exists between those routers.

Even a connected directed acyclic graph of iBGP sessions can violate path visibility. For example, in Figure 4-4, routers RR_2 and Y do not learn route r_1 to destination d (learned via eBGP by router RR_1), because X will not readvertise routes learned from its iBGP session with RR_1 to other iBGP sessions. We call this path visibility fault an *iBGP signaling partition*: a path exists, but neither RR_2 nor Y has a route for it. Note that simply adding a regular iBGP session between routers RR_1 and RR_2 would solve the problem.

In addition to causing network partitions, iBGP signaling partitions may result in suboptimal routing. For example, in Figure 4-4, even if RR_2 or Y learned a route to d via eBGP, that route might be worse than the route learned at RR_1. In this case, RR_2 and Y would ultimately select a suboptimal route to the destination, an event that an operator would likely fail to notice.

rcc detects iBGP signaling partitions. It determines if there is *any* combination of eBGP-learned routes such that at least one router in the AS will not learn at least one route to the destination, by performing a check based on the following result, which we proved in Section 3.3.

Theorem 3.4. *Suppose that the graph defined by an AS's iBGP relationships, I, is acyclic. Then, I does not have a signaling partition if, and only if, the eBGP-speaking routers that are not route reflector clients form a clique.*

rcc checks this condition by constructing the iBGP signaling graph I from the *sessions* table (Table 4-1). It assumes that the IGP graph is connected, then determines whether I is connected and acyclic and whether the routers at the top layer of I form a clique.

■ 4.3 Route Validity Faults

BGP should satisfy *route validity*, as defined formally in Chapter 3 (Definition 3.7). Table 4-2 summarizes the route validity faults that *rcc* checks. The biggest challenge for checking route validity is that the definition says that the routes the routers in an AS select should induce only policy-conformant paths (see Definition 3.6), but *rcc* operates without a specification of the intended policy. This section focuses on *rcc*'s approach to detecting potential policy-related problems.

Requiring operators to provide a high-level policy specification would require designing a specification language and convincing operators to use it, and it provides no guarantees that the results would be more accurate, since errors may be introduced into the specification itself. Instead, *rcc* forms *beliefs* about a network operator's intended policy in two ways: (1) assuming that intended policies conform to best common practice and (2) analyzing the configuration for common patterns and looking for deviations from those patterns. (The idea of forming beliefs about intended protocol behavior is inspired from similar ideas in systems [25].) *rcc* then finds cases where the configuration appears to violate these beliefs. It is noteworthy that, even in the absence of a policy specification, this technique detects many meaningful configuration faults and generates few false positives.

■ 4.3.1 Violations of Best Common Practice

We can derive some notions of high-level policy from our knowledge of best common practice (*i.e.*, the manner in which many ASes tend to configure their networks). In par-

ticular, *rcc* looks for two violations of best common practice: (1) advertising a route from one "peer" to another (*i.e.*, a violation of the common business practices defined in Table 2-3 (Section 2.3.1); and (2) not advertising routes in a consistent manner at all peering points [35]. In this section, we explain both of these practices in more detail.

A route that an AS learns from one of its "peers" should not be readvertised to another peer. Checking this condition requires determining how a route propagates through an AS. Figure 4-5 illustrates how *rcc* performs this check. Suppose that *rcc* is analyzing the configuration from AS X and needs to determine that no routes learned from AS B are exported to AS A. First, *rcc* determines all routes that AS X exports to AS A, typically a set of routes that satisfy certain constraints on their attributes. For example, router R_1 may export to AS A only routes that are "tagged" with the label "1000". As described in Section 2.3, ASes often assign such "community" labels to a route to control how other routers rank or filter it. *rcc* then checks the *import* policies for all sessions to AS B, ensuring that no import policy will set route attributes on any incoming route that would place it in the set of routes that would be exported to AS A. To perform this check, *rcc* must know which of an AS's neighboring ASes are peers; thus, this check requires this additional input from the network operator.

Additionally, an AS should advertise routes with equally good attributes to each peer at every peering point. An AS should not advertise routes with inconsistent attributes, since doing so may prevent its peer from implementing "hot potato" routing: If ASes 1 and 2 are peers, then the export policies of the routers in AS 1 should export routes to AS 2 that have equal AS path length and MED values. If not, router X could be forced to send traffic to AS 1 via router Y ("cold potato" routing). This behavior typically violates peering agreements. Recent work has observed that this type of inconsistent route advertisement sometimes occurs in practice [35].

An AS's policies may violate this best common practice for two reasons. First, an AS may apply different export policies at different routers to the same peer. Checking for consistent export involves comparing export policies on each router that has an eBGP session with a particular peer. Static configuration analysis is useful because it can efficiently compare policies on many different routers. In practice, this comparison is not straightforward because differences in policy definitions are difficult to detect by direct inspection of the distributed router configurations. *rcc* facilitates comparing export policies across sets of routers by normalizing all of the export policies for an AS, as described in Table 4-1 (Section 4.1). Second, an iBGP signaling partition can create inconsistent export policies because routes with equally good attributes may not propagate to all peering routes. For example, consider Figure 4-4 again. If routers W and Z both learn routes to some destination d, then route W may learn a "better" route to d, but routers Y and Z will continue to select the less attractive route. If routers X and Y readvertise their routes to a peer, then the routes advertised by X and Y will not be equally good. Thus, *rcc* also checks whether routers that advertise routes to the same peer are in the same iBGP signaling partition (as described in Section 4.2, *rcc* checks for all iBGP signaling partitions, but ones that cause inconsistent advertisement are particularly serious).

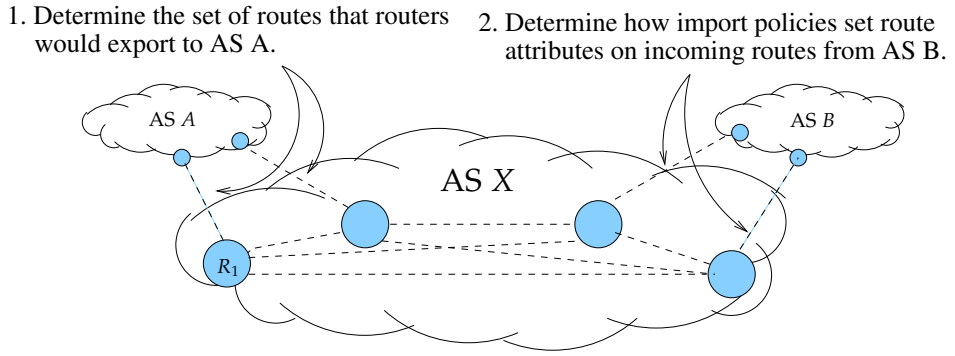


Figure 4-5: How *rcc* computes route propagation.

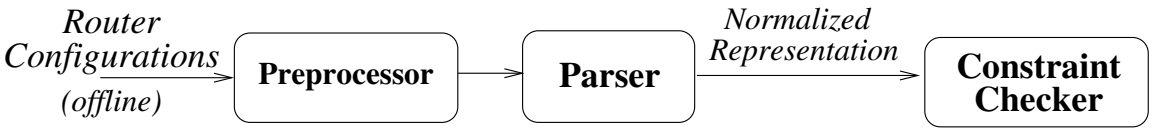


Figure 4-6: Overview of *rcc* implementation.

■ 4.3.2 Configuration Anomalies

When the configurations for sessions at different routers to a neighboring AS are the same except at one or two routers, the deviations are likely to be mistakes. This test relies on the belief that, if an AS exchanges routes with a neighboring AS on many sessions and most of those sessions have identical policies, then the sessions with slightly different policies may be misconfigurations. Of course, this test could result in many false positives because there are legitimate reasons for having slightly different import policies on sessions to the same neighboring AS (*e.g.*, outbound traffic engineering), but it does provide a useful sanity check.

■ 4.4 Implementation

rcc is implemented in Perl and has been downloaded by over seventy network operators. The parser is roughly 60% of the code. Much of the parser's logic is dedicated to policy normalization. Figure 4-6 shows an overview of *rcc*, which takes as input the set of configuration files collected from routers in a single AS using a tool such as "rancid" [113]. *rcc* converts the vendor-specific BGP configuration to a vendor-independent normalized representation. It then checks this normalized format for faults based on a set of correctness constraints. *rcc*'s functionality is decomposed into three distinct modules: (1) a preprocessor, which converts configuration into a more parsable version; (2) a parser, which generates the normalized representation; and (3) a constraint checker, which checks the constraints.

Configuration on router "gw1":

```
neighbor 10.1.2.3 remote-as 3
neighbor 10.1.2.3 route-map IMPORT_CUST in
route-map IMPORT_CUST deny 10
  match as-path 99
route-map IMPORT_CUST permit 20
  match as-path 88
  match community 10
  set localpref 80
ip as-path access-list 99 permit ^65000
ip as-path access-list 88 permit ^3
ip community-list 10 permit 0:1000
```

Normalized Representation:

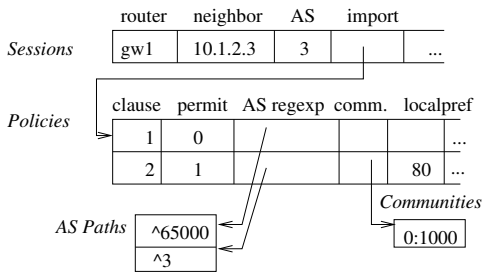


Figure 4-7: BGP configuration in normalized format.

■ 4.4.1 Preprocessing and Parsing

The *preprocessor* adds scoping identifiers to configuration languages that do not have explicit scoping (*e.g.*, Cisco IOS) and expands macros (*e.g.*, Cisco's "peer group", "policy list", and "template" options). After the preprocessor performs some simple checks to determine whether the configuration is a Cisco-like configuration or a Juniper configuration, it launches the appropriate parser. Many configurations (*e.g.*, Avici, Procket, Zebra, Quarry) resemble Cisco configuration; the preprocessor translates these configurations so that they more closely resemble Cisco syntax.

The *parser* generates the normalized representation from the preprocessed configuration. The parser processes each router's configuration independently. It makes a single pass over each router's configuration, looking for keywords that help determine where in the configuration it is operating (*e.g.*, "`route-map`" in a Cisco configuration indicates that the parser is entering a policy declaration). The parser builds a table of normalized policies by dereferencing all filters and other references in the policy; if the reference is defined after it is referenced in the same file, the parser performs lazy evaluation. When it reaches the end of a file, the parser flags any policies references in the configuration that it was unable to resolve. The parser proceeds file-by-file (taking care to consider that definitions are scoped by each file), keeping track of normalized policies and whether they have already appeared in other configurations.

Figure 4-7 shows *rcc*'s normalized representation for a fragment of Cisco IOS. In *rcc*, this normalized representation is implemented as a set of mySQL database tables corresponding to the schema shown in Table 4-1. This Cisco configuration specifies a BGP session to a neighboring router with IP address `10.1.2.3` in AS 3. This statement is represented by a row in the *sessions* table. The second line of configuration specifies that the import policy (*i.e.*, "route map") for this session is defined as "`IMPORT_CUST`" elsewhere in the file; the normalized representation represents the import policy specification as a pointer into a separate table that contains the import policies themselves. A single policy, such as `IMPORT_CUST`, is represented as multiple rows in the *policies* table. Each row represents a single clause of the policy. In this example, `IMPORT_CUST` has two clauses: the first rejects all routes whose AS path matches the regular expression number "99" (specified as "`^65000`" elsewhere in the configuration), and the second clause accepts all routes that match AS path number "88" and community number "10" (*i.e.*, `0:1000`) and sets the "lo-

cal preference" attribute on the route to a value of 80. Each of these clauses is represented as a row in the *policies* table; specifications for regular expressions for AS paths and communities are also stored in separate tables, as shown in Figure 4-7.

rcc's normalized representation does not store the names of the policies themselves (*e.g.*, "`IMPORT_CUST`", AS regular expression number "88", etc.). Rather, the normalized format only stores a description of what the route policy does (*e.g.*, "set the local preference value to 80 if the AS path matches regular expression `^3`"). Two policies may be written using entirely different names, regular expression numbers, or even in different languages, but if the policies perform the same operations, *rcc* will recognize that they are in fact the same policy.

■ 4.4.2 Constraint Checking

rcc implements constraints that detect each problem in Table 4-2 by executing SQL queries against the normalized format and analyzing the results of these queries in Perl.

rcc checks many constraints by executing simple queries against the normalized representation. Checking constraints against the normalized representation is simpler than analyzing distributed router configurations. Consider the test in Table 4-2 called "iBGP configured on one end"; this constraint requires that, if a router's configuration specifies an iBGP session to some IP address, then (1) that IP address should be the loopback address of some other router in the AS, and (2) that other router should be configured with an iBGP session back to the first router's loopback address. *rcc* tests this constraint as a single, simple "select" statement that "joins" the *loopbacks* and *sessions* tables. Other tests, such as checking properties of the iBGP signaling graph, require reconstructing the iBGP signaling graph using the *sessions* table, a task that is much easier than examining dependencies across a large set of router configuration files.

As another example, to check that no routing policy in the AS prepends any AS number other than its own, *rcc* executes a "select" query on a join of the *sessions* and *policies* tables that returns the ASes that each policy prepends (if any) and the routers where each policy is used. *rcc* then checks the *global* table to ensure that for each router, the AS number configured on the router matches the ASes that any policy on that router prepends.

■ 4.5 Evaluating Operational Networks with *rcc*

Our goal is to help operators move away from today's mode of stimulus-response reasoning by allowing them to check the correctness of their configurations *before* deploying them on a live network. *rcc* has helped network operators find faults in deployed configurations; we present these findings in this section. Because we used *rcc* to test configurations that were *already deployed* in live networks, we did not expect *rcc* to find many of the types of transient misconfigurations that Mahajan *et al.* found [85] (*i.e.*, those that quickly become apparent to operators when the configuration is deployed). If *rcc* were applied to BGP configurations before deployment, we expect that it could prevent more than 75% of the "origin misconfiguration" incidents and more than 90% of the "export misconfiguration" incidents described in that study.[2]

[2] *rcc* detects the following classes of misconfiguration described by Mahajan *et al.*: reliance on upstream filtering, old configuration, community, forgotten filter, prefix-based config, bad ACL or route map, and typo.

4.5.1 Analyzing Real-World Configurations

We made *rcc* available to operators, hoping that they would run it on their configurations and report their results. As a result, we were able to use *rcc* to evaluate the configurations from 17 real-world networks, including BGP configurations from every router in 12 ASes.

Network operators are reluctant to share router configuration because it often encodes proprietary information. Also, many ISPs do not like researchers reporting on mistakes in their networks. (Previous efforts have enjoyed only limited success in gaining access to real-world configurations [134].) We learned that providing operators with a useful tool or service increases the likelihood of cooperation. When presented with *rcc*, many operators opted to provide us with configurations, while others ran *rcc* on their configurations and sent us the output.

rcc detected over 1,000 configuration faults. The size of these networks ranged from two routers to more than 500 routers. Many operators insisted that the details of their configurations be kept private, so we cannot report separate statistics for each network that we tested. Every network we tested had BGP configuration faults, and operators were usually unaware of the faults in their networks.

4.5.2 Fault Classification and Summary

Table 4-3 summarizes the faults that *rcc* detected. *rcc* discovered potentially serious configuration faults as well as benign ones. The fact that *rcc* discovers benign faults underscores the difficulty in specifying correct behavior. Faults have various dimensions and levels of seriousness. For example, one iBGP partition indicates that *rcc* found one case where a *network* was partitioned, but one instance of unintentional transit means that *rcc* found two *sessions* that, together, caused the AS to carry traffic in violation of high-level policy. The absolute number of faults is less important than noting that many of the faults occurred at least once.

Figure 4-8 shows that many faults appeared in many different ASes. We did not observe any significant correlation between network complexity and prevalence of faults, but configurations from more ASes are needed to draw any strong conclusions. The rest of this section describes the extent of the configuration faults that we found with *rcc*. We survey faults related to path visibility, route validity, and determinism, respectively.

4.5.3 Path Visibility Faults

The path visibility faults that *rcc* detected involve iBGP signaling and fall into three categories: problems with "full mesh" and route reflector configuration, problems configuring route reflector clusters, and incomplete iBGP session configuration. Detecting these faults required access to the BGP configuration for every router in the AS.

iBGP signaling partitions. iBGP signaling partitions appeared in one of two ways: (1) the top layer of iBGP routers was not a full mesh; or (2) a route reflector cluster had two or more route reflectors, but at least one client in the cluster did not have an iBGP session with every route reflector in the cluster. Together, these accounted for 9 iBGP signaling partitions in 5 distinct ASes, one of which was benign. While most partitions involved route reflection, we were surprised to find that even small networks had iBGP signaling partitions. In one network of only three routers, the operator had failed to configure a full

Problem	Latent	Benign
Path Visibility		
Dissemination Problems		
Signaling partition:		
- of route reflectors	4	1
- within a RR "cluster"	2	0
- in a "full mesh"	2	0
Routers with duplicate:		
- loopback address	13	120
iBGP configured on one end or not to loopback	420	0
Route Validity		
Filtering Problems		
transit between peers	3	3
inconsistent export to peer	231	2
inconsistent import	105	12
eBGP session:		
- w/no filters	21	—
- w/undef. filter	27	—
- w/undef. policy	2	—
filter:		
- w/missing prefix	196	—
policy:		
- w/undef. AS path	31	—
- w/undef. community	12	—
- w/undef. filter	18	—
Dissemination Problems		
prepending with bogus AS	0	1
originating unroutable dest.	22	2
incorrect next-hop	0	2
Determinism		
Ranking Problems		
nondeterministic MED	43	0
age-based tiebreaking	259	0

Table 4-3: BGP configuration faults in 17 ASes.

mesh; he told us that he had "inadvertently removed an iBGP session". *rcc* also found two cases where routers in a cluster with multiple route reflectors did not have iBGP sessions to all route reflectors in that cluster.

rcc discovered one benign iBGP signaling partition. The AS in question had a group of routers that did not exchange routes with the rest of the iBGP-speaking routers, but the routers that were partitioned introduced all of the routes that they learned from neighboring ASes into the IGP, rather than readvertising them via iBGP. The operator of this network told us that these routers were for voice over IP (VoIP) traffic; presumably, these routers injected all routes for this application into the IGP to achieve fast convergence after a failure or routing change. In cases such as these, BGP configuration cannot be checked in isolation from other routing protocols.

Route reflector cluster problems. In an iBGP configuration with route reflection, multiple route reflectors may serve the same set of clients. This group of route reflectors and its clients is called a "cluster"; each cluster should have a unique ID, and all routers in the cluster should be assigned the same cluster ID. If a router's BGP configuration does not

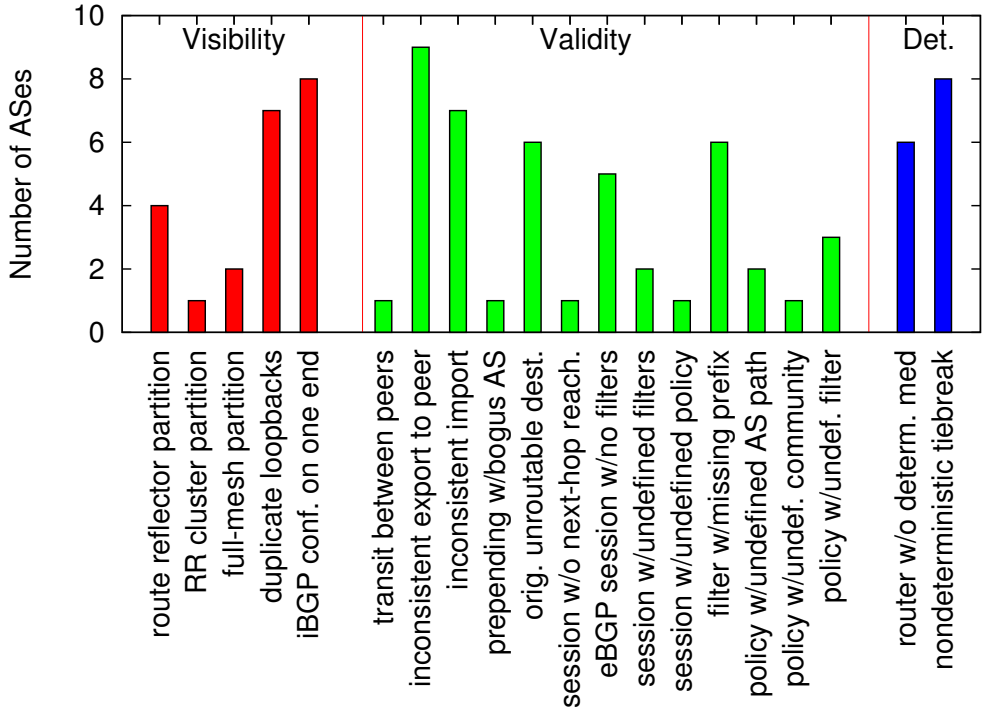


Figure 4-8: Number of ASes in which each type of fault occurred at least once.

specify a cluster ID, then typically a router's loopback address is used as the cluster ID. If two routers have the same loopback address, then one router may discard a route learned from the other, thinking that the route is one that it had announced itself. *rcc* found 13 instances of routers in distinct clusters with duplicate loopback addresses and no assigned cluster ID. Often, these apparent mistakes may be benign: different physical routers in the same AS may legitimately have identical loopback addresses. For example, routers in distinct IP-layer virtual private networks may route the same IPv4 address space.

Incomplete iBGP sessions. *rcc* discovered 420 incomplete iBGP sessions (*i.e.*, a configuration statement on one router indicated the presence of an iBGP session to another router, but the other router did not have an iBGP session in the reverse direction). Many of these faults are likely benign. The most likely explanation for the large number of these faults is that network operators may disable sessions by removing the configuration from one end of the session without ever "cleaning up" the other end of the session.

■ 4.5.4 Route Validity Faults

In this section, we discuss route validity faults. We first discuss filtering-related faults; we classify faults as latent unless a network operator explicitly told us that the fault was benign. We also describe faults concerning undefined references to policies and filters. Some of these faults, while simple to check, could have serious consequences (*e.g.*, leaked routes), if *rcc* had not caught them and they had been activated. Finally, we present some interesting route validity faults related to route dissemination, all of which were benign.

Filtering Problems

Decomposing policies across configurations on different routers can cause faults, even for relatively simple policies. *rcc* discovered the following problems:

Transit between peers. *rcc* discovered three instances where routes learned from one peer or provider could be readvertised to another; typically, these faults occurred because an export policy for a session was intended to filter routes that had a certain community value, but the export policy instead referenced an undefined community.

Obsolete contractual arrangements can remain in configuration long after those arrangements expire. *rcc* discovered one AS that appeared to readvertise certain prefixes from one peer to another. Upon further investigation, we learned that the AS was actually a previous owner of one of the peers. When we notified the operator that his AS was providing transit between these two peers, he told us, "Historically, we had a relationship between them. I don't know what the status of that relationship is these days. Perhaps it is still active—at least in the configs!"

Inconsistent export to peer. We found 231 cases where an AS advertised routes that were not "equally good" at every peering point. It is hard to say whether these inconsistencies are benign without knowing the operator's intent, but roughly twenty of these inconsistencies were certainly accidental. For example, one inconsistency existed because of an undefined AS path regular expression referenced in the export policy; these types of inconsistencies have also been observed in previous measurement studies [35].

Inconsistent import policies. A recent measurement study observed that ASes often implement policies that result in late exit (or "cold potato") routing, where a router does not select the BGP route that provides the closest exit point from its own network [126].[3] *rcc* found 117 instances where an AS's import policies explicitly implemented cold potato routing, which supports this previous observation. In one network, *rcc* detected a different import policy for *every* session to each neighboring AS. In this case, the import policy was labeling routes according to the router at which the route was learned.

Inconsistent import and export policies were not always immediately apparent to us upon casual inspection, even after *rcc* detected them. In one case, two sessions applied policies with the same name, and both policies were defined with verbatim configuration fragments. The difference resulted from the fact that the difference in policies was three levels of indirection deep. For example, one inconsistency occurred because of a difference in the definition for an AS path regular expression that the export policy referenced (which, in turn, was referenced by the session parameters).

rcc also detected filtering problems on single-router configurations:

Undefined references in policy definitions. Several large networks had router configurations that referenced undefined variables and BGP sessions that referenced undefined filters. These faults can sometimes result in unintentional transit or inconsistent export to peers or even potential invalid route advertisements. In one network, *rcc* found four routers with undefined filters that would have allowed a large ISP to accept and readvertise *any* route to the rest of the Internet (such a failure actually occurred in 1997 [121], as

[3] Inconsistent import policy technically concerns how configuration affects *ranking*, and it is more often intentional than not. Nevertheless, this test occasionally highlights anomalies that operators are interested in correcting, and it serves as a useful sanity check when looking for other types of anomalies (such as dynamically detecting inconsistent route advertisements from a neighboring AS [35]).

we described in Section 1.2.2); this potentially active fault could have been catastrophic if a customer had (unintentionally or intentionally) announced invalid routes, since ASes typically do not filter routes coming from large ISPs. This misconfiguration occurred even though the router configurations were being written with scripts; an operator had apparently made a mistake specifying *inputs* to the scripts.

Non-existent or inadequate filtering. Filtering can go wrong in several ways: (1) no filters are used whatsoever, (2) a filter is specified but not defined, or (3) filters are defined but are missing prefixes or otherwise out-of-date (*i.e.*, they are not current with respect to the list of private and unallocated IP address space [21]).

Every network that *rcc* analyzed had faults in filter configuration. Some of these faults would have caused an AS to readvertise *any* route learned from a neighboring AS. In one case, policy misconfiguration caused an AS to transit traffic between two of its peers. Table 4-3 and Figure 4-8 show that these faults were extremely common: *rcc* found 21 eBGP sessions in 5 distinct ASes with no filters whatsoever and 27 eBGP sessions in 2 ASes that referenced undefined filters. Every AS had partially incorrect filter configuration, and most of the smaller ASes we analyzed either had minimal or no filtering. Only a handful of the ASes we analyzed appeared to maintain rigorous, up-to-date filters for private and unallocated IP address space. These findings agree with those of our recent measurement study, which also suggests that many ASes do not perform adequate filtering [34].

The reason for inadequate filtering seems to be the lack of a process for installing and updating filters. One operator told us that he would be willing to apply more rigorous filters if he knew a good way of doing so. Another operator runs sanity checks on filters and was surprised to find that many sessions were referring to undefined filters. Even a well-defined process can go horribly wrong: one operator intended to use a feed of unallocated prefixes to automatically install filters, but instead ended up readvertising them. Because there is a set of prefixes that every AS should always filter, some prefixes should be filtered by default.

Dissemination Problems

rcc found only benign faults involving dissemination.

Unorthodox AS path prepending practices. An AS will often prepend its own AS number to the AS path on certain outbound advertisements to affect inbound traffic. However, we found one AS that prepended a neighbor's AS on *inbound advertisements* in an apparent attempt to influence *outbound traffic*. One network operator also mentioned that ASes sometimes prepend the AS number of a network that they want to prevent from seeing a certain route (*i.e.*, by making that AS discard the route due to loop detection), effectively "poisoning" the route. We did not witness this poisoning in any of the configurations we analyzed.

iBGP sessions with "next-hop self". We found two cases of iBGP sessions that violated common rules for setting the next-hop attribute, both of which were benign. First, *rcc* detected route reflectors that appeared to be setting the "next hop" attribute. Although this practice is not likely to create active faults, it seemed unusual, since the AS's exit routers typically set the next-hop IP address, and route reflectors typically do not modify route attributes. Upon further investigation, we learned that some router vendors do not allow a route reflector to reset the next-hop attribute. Even though the configuration specified that

the session would reset the next-hop attribute, the configuration statement had no effect because the software was designed to ignore it. The operator who wrote the configuration specified that the next-hop attribute be reset on these sessions to make the configuration appear more uniform. Second, routers sometimes reset the next-hop on iBGP sessions to themselves on sessions to a route monitoring server to allow the operator to distinguish which router sent each route to the monitor.

4.5.5 Determinism Faults

rcc discovered more than two hundred routers that were configured such that the arrival order of routes affected the outcome of the route selection process (*i.e.*, these routers had either one or both of the two configuration settings that cause nondeterminism). Although there are occasionally reasonably good reasons for introducing ordering dependencies (*e.g.*, preferring the "most stable" route; that is, the one that was advertised first), operators did not offer good reasons for why these options were disabled. In response to our pointing out this fault, one operator told us,"That's a good point, but my network isn't big enough that I've had to worry about that yet." Nondeterministic features should be disabled by default.

4.6 Take-away Lessons

In recent years, much work has been done to understand BGP's behavior, and much has been written about the wide range of problems it has. Some argue that BGP has outlived its purpose and should be replaced; others argue that faults arise because today's configuration languages are not well-designed. We believe that our evaluation of faults in today's BGP configuration provides a better understanding of the types of errors that appear in today's BGP configuration and the problems in today's configuration languages. We now briefly explore how our findings may help inform the design of Internet routing systems in the future.

First, operational networks—even large, well-known, and well-managed ones—have faults. Even the most competent of operators find it difficult to manage BGP configuration. In particular, iBGP is misconfigured often. In fact, in the absence of a guideline such as Theorem 3.4, it is hard for a network operator to know what properties the iBGP signaling graph should have. Ideally, network operators should be able to configure an AS without having to worry about whether these types of constraints are satisfied in the first place; in other words, a network operator should not be allowed to express a configuration that violate properties such as path visibility and route validity. In Chapter 7 (Section 7.3.4), we discuss a system for disseminating BGP routes within an AS called the Routing Control Platform (RCP) [13, 31], which could explicitly enforce properties such as route validity and path visibility.

Second, we found that route filters are poorly maintained. Routes that should never be seen on the global Internet (*e.g.*, routes for private addresses) are rarely filtered, and the filters that are used are often misconfigured and outdated. We can make significant strides towards fixing these types of problems simply by changing the default behavior of router filters. For example, because private address space (*i.e.*, as specified in RFC 3330 [71]) should not be advertised on the global Internet, routers could, by default, prevent routes

for this address space from leaking across AS boundaries. Of course, network operators who required routes for this address space to be advertised across AS boundaries (*e.g.*, in cases where a single administrative domain comprises multiple ASes) could configure that behavior as an *exception*, but the default behavior would reduce the likelihood of erroneous route leaks.

Third, the majority of the configuration faults that *rcc* detected resulted from the fact that an AS's configuration is distributed across its routers. Maintaining network-wide policy consistency appears to be difficult; invariably, in most ASes there are routers whose configuration appears to contradict the AS's desired policy. A routing architecture or configuration management system that enabled an operator to configure the network from a centralized location with a high-level language would likely prevent many serious faults.

Finally, although operators use tools that automate some aspects of configuration, these tools are not a panacea. In fact, we found cases where the incorrect use of these tools *caused* configuration faults. This observation suggests that static configuration analysis will play an important role in the configuration workflow, regardless of future improvements in configuration languages or routing architectures. As long as the routing protocol offers flexible configuration, the potential for incorrect behavior exists. Our work has demonstrated that detecting incorrect behavior *proactively* using static configuration analysis is not only surprisingly effective, but it is also necessary for detecting faulty configurations before they introduce erroneous behavior on a live network.

■ 4.7 Summary

Despite the fact that BGP is almost 10 years old, operators continually make the same mistakes as they did during BGP's infancy. Our work takes a step towards improving this state of affairs by making the following contributions:

- We use the correctness specification from Chapter 3 to design and implement *rcc*, a static analysis tool that detects faults by analyzing the BGP configuration across a single AS. With *rcc*, network operators can find many faults *before* deploying configurations in an operational network. *rcc* has been downloaded by over seventy network operators.
- We use *rcc* to explore the extent of real-world BGP misconfigurations. We have analyzed real-world, deployed configurations from 17 different ASes and detected more than 1,000 BGP configuration faults that had previously gone undetected by operators.

In light of our findings, we suggest two ways to make interdomain routing less prone to configuration faults. First, protocol improvements, particularly in intra-AS route dissemination, could avert many BGP configuration faults. The current approach to scaling iBGP should be replaced. Route reflection serves a single, relatively simple purpose, but it is the source of many faults, many of which cannot be checked with static analysis of BGP configuration alone [61]. The protocol that disseminates BGP routes within an AS should enforce path visibility and route validity; the Routing Control Platform offers one possible solution.

Second, BGP should be configured with a centralized, higher-level specification language. Today's BGP configuration languages enable an operator to specify router-level *mechanisms* that implement high-level policy, but the distributed, low-level nature of the configuration languages introduces complexity, obscurity, and opportunities for misconfiguration rather than design flexibility or expressiveness. For example, *rcc* detects many faults in implementation of some high-level policies in low-level configuration; these faults arise because there are many ways to implement the same high-level policy, and the low-level configuration is unintuitive. Ideally, a network operator would never touch low-level mechanisms (*e.g.*, the community attribute) in the common case. Rather than configuring routers with a low-level language, an operator should configure the *network* using a language that directly reflects high-level policies.

There are three types of baseball players: those who make it happen, those who watch it happen, and those who wonder what happens.
- Tommy Lasorda

CHAPTER 5
Predicting BGP Routes with Static Analysis

To control the flow of traffic through their networks, operators need to know how configuration changes affect the routes that each router in the network selects. The outcome of this route selection depends on the routes advertised by neighboring domains, the internal topology, the interdomain routing policies, and the intradomain link weights. To avoid costly debugging time and catastrophic mistakes, operators must be able to quickly predict the routes that each router selects. Our approach to solving this problem is to develop a network-wide model of BGP route selection that enables fast, efficient computation of these routes. In this chapter, we present efficient algorithms for computing the routing decision at each router in an AS. Ordinarily, such computation would require a complex simulation of routing protocol dynamics that might not reflect the outcome on a live network anyhow if the routing system oscillates or has multiple stable outcomes. Using the correctness specification from Chapter 3 and a tool like *rcc* (Chapter 4) to check that the properties of the correctness specification hold, however, we can exploit the fact that a routing protocol that satisfies some of these properties makes computing the network-wide outcome of BGP route selection much simpler.

In designing these route prediction algorithms, we grapple with two features of the BGP: violations of determinism (Definition 3.12), and limited visibility into the available routes for each destination. This chapter presents three main contributions.

1. **Algorithms for predicting BGP route selection.** Rather than analyzing BGP *dynamics*, we present efficient algorithms to compute the outcome of the distributed route-selection process using only *static* inputs. Our algorithms exploit the following observation: when a routing system converges, the outcome does not depend on the order and timing of the messages, allowing our algorithms to model a message ordering that efficiently computes the outcome of BGP route selection.

   Section 5.3 presents practical constraints that enable efficient computation of network-wide BGP route selection and describes an overview of the algorithms presented in the rest of the chapter. After we introduce some notation (Section 5.4), Section 5.5 presents an algorithm that computes the outcome of BGP route selection

for the simple case where every router in the AS receives every router's best eBGP-learned route for a destination (which corresponds to a full mesh iBGP configuration) and where determinism is satisfied (which corresponds to a routing protocol where all route attributes can be compared across all routes; *i.e.*, without MED[1]). This algorithm turns out to be quite simple.

The balance of the chapter deals with route prediction in networks that employ MED, route reflection, or both. We first present algorithms that handle MED and route reflectors in isolation. We then discuss why the interaction between these two features precludes efficient route prediction. Section 5.6 focuses on algorithms that capture the effects of the MED attribute, assuming a full-mesh iBGP configuration. In Section 5.7, we consider iBGP configurations that use route reflection.

2. **Prototype implementation.** We describe a prototype implementation of a tool based on our prediction algorithms. This tool provides fast, accurate answers to "what if" questions about the effects of configuration changes on the flow of traffic through the network. We tested this prototype on a large tier-1 ISP to demonstrate that our prediction algorithms are fast and accurate enough to be used in practice. Section 5.8 summarizes the design, evaluation, and validation of this tool [37].
3. **Proposed improvements to BGP.** Two features—the MED attribute and route reflection—complicate route prediction and create difficulties for the operation of BGP itself. Section 5.9 suggests ways to improve the design and operation of BGP to avoid the harmful effects without sacrificing the policy semantics of MEDs and the scalability provided by route reflectors.

■ 5.1 Motivation and Overview

This section briefly reviews the BGP route selection process (Table 2-2 and Section 2.2.2 provide coverage in greater depth) and discusses why network operators need algorithms to efficiently compute the outcome of this route selection process.

Recall from Section 2.2.2 that the route selected by each router depends on the interactions between three routing protocols, as shown in Figure 5-1. First, routers in the AS use *external BGP (eBGP)* to receive route advertisements from neighboring ASes. For example, the routers W, X, and Y each have eBGP sessions with neighboring ASes. As described in Section 2.3.1, to influence the route selection process, a router may apply an *import* policy to modify the attributes of the routes learned from a neighbor.

Second, the routers use *internal BGP (iBGP)* to disseminate the routes to the rest of the AS. In the simplest case, each router has an iBGP session with every eBGP-speaking router, forming an "full mesh" configuration. If two routes are equally good through the first four steps in Table 2-2, the router favors an eBGP-learned route over an iBGP-learned one. In Figure 5-1, router Z receives three iBGP routes, from routers W, X, and Y. If the routes that Z learns have equal local preference, AS path length, origin type, and MED values, it uses the IGP to break ties between the remaining routes 2-2.

[1]Throughout the chapter, we often describe BGP "without MED". Network configurations "without MED" could also be viewed as a configuration that compares the MED attributes across all routes (*e.g.*, in Cisco IOS, this behavior can be enabled with `always-compare-med` setting).

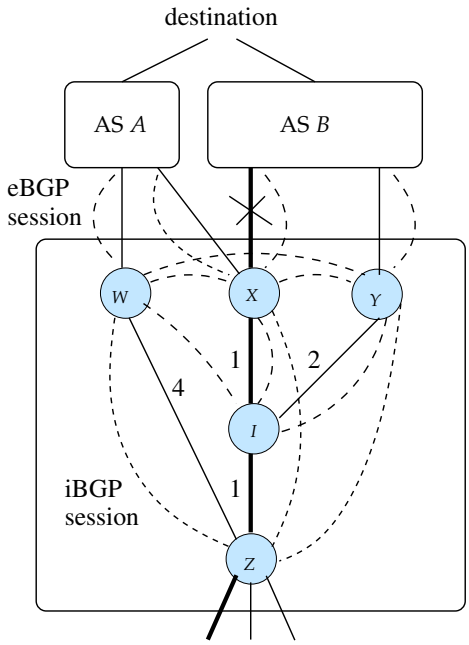


Figure 5-1: Network with three egress routers connecting to two neighboring ASes: Solid lines correspond to physical links (internal links are annotated with IGP link weights) and dashed lines correspond to BGP sessions. Thick lines illustrate the shortest path from one router to its closest egress point for reaching the destination.

Third, the routers run an Interior Gateway Protocol (IGP) to learn how to reach each other. Two common IGPs today are OSPF [91] and IS-IS [104], which compute shortest paths based on configurable link weights; the routers also use the IGP path costs in the sixth step in Table 2-2. In Figure 5-1, router Z selects the route with the smallest IGP path cost of 2, learned from router X.[2]

After selecting a route to each destination, each router combines the BGP and IGP information to construct a forwarding table that maps the destination prefix to the outgoing link along the shortest path. In Figure 5-1, the forwarding path consists of the thick lines from the ingress link at router Z to the egress link at router X.

If the link from X to AS B becomes persistently congested, the network operator may need to adjust the configuration of the routing protocols to direct some of the traffic to other egress routers. For example, the operator could modify the import policy at router X for the routes it learns from AS A and AS B to make the BGP routes for some destinations look less attractive than the routes received at other routers [32]. Changing the import policy in this way causes the route that X readvertises via iBGP to carry a smaller *local preference*, which influences the routes that other routers in the network select. For example, changing the import policy at X has the *indirect* effect of directing some of the traffic entering at router Z to egress router Y (the next-closest egress point, in terms of the IGP path costs), thereby alleviating the congestion on the link connecting X to AS B. Net-

[2]If two routes have the same IGP path cost, the router performs an arbitrary tiebreak in the seventh step in Table 2-2.

work operators make similar kinds of configuration changes for a variety of other reasons, such as exploiting new link capacity, preparing for maintenance on part of the network, or reacting to equipment failures.

Operators must predict the effects of changes to the routing protocol configuration before modifying the configuration on a live network. Human intuition is not sufficient for understanding the complex interactions between three routing protocols running on a large, dynamic network. Experimenting on a live network runs the risk of making disruptive configuration changes that degrade performance. Instead, we believe that operators should have an accurate and efficient tool that computes the effects of configuration changes on the flow of traffic through the network. This tool should allow a network operator (or an automated optimization algorithm) to efficiently explore the large space of possible configurations.

■ 5.2 Problem Statement and Challenges

Our goal is to compute the *outcome*—the routing decision for each router—once the routing protocols have converged. Accordingly, we present algorithms that accurately and quickly determine how the network configuration and the routes learned via eBGP affect the flow of traffic through an AS. While some existing tools simulate BGP's behavior [7, 16, 127], this work is the first to develop algorithms that determine the outcome of the BGP route selection process at each router in an AS without simulating the dynamics of the protocol.

Efficiently predicting the route that each router in an AS ultimately selects is challenging because the route selected by one router often depends on the routes selected by other routers in the AS. Consider Figure 5-2. In this example, router R_1 receives two routes via eBGP, while R_2 receives a single route via eBGP. To determine the route that each one of these routers ultimately selects, we must first determine the candidate routes available to each router. Of course, the set of candidate routes available to each of these routers depends on the route that the other selects! This circular dependency seems to imply some "back and forth" reasoning (*i.e.*, determining the route that R_1 selects depends on the route that R_2 selects, which in turn depends on the route that R_1 selects, etc.). Efficiently resolving these types of circular dependencies is the focus of this chapter.

> **Problem:** Given only a static snapshot of the routing configuration for the routers in an AS and the eBGP-learned routes received by the routers in the AS, determine the route that each router in an AS selects for each destination, while considering each AS's available candidate routes only once.

Solving this problem would be easy if (1) the decision process in Table 2-2 allowed each router to form a ranking of the candidate routes that satisfied determinism (Definition 3.12); and (2) the dissemination of routes in iBGP ensured each router received the best route for a destination from every eBGP-speaking router. If these two properties held, then a simple algorithm that simply considered which route each router would select from all of the eBGP-learned routes would correctly compute the outcome of BGP route selection without having to revisit any routers. Unfortunately, two features of BGP cause these properties to be violated, thus making route prediction more challenging:

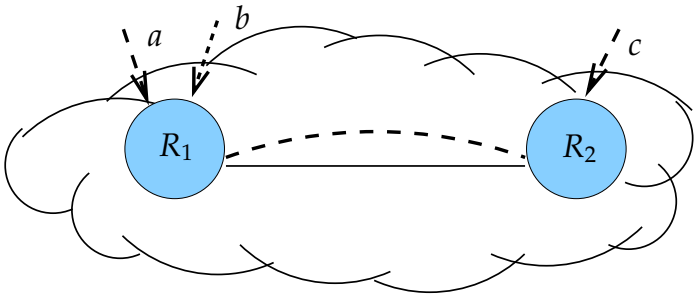


Figure 5-2: Route prediction requires resolving circular dependencies. Determining the route that R_2 ultimately selects (*i.e.*, a, b, or c) first requires determining whether R_1 selects route a or b. Ultimately, R_1's selected route could depend on whether it learns route c from R_2, which requires revisiting R_1.

First, **in practice, each router's ranking function violates determinism**; the violation is caused by BGP's MED attribute. An eBGP neighbor can set the MED attributes of route advertisements on different BGP sessions to influence the behavior of the routers that receive these routes in a neighboring AS. For example, in Figure 5-1, AS B may send a route with a MED of 10 to router Y and a route with a MED of 20 to router X; as a result, Z would select the route from Y with the smaller MED, even though the IGP path to X is shorter. The MED comparison in step 4 of the decision process applies only to routes learned from the *same* next-hop AS. When MEDs are used in this fashion, however, each router's ranking function violates determinism (see Figure 3-10). In other words, the choice of one route over another may depend on the presence or absence of a third route [18].

In Chapter 3, we described how violations of determinism can cause safety problems. Determinism problems also complicate route prediction, because each router's rankings among a set of candidate routes may depend on the routes that other routers in the network select. An efficient route prediction algorithm must resolve these dependencies.

Second, **routers in an AS may not receive every eBGP-speaking router's best route for a destination**. The quadratic scaling of a full-mesh iBGP configuration forces large networks to distribute routes in a hierarchical fashion. A router configured as a route reflector selects a single best route and forwards the route to its clients (see Section 2.3.1 for details). Using route reflectors reduces the number of iBGP sessions, as well as the number of routes the clients need to receive and store. Because each route reflector forwards only its *best* route to its iBGP neighbors, however, the candidate routes available at one router depend on decisions at other routers. In particular, a route reflector may make a different choice in step 6 of the route selection process (*i.e.*, shortest IGP path to the next-hop IP address) than its clients would have because it is located at a different place in the IGP topology than its clients.

In general, these two features of BGP cannot be ignored because operators use them often. To illustrate the extent to which these artifacts of BGP complicate route prediction, we present the "ideal" route prediction algorithm in Section 5.5, before considering more complicated algorithms that capture the effects of these two artifacts.

5.3 Modeling Constraints and Algorithm Overview

In this section, we impose three constraints that the routing system must satisfy to enable efficient and accurate route prediction.

Next, we describe how these constraints enable us to decompose the algorithm into three stages—applying the import policy to eBGP-learned routes, selecting the best BGP route at each router, and computing the forwarding path. The algorithm takes as input the router configuration and a static snapshot of the routes learned via eBGP and outputs the route that each router in the AS selects, for each destination. Because the first and third stages of the algorithm are relatively simple, the rest of the chapter focuses on the second stage of computing the best BGP route at each router for each destination prefix.

5.3.1 Modeling Constraints

Efficiently computing the effects of a configuration or topology change is possible when three important conditions hold. These constraints include the various correctness properties specified in Chapter 3 that can be verified with static configuration analysis (Chapter 4). Specifically, we assume that safety, and the second condition of determinism are satisfied. Imposing these constraints free our prediction algorithms from needing to consider whether different orderings of routing will produce different results. This property allows us to focus on designing algorithms that emulate a particular message ordering that prevents the algorithm from having to revisit routers where it has already made a prediction. The rest of this section explains how these constraints and others help simplify the prediction algorithms.

First, the inputs to the algorithm must be stable. In particular,

Constraint 5.1 (Slowly changing inputs) *The eBGP-learned routes change slowly with respect to the timescale of network engineering decisions.*

If the eBGP-learned routes change frequently, the internal routing system does not have time to propagate the effects of one eBGP advertisement before the next one arrives. In practice, most BGP routes are stable for days or weeks at a time [79], and the vast majority of traffic is associated with these stable routes [120]. This allows the routing algorithm to operate on a static snapshot of the eBGP routes. Any eBGP routing change can be treated as a separate problem instance.

Second, the routers in the AS must ultimately reach a stable outcome. In particular,

Constraint 5.2 (Safety and uniqueness) *Given stable eBGP-learned routes and fixed iBGP and IGP topologies, each router inside the AS converges to a unique routing decision.*

If the routers continually change the routes that they select, accurately predicting how the flow of data traffic becomes significantly more challenging. Fortunately, previous work [61] has identified sufficient conditions for an internal routing configuration to satisfy Constraint 5.2. We describe these conditions in more detail in Section 5.7 when we address the challenges introduced by route reflectors.

Third, the routing decisions at each router should not depend on message ordering or timing:

Constraint 5.3 (Determinism, Condition #2) *The routing decision at each router depends only on the routes received from its neighbors and* not *the order or timing of these routing messages.*

Some router vendors have an additional step in the BGP decision process that favors the "oldest" route before the final tie-breaking step of comparing the router IDs. The "age-based tie-breaking" favors more stable routes, making the outcome of the BGP decision process dependent on the *order* the router receives the advertisements (and, hence, violating determinism). Disabling age-based tie-breaking forces a deterministic selection based on the smallest router ID, as in Figure 2-2; other BGP features, such as route flap damping [137], can help reduce the likelihood of selecting unstable routes.

Constraint 5.2, which guarantees that the routing system will converge to a unique outcome, and 5.3, which guarantees that this outcome does not depend on the ordering of routing messages, allow us to make the following observation:

Observation 5.1 *If a routing system is guaranteed to converge to a unique outcome, that outcome is independent of the order in which routers exchange routes and apply the decision process.*

This observation implies that the algorithm can consider the evolution of the routing system under *any particular message ordering*, without the risk of arriving at the wrong answer.

It is worth noting that our algorithms do not require the routing system to satisfy route validity or the first condition of determinism (recall that determinism is only a sufficient condition for iBGP to satisfy safety but is not necessary). The algorithms in this chapter are only concerned with predicting the outcome of BGP selection process, not whether the resulting routes induce paths that violate route validity. In Sections 5.6 and 5.7, we permit routers' selection functions to violate the first condition of determinism because these violations capture BGP's default behavior (*i.e.*, this condition is violated whenever routers only compare the MED attribute across routes received from the same neighboring AS). The algorithms do not explicitly require path visibility; rather, the algorithms in Sections 5.5 and 5.6 implicitly assume path visibility is satisfied (*i.e.*, they assume a "full mesh" iBGP configuration).

■ 5.3.2 Overview: Decomposing BGP Route Selection into Three Stages

Following the approach applied in other recent work [33, 47], the algorithms in this chapter compute the effects of a particular message ordering using an *activation sequence*, an *offline* analysis technique that "activates" one or more routers at each discrete step. When activated, a router applies the decision process in Table 2-2 and propagates the best route to its iBGP neighbors. Our algorithms are based on an activation sequence that allows us to decompose route prediction into three distinct stages, as shown in Figure 5-3:

1. Receiving the eBGP routes and applying import policy. This stage takes as input all of the eBGP-learned routes at each router and applies the appropriate import policies at each router *before exchanging any iBGP update messages* and outputs the set of eBGP-learned

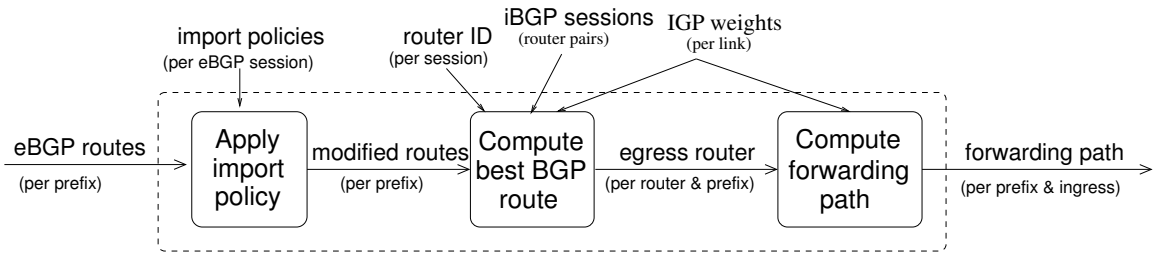


Figure 5-3: Our algorithms decompose network-wide BGP route selection into three independent stages. The algorithms take as input the eBGP-learned routes from neighboring ASes, the router IDs of each BGP session, and the routing configurations from all of the routers in the AS, which provide information about the IGP topology, the iBGP topology, and the import policies (*i.e.*, rankings) of each router.

routes after these import policies have been applied. This stage activates all of the edge routers at the same time.

Each eBGP-learned route has attributes (such as the destination prefix and the AS path) and is associated with an eBGP session. The import policy may filter the route or set certain attributes such as local preference, origin type, and multiple-exit discriminator (MED), according to attributes in the advertised route (*e.g.*, based on ASes in the AS path). Because applying the import policy is a local operation for each eBGP-learned route at each router, the first stage emulates the operations a real router would perform upon receiving each of the eBGP routes. These routes, with modified attributes, are the input to the second stage.

2. Computing the best BGP route at each router. When iBGP satisfies path visibility, many routes from the first stage could never be selected by any router as the best route. For example, an eBGP-learned route with a local preference of 90 would *never* be selected over another route with a local preference of 100. In addition, different routers in the AS may select different best BGP routes because they have different IGP path costs to the egress router. Also, a router can only consider the routes advertised by its iBGP and eBGP neighbors, which may influence the final decision at that router. This stage takes as input the set of eBGP-learned routes after the import policies of each router have been applied and outputs a single best egress router for each ingress router and destination prefix. Constructing an efficient activation sequence for this stage is challenging, and is the focus of the next four sections of the chapter.

Using Observation 5.1, our goal is to devise an *activation sequence*, which "activates" one or more routers at any given phase. When activated, a router r applies the BGP decision process to compute a best route from its available candidate routes, which it then may propagate to other routers via iBGP. In an actual network, routers may be activated in any order and may change their best route many times before the network converges. This chapter focuses on devising activation sequences that allow us to efficiently compute the final routing decision.

3. Computing the forwarding path through the AS: The third stage of the algorithms compute the effects of the IGP link weights on the construction of the forwarding path through the AS from an ingress router toward a destination prefix. Given the selected BGP route, the ingress router forwards packets along the outgoing link (or links) along shortest paths to the egress router, and the process repeats at the next router. Ideally, the traffic

Symbol	*Description*	*Section*
	FUNCTIONS ON ROUTES	
λ_r	Takes a set of routes received at router r and outputs the best route at router r, according to the BGP decision process applied at router r	5.4
γ	Takes a set of routes and extracts the subset whose attributes are equally good up through the first four steps of the decision process	5.4
σ	Takes a set of routes and extracts the subset whose attributes are equally good up through the first *three* steps of the decision process	5.6.2
	SETS OF ROUTES OR ROUTERS (INITIAL INPUTS)	
R	routers in the AS	5.4
A	routers that have been activated	5.7.2
E	eBGP-learned routes	5.4
E_r	eBGP-learned routes at router r	5.4
N	number of eBGP-learned routes (*i.e.*, $\|E\|$)	5.4
	SETS OF ROUTES (INTERMEDIATE AND FINAL OUTPUTS)	
I_r	iBGP-learned routes at router r	5.4
P_r	All routes learned at router r	5.4
b_r	The best route that router r selects.	5.4
C	The set of candidate routes at some intermediate activation. A subset of E.	5.5
C_r	The set of candidate routes at router r at some intermediate activation. A subset of E_r.	5.6.2
B	The set of best routes computed by the algorithm. A subset of C.	5.5
L	The set of routes eliminated at some activation step.	5.6.2
	IBGP TOPOLOGY	
S	iBGP sessions.	5.7.2
G	iBGP session graph. $G = (R, S)$.	5.7.2

Table 5-1: Description of the notation used in this chapter, and the sections where each piece of notation is introduced.

flows along the shortest path (or paths) all the way from the ingress router to the selected egress router. However, in practice, routers along the shortest path may have selected a *different* egress router, thus violating route validity (Definition 3.7). These violations can occur if the iBGP session configuration limits the BGP routing options at the routers [61]. By considering one router at a time, the third stage can compute an accurate view of the forwarding path(s) even when deflections occur.

While all three steps are necessary for determining the flow of traffic through a network from a static snapshot of the network state, the rest of this chapter focuses on the second step (*i.e.*, computing the best BGP route at each router), since performing this step correctly and efficiently is considerably more difficult than either of the other two steps.

■ 5.4 Preliminaries

We first introduce some notation. Table 5-1 lists the notation we use for the remainder of this chapter and summarizes where this notation is introduced. We assume that the AS has a set of N eBGP-learned routes, E, for a given destination prefix, which it learns at

R routers. E contains the eBGP-learned routes after each router in the AS has applied its local import policy (which may filter some set of the routes it receives and set or modify the route attributes of others). For convenience, we define $E_r \subseteq E$ as the set of eBGP-learned routes at router $r \in R$. At any given time, a router r also has zero or more iBGP-learned routes $I_r \subseteq E$. We define two useful functions:

- λ_r, which takes a set of routes at router r and produces the best route at router r according to the BGP decision process in Table 2-2. The subscript on λ_r is necessary because different routers can apply the BGP decision process to the same set of routes and obtain different results based on the BGP session from which they learn the route and their location in the topology. For example, in Figure 5-1, router X would treat the route learned from AS B as an eBGP-learned route with the router ID of the eBGP session with B. On the other hand, Z sees an iBGP-learned route with an IGP path cost of 2 and the router ID associated with the iBGP session to X.
- γ, which takes a set of BGP routes, C, and produces $C' \subseteq C$, such that routes in C' are the network-wide best routes based on the first four steps in Table 2-2.

  Unlike λ_r, γ has *global* (*i.e.*, network-wide) context; that is, its context is not router-specific. When the routers' selection functions do not satisfy determinism, each router's best route is not guaranteed to be in the set $\cup_r \lambda_r(E_r)$. In Sections 5.6 and 5.7, we will apply γ to a set of routes when it is safe to eliminate all routes that could *never* be the best route at any router. In these sections, we will see that as long as all routers have either complete visibility of the routes that the AS learns via eBGP or selection functions that satisfy determinism, every router will ultimately select a route from $\gamma(E)$.

■ 5.5 Simple Case: BGP with Determinism and Full Visibility

In this section, we describe an algorithm that predicts the outcome of BGP route selection when a network employs a full mesh iBGP topology and the MED attribute is compared across all routes (which we also refer to as "no MED" or "without MED"). This algorithm works as long as routers compare the MED attribute across all candidate routes and when the network does not use route reflection. This scenario may be the case for many small stub ASes that do not have customers of their own: a network that does not have many routers will typically configure its iBGP topology as a full mesh, and a stub AS typically does not receive (or honor) MEDs from the ASes from which it buys transit. In practice, some transit ISPs even configure their routers to compare the MED attribute across all candidate routes (often to avoid problems with oscillation), and most small networks do not use route reflection.

After describing the route prediction algorithm and proving its correctness, we explain two basic properties that hold in this case that make the prediction algorithm quite simple and explain why two artifacts of BGP—MED and route reflection—can cause these properties to be violated.

A full mesh iBGP topology provides full visibility of BGP routes at each router: every router learns the set of routes selected by every eBGP-speaking router in the AS. Furthermore, when the MED attribute is compared across all routes (as opposed to just those from

Algorithm: Full Mesh, No MED

```
SELECTBEST_EBGP(E, R)
    // Build the set of locally best routes at each router.
    // This set is the set of candidate best eBGP routes.
    C ← ∪_r λ_r(E_r)

    // Eliminate all routes from C that
    // do not have highest local preference,
    // shortest AS path length, lowest origin type, lowest MED
    B ← γ(C)
```

Figure 5-4: Algorithm for computing the best route at eBGP routers, assuming that MED is compared across all routes (*i.e.*, that there exists a total ordering of routes at each router).

the same neighboring AS) a router's ranking over the set of routes it learns form a total ordering, which implies that the first condition of determinism (Definition 3.12) is satisfied. These characteristics allow us to devise a relatively simple algorithm to compute the outcome of BGP route selection at each router in the AS.

In this case, the algorithm for computing the best route at every eBGP-speaking router is shown in Figure 5-4. The algorithm takes as input the set of all eBGP-learned routes, E, and the set of all eBGP-speaking routers, R, and produces the set of best eBGP routes, B. E_r refers to all eBGP-learned routes learned by router r, and C represents the set of candidate routes after each router selects the best route from the set of its eBGP-learned routes. The output of this algorithm is $B = \gamma(C)$, the set of all best routes to this destination, such that $b_r = \lambda_r(B)$. The algorithm proceeds in two steps. The first step computes the locally best BGP route at each eBGP-speaking router; this step guarantees that each router selects no more than one eBGP-learned route. The second step eliminates any route from this set for which a router would select another router's iBGP route over its own locally best route.

The first step of the algorithm scans all N eBGP-learned routes and selects the best eBGP-learned route at each router, if any; at most $|R|$ routes remain after this step. The second step selects, for each router $r \in R$, the best route from R. Thus, the running time will be $O(N + |R|^2)$, where N is the number of eBGP-learned routes, and $|R|$ is the number of routers in the system (a full mesh iBGP configuration will have $|R|(|R| - 1)$ iBGP sessions). When $|R| > N$, the N term is dominated, so the running time is $O(|R|^2)$. When $N > |R|$, however, a simpler approach to the algorithm would simply be to apply $\lambda_r(E)$ at each router, which has $O(N|R|)$ running time. Thus, the computational complexity for route prediction is proportional to the total number of routes in the system.

To prove that this algorithm is correct, we must show that this algorithm accurately emulates *one* activation sequence; Observation 5.1 guarantees that as long as the algorithm correctly emulates some activation, it will correctly emulate BGP route selection.

Theorem 5.1 *When each router can produce a total ordering over all possible candidate routes, the algorithm in Figure 5-4 correctly computes the outcome of the decision process for all routers that select an eBGP-learned route as their best route.*

Proof. We prove this theorem constructively, by showing that the algorithm correctly em-

§	*MED*	*RR*	*Running Time*	Prop. 5.1	Prop. 5.2
5.5	No	No	$O(N + \|R\|^2)$	•	•
5.6.2	Yes	No	$O(N \log N + N\|R\|)$		•
5.7.2	No	Yes	$O(N + \|S\|)$	•	•
5.7.3	Yes	Yes	$O(N \log N + N\|R\| + N\|S\|)$		

Table 5-2: Properties of the BGP route prediction algorithms in each of the four cases (with and without MED, and with and without route reflection).

ulates an activation sequence and message ordering that could occur in BGP. Consider the following ordering:

1. All routers receive routes to the destination via eBGP. Then, every router is activated simultaneously.
2. Every router advertises its locally best route via iBGP. After all iBGP messages have been exchanged, every router is activated simultaneously.

In the first phase, each router r computes $\lambda_r(E_r)$, resulting in a set of candidate routes $C = \cup_r \lambda_r(E_r)$, as in the first line of the algorithm in Figure 5-4. Then, each router learns these routes. Note that $B \subseteq C$ by definition, which means that each router that learns a route to the destination via eBGP has either zero or one route in B. We consider both cases. If a router r has a route in C but not in B, then r's eBGP-learned route $b_r = \lambda_r(E_r)$ must have been worse according to the first four steps of the decision process than some other route, $b_s = \lambda_s(E_s)$ in C (otherwise, $\gamma(C)$ would not have eliminated it). But in a full mesh iBGP topology, r would learn a route via iBGP that is at least as good as b_s, so b_r would also be eliminated in phase 2 of the activation. Of course, if a router has a route in C, then that must be the route that it would select after phase 2 of activation: it is equally good as all routes in $\gamma(C)$ through the first 4 steps of the decision process (by construction), and it prefers its own best route over any iBGP-learned route (by step 5 of the decision process). ■

This simple algorithm works because two properties hold. First, when MED is compared across all routes, every router that selects a route from the set of eBGP-learned routes will select its locally best route. Second, when the iBGP topology is a full mesh, each BGP-speaking router ultimately selects a route in $\gamma(E)$; that is, every router ultimately selects a route that has the maximum local preference, minimum AS path length, lowest origin type, and lowest MED (assuming MEDs are compared across all routes). Table 5-2 summarizes when these two properties hold, for all possible combinations of MED and route reflection (the rest of this section treats defines these two properties more formally). The table also indicates the computational complexity for computing the best route at each router, for each scenario. We now formalize these two properties, explain why they make route prediction simple, and present cases where BGP violates each of them.

■ 5.5.1 Property #1: Every best route is some router's best eBGP route.

This property holds only if every router's selection function satisfies determinism. We now formalize this property, prove that determinism is required to ensure that it holds,

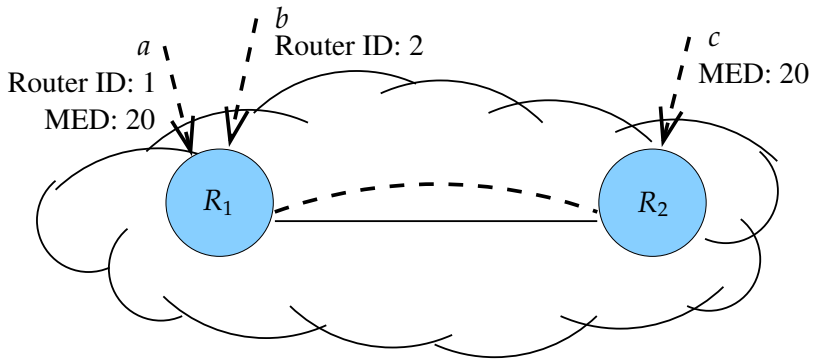


Figure 5-5: With MED, a router may select a route that is no router's best eBGP route, thus violating Property 5.1.

and show an example where this property is violated if BGP does not satisfy determinism.

Property 5.1 *If determinism is satisfied, then each router ultimately either selects its own best eBGP-learned route or some iBGP-learned route. Formally,* $b_r \in E_r \Rightarrow b_r = \lambda_r(E_r)$.

Proof. By definition, each router r applies the route selection process to the union of the routes it learns via eBGP and iBGP: $b_r = \lambda_r(E_r \cup I_r)$. Therefore, either $b_r \in E_r$ or $b_r \in I_r$. Furthermore, because the first condition of determinism is satisfied, the router r's preferences over routes in $E_r \cup I_r$ form a total ordering, so either $b_r = \lambda_r(E_r)$ or $b_r = \lambda_r(I_r)$. But, if $b_r \neq \lambda_r(E_r)$, then $b_r = \lambda_r(I_r)$, so $b_r \in I_r$ and $b_r \notin E_r$. ■

Property 5.1 makes it possible to propagate the effects of route selection at each router only once, because each router ultimately selects its locally best eBGP-learned route or some other router's locally best route.

Unfortunately, when the MED attribute is only compared among routes from the same AS, BGP does not satisfy determinism, so this property no longer holds. Figure 5-5 shows an example where this property is violated. In this example, router R_1's ranking between a and b depends on whether it learns route c. Thus, even though route a is R_1's locally best route (by the router ID tiebreak), R_2 ultimately selects route b (c eliminates a due to MED due to MED, and R_1 selects b over a because b is learned via eBGP), which is *no* router's best eBGP route. As such, the simple algorithm that selects each router's best route from the set of best eBGP routes does not work: a naïve algorithm would result in precisely the "back and forth" behavior described in Section 5.2. Section 5.6 describes an alternate route prediction algorithm that handles this case.

■ 5.5.2 Property #2: Every best route is in $\gamma(E)$.

This property states that every router selects a route that is equally good up through the MED comparison step of the decision process. Intuitively, it might seem that this property would always hold—why would a router ever select a route with a lower local preference, longer AS path, higher origin type, or higher MED value if it had a better route available?

In fact, in certain iBGP configurations, a route reflector can prevent a router from *learning* an eBGP-learned route with a lower MED value than the one it selects. Property 5.2 holds if either the iBGP topology is a full mesh or determinism is satisfied. We now formally state the conditions when this property holds, show an example where a BGP configuration can violate this property, and briefly discuss its implications for route prediction.

Property 5.2 *If (1) every router in the AS receives the best eBGP-learned route from every other router in the AS or (2) all route attributes are compared across all routes (*i.e.*, it is possible to construct a total ordering over all routes) and every router receives at least one route in $\gamma(E)$, then every router r will ultimately select a route, $b_r \in \gamma(E)$, where E is the set of all eBGP-learned routes.*

Proof. Define $P_r \subseteq E$, the set of routes that router r learns (*i.e.*, $P_r = E_r \cup I_r$). Assume that some router r selects $b_r = \lambda_r(P_r) \notin \gamma(E)$. This property implies that $P_r \cap \gamma(E) = \phi$ (*i.e.*, that P_r contains *no* routes in $\gamma(E)$; otherwise, b_r would be better than all routes in $\gamma(E)$, which contradicts the definition of γ. But, if $P_r \cap \gamma(E) = \phi$, then the iBGP topology is such that r does not learn all routes, because at least one router $s \in R$ selects a route from $\gamma(E)$, and router r would have learned that route from s. If path visibility is satisfied and $b_r \notin \gamma(E)$, this also implies that some route attribute is not compared across all routes (*i.e.*, it is not possible to form a total ordering): otherwise, given a total ordering, if one router selects a route from $\gamma(E)$, then every router either learns that route and selects it, or selects its own route (which must be in $\gamma(E)$, by total ordering) and propagates that route. ■

Property 5.2 makes it possible to compute the route that each router r selects by applying λ_r to the set of *all* locally best routes, B (*i.e.*, $b_r = \lambda_r(B)$), thus eliminating other routes.

Unfortunately, this property is not guaranteed when determinism is violated and every router does not learn every eBGP-learned route. Consider the example shown in Figure 5-6. The network learns routes to some destination at routers X, Y, and Z that are equally good up to MED comparison. All three routers are clients of the route reflector RR. The routes at X and Y are learned from the same next-hop AS, and r_Y has a lower MED value. One might think that router X would never select route a, since, after all, it has a higher MED value than route b, but that is not the case in this figure: RR learns routes a, b, and c, and selects route c as its best route, because c has the shortest IGP path cost. As a result, X never learns route b.

When Property 5.2 is not satisfied, route prediction must essentially resort to simulation. The problem in this case is that it is impossible to know when activating any given router that it is safe to eliminate *any* route that it learns via eBGP. We discuss this problem in more detail in Section 5.7.

■ 5.6 Route Computation without Determinism

In this section, we present how to model path selection when the MED attribute is compared only across routes learned from the same AS, rather than across all routes for a destination prefix. MED prevents each router from having a total ordering over all possible candidate routes, so it is actually possible to have $b_r \in E_r$ without $b_r = \lambda_r(E_r)$. In Section 5.6.1, we describe this problem in more detail and describe why the simple approach

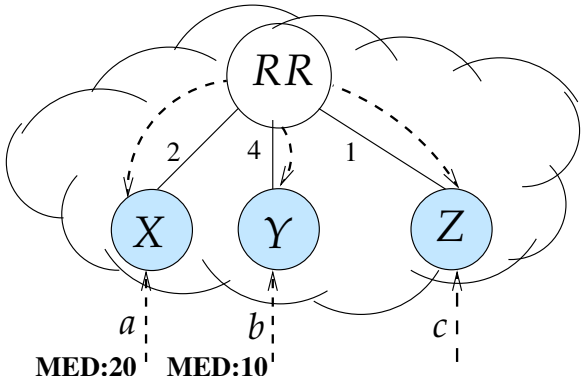


Figure 5-6: When an AS's iBGP topology uses route reflectors and MED, a router may not always select a route in $\gamma(E)$.

presented in Section 5.5 fails; then, we present an algorithm that accurately computes the outcome of BGP path selection when MED is compared only across routes from the same AS.

■ 5.6.1 Problems Introduced by MED

The algorithm from Section 5.5 assumes that each router's ranking between two routes is independent of whether other routes are present (*i.e.*, $\lambda_r(\{a,b\}) = a \Rightarrow \lambda_r(\{a,b,c\}) \neq b,\ \forall a,b,c$). When MED is only compared across routes from the same AS, the algorithm cannot simply select the locally best route at each router, because a router may ultimately select a best route that it learned via eBGP that was not its locally best route. This point has serious implications, because we can no longer assume that if a router selects an eBGP-learned route to a destination, that eBGP-learned route will be that router's locally best route; rather, the route that the router ultimately selects may be worse than the "best" route at that router when compared only against routes learned via eBGP at that router. Thus, the approach from Section 5.5, which computes b_r by taking the locally best route at each router from $\gamma(E)$, may not compute the correct result. Using the example in Figure 5-7, we explain why two seemingly-natural approaches to computing the routes do not work:

- *Local route elimination is not correct.* The algorithm in Figure 5-4 would first apply $\lambda_r(E_r)$ at each router. In Figure 5-7, given the choice between the two eBGP-learned routes a and c, router X prefers c, because c has a smaller router ID. Between routes a, c, and d (which it learns via Y), however, router X prefers route a, because route d eliminates route c due to its lower MED value. Thus, router X's preference between routes a and b depends on which route Y selects. The algorithm in Figure 5-4 would compute $\lambda_X(\{a,c\}) = c$ and $\lambda_Y(\{b,d\}) = d$ (resulting in $C = \{c,d\}$), and ultimately compute $B = \{d\}$ because d has a smaller MED value than c. In reality, though, router X would select route a over d, because a is an eBGP-learned route from a different neighboring AS.

- *Global route elimination is not correct.* It might also seem reasonable to apply γ globally, followed by applying λ_r locally at each router. In Figure 5-7, a global comparison of

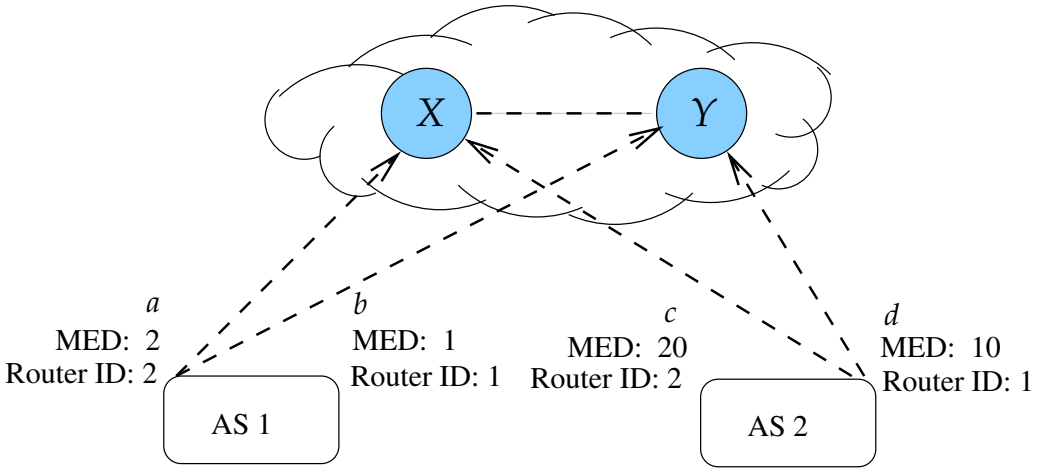


Figure 5-7: Interaction between MED and router ID in the BGP route selection process. X and Y are routers, each with direct eBGP sessions to ASes 1 and 2. a, b, c, and d are routes learned via eBGP.

Algorithm: Full Mesh, MED

```
SELECTBEST_EBGP_MED(E, R)
    // Eliminate all routes from C that
    // do not have highest local preference,
    // shortest AS path length, lowest origin type
    C ← σ(E)
    B_0 ← φ; i ← 0
    do
        B_{i+1} ← ∪_r λ_r(C_r ∪ B_i); i ← i + 1
    while B_{i+1} ≠ B_i
    return B_i
```

Figure 5-8: Algorithm for computing the best route at eBGP routers, assuming that MED is only compared across routes *from the same neighboring AS*.

the routes (*i.e.*, applying $\gamma(\{a, b, c, d\})$), would first eliminate a and c based on MED, and then router X would select route d (because d is preferred to b based on the router ID comparison applied at router Y). This conclusion is incorrect, because X would *always* prefer route a over route d, because a is learned via eBGP (step 5) and a and d are equally good up through step 4 (recall that a router does not compare the MEDs of routes with different next-hop ASes).

The crux of the problem is that the MED attribute makes it impossible to produce an ordering of the routes at X that is independent of the presence or absence of other routes.

■ 5.6.2 Algorithm: Full Mesh, MED

To correctly handle the interaction between the MED and router ID attributes, the algorithm emulates a message ordering that propagates the effects of MED on each router's best route. Figure 5-8 summarizes this algorithm. For this algorithm, we define a new

function, σ, which takes a set of routes and returns all routes equally good up through the first three steps of the BGP decision process (*i.e.*, local preference, AS path length, and origin type). When applied to the network in Figure 5-7, the algorithm starts with all routes in $\sigma(E)$ and proceeds as follows:

1. B_1 gets the locally best routes from X and Y: c and d, respectively. That is, $B_1 = \{c, d\}$.
2. On the second iteration, X compares the routes from C that it learns via eBGP, a and c, along with route d from B_1, so $\lambda_X(\{a, c, d\}) = a$. Similarly, $\lambda_Y(\{b, c, d\}) = d$. Thus, $B_2 = \{a, d\}$.
3. On the third iteration, the process repeats, and $B_3 = \{a, d\}$, at which point the algorithm terminates.

This algorithm computes the correct routing decision for each router: a at router X and d at router Y. At router Y, d is better than a (step 5), b (step 7) and c (step 4). At router X, a is better than d (step 5); a is not better than b, but this does not matter because router Y does not select b, and a is not better than c, but this does not matter because c is always worse than d (step 4).

Theorem 5.2 *When MED is compared only across routes from the same neighboring AS, the algorithm from Figure 5-8 accurately emulates the results of one activation sequence and message ordering for all routers that select an eBGP-learned route as their best route.*

Proof. Computing $\sigma(E)$ produces the set C, which is simply the set of eBGP-learned routes, E, minus the routes that could *never* be the best route at any router (*i.e.*, because they have a lower local preference, longer AS path length, or higher origin type). Because the iBGP topology forms a full mesh, as long as there is a route in E at *any* router that is better in the first three steps of the decision process, no router will select a route that is not in $\sigma(E)$. The remainder of the algorithm evaluates a routing system with the routes in $\sigma(E)$.

The remainder of the algorithm follows an activation sequence where each phase (or iteration of the loop) activates all of the routers simultaneously. The proof proceeds by induction. After the first iteration of the loop, $B_0 = \phi$ and $b_r = \lambda_r(C_r)$, where C_r is all of the routes learned at router r via eBGP with the highest local preference, shortest AS path length, and lowest origin type. By definition, $\lambda_r(C_r)$ returns each router's locally best route according to the BGP decision process, which is the same as that which the BGP decision process would select for each router after phase 1 of the activation sequence. In a network with a full mesh iBGP configuration, each router r then sends its locally best route, b_r, to every other router.

Suppose the algorithm correctly computes the outcome of the BGP decision process for the first i iterations of the activation sequence. Suppose that there is some router r for which the algorithm, at iteration $i + 1$, computes $b'_{r,i+1}$, the element in B_{i+1} that is the best route at router r, such that $b'_{r,i+1} \neq b_{r,i+1}$. Then, it must be the case that $b_{r,i+1} \notin C_r \cup B_i$; otherwise, $\lambda_r(C_r \cup B_i)$ would also have selected $b_{r,i+1}$. Either $b_{r,i+1}$ is an eBGP-learned route or it is an iBGP-learned route. If it is eBGP-learned, then it must be in C_r, as we previously established. If it is iBGP-learned, then it must be in B_i, because every iBGP-learned route is the best route of some other router in the AS. But if either $b_{r,i+1} \in C_r$ or $b_{r,i+1} \in B_i$, then $b_{r,i+1} \in C_r \cup B_i$, which is a contradiction.

The algorithm terminates when $B_i = B_{i+1}$; that is, when activating all of the routers in the AS does not cause any router to select a new best route and generate a new BGP update message. We have shown that the algorithm correctly predicts the outcome of BGP route selection after k iterations for any k. Further, we assumed that the routing system satisfies safety; that is, given a stable topology, it is guaranteed to converge to a path assignment where no router changes its best route. When the BGP routing system converges to this path assignment, no router changes the route it selects and, hence, no new routing messages are generated. Since, after i iterations, the algorithm correctly predicts the outcome of BGP route selection and the algorithm activates every router in the AS on every iteration, then it will terminate precisely when it has reached the BGP path assignment when no new BGP messages are generated (*i.e.*, the unique solution). ■

The algorithm in Figure 5-8 is correct, but it is not efficient: each iteration of the loop repeatedly considers routes that have been "eliminated" by other routes. A more efficient algorithm would eliminate routes from consideration at each iteration if we know that they could never be the best route at any router—such is the spirit of applying $\sigma(E)$ across the initial set of routes. Unfortunately, because the MED attribute is not comparable across all routes, it is possible for a route that is not in the set B_i to emerge in the set B_j for some $j > i$. We now formally define a condition under which routes may be eliminated, which will allow us to devise a more efficient prediction algorithm.

Lemma 5.1 *Suppose there exist two routes: (1) $s \in C_r$ at router r and (2) $t \in C_{r'}$ at router $r' \neq r$. If $t \in B_i$, $\lambda_r(s,t) = t$, and router r learns route t* (e.g., *as in a full mesh iBGP configuration), then $s \notin B_j \; \forall j > i$.*

Proof. First, note that as long as $t \in B_j$, then $s \notin B_j$ because route t is preferable to s Also note that because all routes in C are equally good up the MED comparison and eBGP-learned routes are preferred over iBGP-learned routes, we know that $\lambda_r(s,t) = t$ because MED(t) < MED(s). Now, suppose there exists some $j > i$ for which $t \notin B_j$. Call the best route at router r' at step i, $v = \lambda_{r'}(C_{r'}) \neq t$; again, we know that MED(v) < MED(t). But this means that MED(v) < MED(s), $\lambda_r(s,v) = v$, and, thus, $s \notin B_j$. ■

We can use this result to devise a more efficient route prediction algorithm that eliminates, at every iteration, a router's locally best route if it has a higher MED value (and same next-hop AS) than some other router's locally best route. This algorithm is described in Figure 5-9 and shown conceptually in Figure 5-10; it can also be thought of in terms of an activation sequence: (1) each router learns routes via eBGP, selects a locally best route, and readvertises via iBGP; (2) each router compares its locally best route with all other routes learned via iBGP, and *eliminates* its own locally best route from the system if it is worse than some other locally best route at another router; (3) the system is restarted (from phase 1) with the eliminated routes removed. This algorithm is computationally more efficient than the one in Figure 5-8; we now analyze its running time complexity.

Computational Complexity. Understanding the running time of the algorithm in Figure 5-9 is easiest when we consider the implementation of the algorithm shown in Figure 5-10. In this figure, the eBGP-learned routes at each router are represented as a stack and are sorted *locally* (*i.e.*, compared only to other routes learned at the same router). The

Algorithm: Full Mesh, MED (Efficient Algorithm)

SELECTBEST_EBGP_MED(E, R)

// Eliminate all routes from C that
// do not have highest local preference,
// shortest AS path length, lowest origin type
$C \leftarrow \sigma(E)$

// Keep track of the best routes at each router.
do
  $B \leftarrow \cup_r \lambda_r(C_r)$
  $L \leftarrow B \setminus \gamma(B)$
  $C \leftarrow C \setminus L$
while $L \neq \phi$

Figure 5-9: Computationally efficient algorithm for computing the best route at eBGP routers, assuming that MED is only compared across routes *from the same AS* (*i.e.*, that there is no total ordering of routes).

top of the stack represents the best route learned at that router; the route that is second from the top is the second best route, and so forth. Then, the algorithm from Figure 5-9 can be interpreted as follows:

- $B \leftarrow \cup_r \lambda_r(C_r)$ is the union of all of the elements at the top of the stack and does not need to be computed explicitly, assuming each stack is sorted. The complexity of sorting N routes distributed across $|R|$ stacks is $O(N \log N)$. Each of N routes may be inserted into as many as $|R|$ stacks, so the complexity of this step is $O(N \log N + N|R|)$.
- $L \leftarrow B \setminus \gamma(B)$ marks a route at the top of a stack if that route is worse than any route at the top of another stack, according to the first four steps of the BGP decision process. This process takes at most two scans of the routes at the top of the $|R|$ stacks, so the running time is $O(|R|)$.
- $C \leftarrow C \setminus L$ "pops" the marked routes from the top of the stacks, where appropriate. This process requires a single scan through $|R|$ stacks and at most $|R|$ pop operations, so the running time is $O(|R|)$.

In the worst case, the above three steps repeat until $N - 1$ routes are popped from the stacks, and each iteration only pops a single route. Thus, in the worst case, the running time for the algorithm is $O(N \log N + N|R|)$.

■ 5.7 Route Computation without Full Visibility

A full mesh iBGP topology does not scale to large networks because a network of $|R|$ routers requires $O(|R|^2)$ iBGP sessions. Network operators use a technique called *route reflection*, which improves scalability by introducing hierarchy but complicates route prediction. First, we define an iBGP signaling topology, expound on problems introduced by route reflection, and describe constraints on iBGP configuration that must hold for modeling to be possible. Next, we propose an algorithm that efficiently computes the outcome

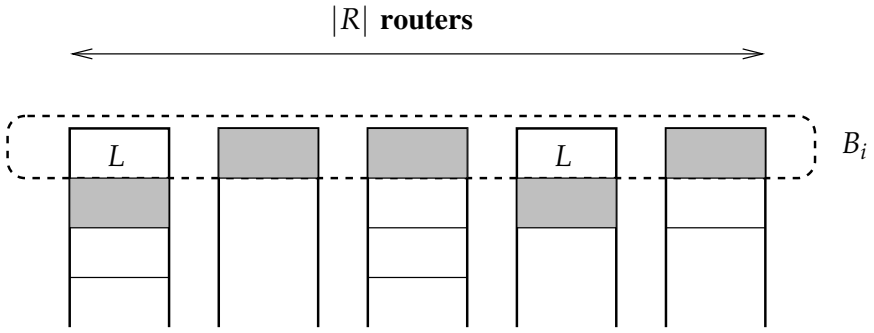


Figure 5-10: Implementation of the route computation algorithm from Figure 5-9. Each stack represents one of $|R|$ total routers, and each stack element represents one of L routes. The top elements of the $|R|$ stacks represent B_i, the elements marked L represent routes that are worse than the routes at the top of the remaining stacks according to the first four steps of the decision process (*i.e.*, local preference, AS path length, origin type, MED), and the shaded routes represent B_{i+1}. The algorithm terminates when no routes are marked L.

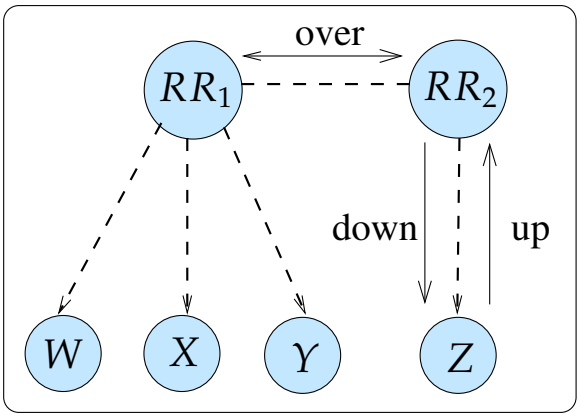


Figure 5-11: Example iBGP signaling graph.

of BGP path selection in a network with route reflection; we then present a minor modification to the algorithm that is necessary if MED is only compared across routes from the same neighboring AS.

■ 5.7.1 Problems Introduced by Route Reflection

A router does not normally forward iBGP-learned routes over other iBGP sessions, but it can be configured as a *route reflector* (RR), which forwards routes learned from one of its route-reflector clients to its other clients. The routers in an AS form a directed graph, $G = (R, S)$, of iBGP sessions called a *signaling graph*. Each edge $a = (u, v) \in S$ where $u, v \in R$ corresponds to an iBGP session between a pair of routers. We then define three classes of edges: (1) $a \in$ **down** if v is a route-reflector client of u; (2) $a \in$ **up** if u is a route-reflector client of v; and (3) $a \in$ **over** if u and v have a regular iBGP session between them. Figure 5-11 shows an example signaling graph. In a full-mesh configuration, every eBGP-speaking router has an edge in **over** with every other router in the AS, and both the **up** and **down**

sets are empty.

Previous work has shown that iBGP satisfies safety as long as the structure of the signaling graph satisfies certain sufficient conditions [61]. Accordingly, we refine Constraint 5.2 in terms of these sufficient conditions to guarantee that an iBGP topology with route reflection satisfies safety (at least when the MED attribute is not used or compared across all routes):

Constraint 5.4 *(1)* $\forall\ u, v, w \in R$, $((u, v) \in$ **down** *and* $(u, w) \notin$ **down**$) \Rightarrow \lambda_u(\{\rho_v, \rho_w\}) = \rho_v$, *where* ρ_v *represents any route learned from* v *and* ρ_w *is any route from* w*; and (2) the edges in* **up** *are acyclic.*

Part (a) is satisfied when routers do not change the attributes of iBGP-learned routes and each router has a lower IGP path cost to its clients than to other routers. The common practices of applying import policies only on eBGP sessions and placing route reflectors and their clients in the same point-of-presence (*i.e.*, "PoP") ensure that these conditions hold. Part (b) states that if a is a route reflector for b, and b is a route reflector for c, then c is not a route reflector for a, consistent with the notion of a route-reflector *hierarchy* (rather than an arbitrary signaling graph).

Even a route reflector configuration that converges can wreak havoc on the algorithms from Sections 5.5 and 5.6. A route reflector hides information by advertising only a *single best route to its iBGP neighbors*. For example, in Figure 5-11, if W and Z have eBGP-learned routes, router Y learns a single route from its route reflector RR_1. Suppose that RR_1 selects the eBGP route advertised by Z. Then, Y would pick Z's route as well, even if Y would have preferred W's route over Z's route. Note that Y makes a different routing decision than it would if it could select its best route from all the eBGP routes (*i.e.*, from both W and Z). In large networks, route reflection reduces the number of routing messages and iBGP sessions, which helps scalability, but it complicates route prediction in the following ways:

1. A router will not typically learn every route that is equally good up through the first four steps of the decision process. That is, it is possible (and likely) that some routers will not learn every route in $\gamma(B)$. In Section 5.7.2, we describe an algorithm that handles this case.
2. If a network uses route reflectors, *and* MED is only compared across routes from the same AS, the routes that some routers ultimately select may be *worse* than some eBGP-learned routes, according to the first four steps of the decision process. That is, it may be the case that $b_r \notin \gamma(E)$ for some router r. This characteristic creates problems not only for efficient route prediction, but also for safety. We discuss this case in Section 5.7.3.

■ 5.7.2 Algorithm: Route Reflection, No MED

Route reflection obviates the need for routers in an AS to form a full mesh topology, but it also means that some routers may not learn all routes in $\gamma(B)$. This artifact has two implications. First, the algorithm cannot simply assign a non-eBGP-speaking router the route from the "closest" eBGP-speaking router, because the former router may never learn the route. Thus, applying $b_r \leftarrow \lambda_r(B)$ may not always be correct. For example, consider

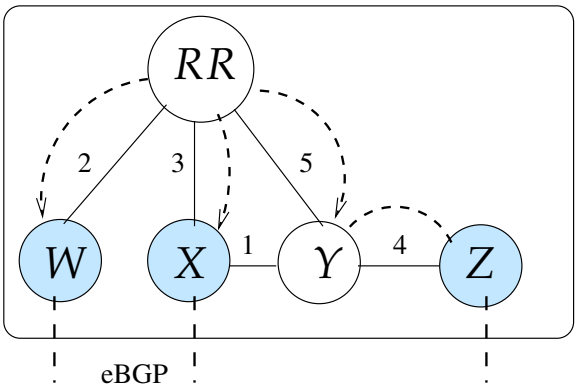


Figure 5-12: When an AS's iBGP topology uses route reflectors, a router may not always discover the route corresponding to its closest egress router.

the network shown in Figure 5-12. W, X, and Y are clients of route reflector RR, and Z is a regular iBGP peer of Y. X and Y have a short IGP path between them, but they are *not* directly connected by an iBGP session. Routers W, X, and Z have eBGP routes that are equally good through the first four steps of the route selection process, and have thus selected their own eBGP-learned routes. In this network, Y's closest egress point is X, but Y selects W, because RR's closest egress router is W.

Second, often there is *no consistent ranking of possible egress routers* from some non-eBGP-speaking router; in other words, *egress determinism* (Definition 3.13) is violated. For example, in Figure 5-12, RR prefers egress router W because its IGP path cost to W is the shortest. Router Y's preferences over possible egress routes depends on the presence or absence of other routes. If the AS learns routes for some destination via eBGP sessions at routers X and Z, then router Y prefers using X as an egress router. On the other hand, if the AS learned routes at W, X, and Z, then Y prefers using Z, which implies that Y prefers egress Z over X and is inconsistent with Y's choice when only X and Z are available egress routers.

To account for the fact that some routes are not visible at some routers, we design an algorithm that emulates a certain activation sequence, making route assignments at each router where possible and propagating the effects of these decisions to other routers, without ever having to revisit any assignment. This algorithm is shown in Figure 5-13. The algorithm first activates the routers from the bottom of the route reflector hierarchy upwards, which guarantees that each router selects a *down* route where possible, as required by Constraint 5.4(a). Because the algorithm moves upwards from the bottom of the hierarchy, it performs computations for each route reflector after all of the routes from its clients become known; computations for these routers never need to be revisited, since, by Constraint 5.4, a router always prefers routes from its "children" (*i.e.*, clients) over routes from its peers or parents. Visiting the routers in the *down* direction ensures that the algorithm performs computations for the remaining routers using all available routes from the **up** and **over** sets. The algorithm defines two partial orderings of the routers based on the elements of the **up** and **down** sets. We can define these two partial orderings because Constraint 5.4(b) requires that the signaling graph does not have any cycles of these edges,

Algorithm: Route Reflection, No MED

SELECTBEST_EBGP_RR(E, R)

// Proceed up the hierarchy, assigning best routes.
// Find a router for which all children are activated.
$A \leftarrow \phi$
while $\exists r \in R$ s.t. $r \notin A$ and $c \in A \; \forall c \in \text{DOWN}(r)$
  $I_r \leftarrow \cup_{c \in \text{DOWN}(r)} b_c$
  $b_r \leftarrow \lambda_r(I_r \cup E_r)$
  $A \leftarrow A \cup r$

// Proceed down the hierarchy.
// Find a router for which all parents are activated.
$A \leftarrow \phi$
while $\exists r \in R$ s.t. $r \notin A$ and $c \in A \; \forall c \in \text{UP}(r)$
  $I_r \leftarrow \cup_{c \in \text{UP}(r) \cup \text{OVER}(r)} b_c$
  $b_r \leftarrow \lambda_r(I_r \cup b_r)$
  $A \leftarrow A \cup r$

Figure 5-13: Algorithm for computing the best route at each router in a network with route reflection but no MED.

so each partial ordering must have a top and bottom element.

Applying this algorithm to the example in Figure 5-12, the shaded routers select best routes in the first step, because each of those routers is at the bottom of the hierarchy and, thus, all of their neighbors in **down** have been activated (because they have none). Y is activated, but it does not select a route at this point because it has no neighbors in **down**. Because these four routers are at the same level in the hierarchy, they can be activated in any order. Then RR is activated; it applies $\lambda_{RR}(\{r_W, r_X\})$ and selects r_W because it has the smallest IGP path cost. The routers are all activated again in the downward direction; Y receives r_W from RR and compares it with r_Z, which is its best route to the destination. X and Z also receive r_W but continue to select their own route, because λ_r prefers eBGP routes over iBGP routes. We now prove that the algorithm shown in Figure 5-13 is correct.

Theorem 5.3 *If each router can form a total ordering over the set of all candidate routes, then the algorithm in Figure 5-13 correctly computes the outcome of the BGP decision process, b_r, for all routers $r \in R$.*

Proof. Assume that some router r selects a route, b_r, that is different from the route assigned by the algorithm in Figure 5-13, b'_r. The mismatch can occur in one of two cases: (1) when b_r is learned from a session in **down**, or (2) when b_r is learned from a session not in **down** (*i.e.*, in either **up** or **over**).

Consider Case 1, where b_r is learned from a session in **down**. Call b'_r the first case of an incorrect computation (*i.e.*, the algorithm has correctly computed the best route for all routers below r in the hierarchy); because we examine the first such mismatch, I_r is correct. If b'_r is also in **down**, then $b'_r = \lambda_r(I_r \cup E_r)$ when the algorithm proceeds up the hierarchy, which implies that b'_r is better than b_r according to the BGP decision process, and r would

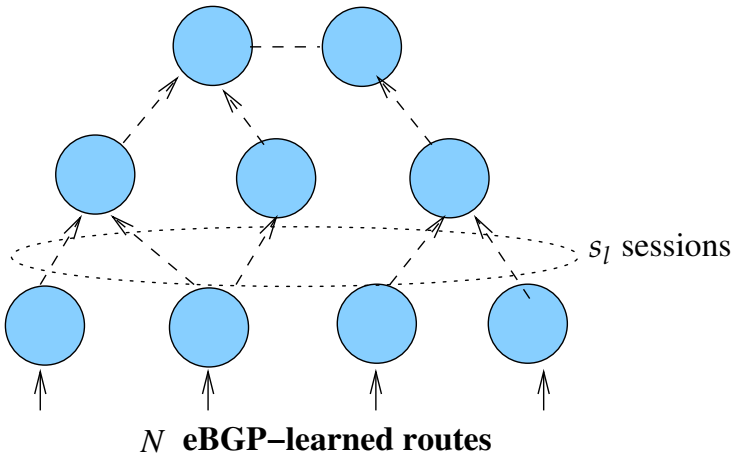


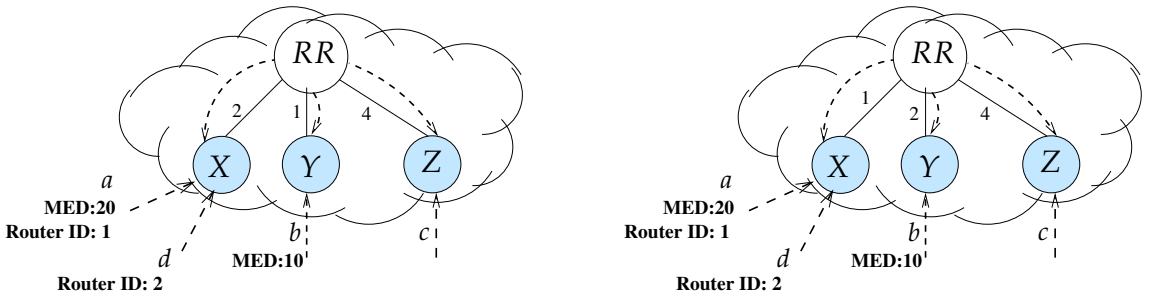
Figure 5-14: Running time analysis of an iBGP graph walk for the algorithm in Figure 5-13.

have actually selected b'_r. If b'_r is in **up** or **over**, then it must have been the case that it was better, according to the BGP decision process, than the displaced route b_r in **down**. But then, by definition of λ_r, router r would have also selected b'_r in BGP. Thus, the algorithm correctly computes b_r for all routers r that select a best route from **down**.

Consider Case 2, where b_r is learned from a session in **up** or **over**. From the first half of the proof, we know that the algorithm correctly computes b_r for all routers that select a route from **down**, so call b'_r the first instance of a mismatch for some router that selects a best route from **up** or **over** (*i.e.*, the algorithm correctly assigns b_r for all routers higher in the hierarchy than r). Again, because we consider the first such mismatch, we know that I_r is correct. If the route that the algorithm selects, b'_r, is in **down**, then, by Constraint 5.4(a), BGP could not have selected b_r, so we have a contradiction. If both b_r and b'_r are learned from sessions in **up** and **over**, then both are in I_r, and, according to the $\lambda_r(I_r \cup b_r)$ step in the algorithm and by definition of λ_r, both the algorithm and the BGP decision process would select the same route. ■

This theorem relates to one from earlier work [47] on sufficient conditions for stable BGP routing *at the AS level*; this work provides a constructive proof showing that the sufficient conditions guarantee safety. In subsequent work, Griffin *et al.* discovered that the sufficient conditions for stable eBGP routing were analogous to those for stable iBGP routing with route reflection [61]. The algorithm from this section applies the iBGP analog of the constructive proof from the work on stable interdomain routing to develop an algorithm for *computing* that stable path assignment.

Computational Complexity. This algorithm traverses the route reflector hierarchy exactly twice. The running time of this algorithm is $O(N + |S|)$, where N is the number of eBGP-learned routes, and $|S|$ is the number of iBGP sessions. To see why this is the case, consider the l-level route reflector hierarchy pictured in Figure 5-14. Starting from the bottom of the hierarchy, the algorithm must perform comparisons over N routes to determine the routes that the M routers at the bottom of the hierarchy select (the number of routers at the bottom of the hierarchy is inconsequential: these comparisons can be performed by



(a) When Y is closer to RR than X, the routing system satisfies safety.

(b) When X is closer to RR than Y, the routing system violates safety and the algorithm in Figure 5-13 is incorrect.

Figure 5-15: A BGP configuration where the algorithm in Figure 5-13 may the incorrect result, depending on the IGP topology and the MED attributes of the routes received via eBGP.

constructing a subset of M routes from the original N routes, which can be performed in a single scan of the N routes). The algorithm then propagates the selection of these M routes to the next level of the hierarchy, where s_l comparisons must be performed across the routers at the next highest level, where s_l is the number of iBGP sessions at level l. Repeating this process up the hierarchy yields a total running time of $O(N + |S|)$.

Recall from Section 5.5 that the running time for the algorithm in the case of full-mesh iBGP, was $O(N + |R|^2)$, or $O(N + |S|)$. Note that the algorithm for the case with route reflection has the *same* running time complexity as before; the running time for computing the outcome of BGP route selection is no more complex, even though the process for computing the outcome is more involved. In an iBGP topology with route reflection, the number of sessions, $|S|$, will actually be less than $|R|^2$; thus, the running time of the algorithm benefits from the fact that route reflectors reduce the number of sessions in the iBGP topology.

■ 5.7.3 Algorithm: Route Reflection, MED

When a network uses both route reflection and MED, the graph walk algorithm in Figure 5-13 no longer works, because it relies on the fact that all routers will ultimately select a route in $\gamma(E)$. In a network with route reflection *and* MED, this is not always true, because when a router selects a locally best route, a route with a lower MED value might not be visible to that router. As a result, some router in the AS might select an eBGP-learned route that is *worse*, according to the first four steps of the BGP route selection process, than eBGP-learned routes selected by other routers! Figure 5-15 shows an example of exactly this scenario.

Note that applying the algorithm from Figure 5-13 does not always correctly compute the outcome of the BGP decision process. Consider the operation of the algorithm from Figure 5-13 on the topology and route announcements shown in Figure 5-15(a). Proceeding up the hierarchy: (1) routers X, Y, and Z would select routes a, b, and c, respectively; (2) RR selects route b because b has a lower MED value than a and a shorter IGP path to

the egress than c. Proceeding down the hierarchy, X selects b because it has a lower MED value than a. At this point, the algorithm in Figure 5-13 would terminate in the case of Figure 5-15(a), but, in fact, depending on the IGP topology X's selection of d could have caused RR to select a new best route. Suppose, instead, that the IGP topology were such that RR were closer to X than to Y, as in Figure 5-15(b). In this case, proceeding up the route reflector hierarchy a second time would cause RR to change its selected route from b to d; subsequently proceeding down the hierarchy would cause X to change its selected route from d to a. In fact, as described in a similar example in Section 3.4, BGP does not satisfy safety in this example—therefore, *no* number of progressions up and down the iBGP hierarchy would cause the algorithm to predict the correct outcome.

In this situation, any router in the AS might ultimately select a route that is not in $\gamma(E)$; as a result, the route prediction algorithm cannot eliminate a route from the set of candidate routes C_r at any router r, as was done in the case where determinism did not hold but every router was guaranteed to learn every eBGP-learned route (Section 5.6, Figure 5-9). As we have seen, the fact that a router may select a route that is not in $\gamma(E)$ as its best route, the algorithm (and BGP route selection, for that matter) is no longer guaranteed to terminate.

It might initially seem reasonable to impose constraints on the iBGP and IGP topologies that guarantee safety and can easily be checked with a tool like *rcc* (Chapter 4). Unfortunately, as the example in Figure 5-15 shows, any condition that guarantees safety would require knowledge of the MED attributes of every eBGP-learned route to a destination, not just the iBGP and IGP topologies. Further, the simplicity of this example demonstrates that any condition that guarantees safety for any combination of eBGP routes would be overly restrictive (*i.e.*, it would essentially require not using route reflectors). Thus, in the case where a BGP configuration uses route reflection and only compares the MED attribute across routes from the same AS, the most efficient algorithm for determining the outcome of BGP route selection (and detecting safety violations) is actually a simulator. In other words, there are no conditions on the topology that can be enforced to guarantee that an algorithm would never have to visit each router in the AS more than once, or even that BGP would satisfy safety.

■ 5.8 Implementation: The Routing Sandbox

In this section, we describe a prototype, the *routing sandbox*, that incorporates BGP route prediction algorithms described in this chapter. Our current prototype separates the computation of egress routers for a given destination from the assignment of other routers to those egress routers. *This separation of functionality requires that $b_r \in \gamma(E)$, which does not hold when* both *MED and route reflection are used (as shown in Section 5.7.3).*

■ 5.8.1 Design Overview

We now highlight the high-level design of the prototype, shown in Figure 5-16. We briefly describe the necessary inputs for driving the prototype, the decomposition of functionality into three distinct modules and the relationships of those modules to the algorithms described in this chapter, and optimizations that reduce computational complexity.

The prototype has three inputs:

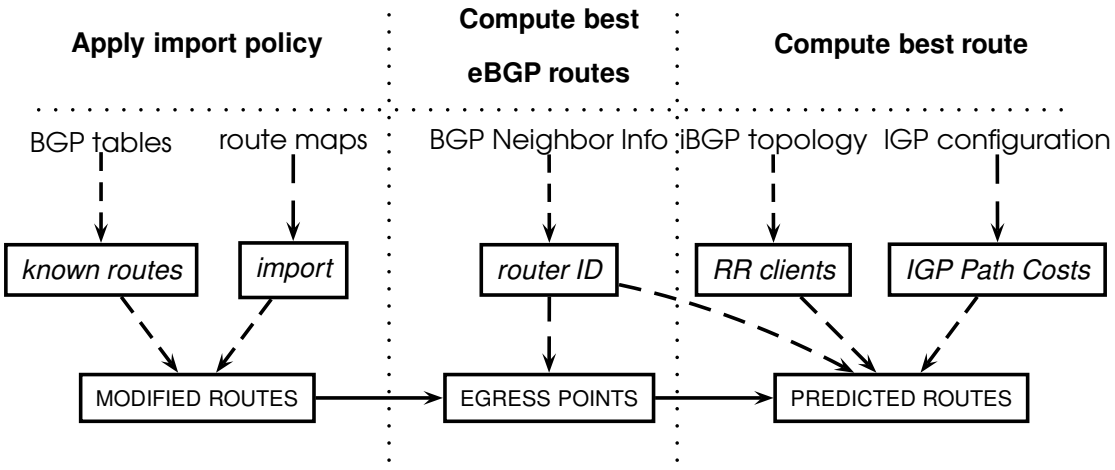


Figure 5-16: Data flow in the prototype. Fonts specify raw inputs**,** *input tables,* **and** DERIVED TABLES**. In practice, operators might collect raw inputs once a day.**

- *BGP routing tables:* The BGP tables for the eBGP-speaking routers provide the first stage of the algorithm with a snapshot of the routes advertised by neighboring ASes. We ignore the router's current view of the best route and the current setting of the local preference attribute, since these relate to the existing network configuration rather than the scenarios we might want to emulate.

- *Router configuration files:* The configuration files are used to (1) determine the import policies (*i.e.*, "route maps", as described in Section 2.3.2) for each eBGP session, (2) determine the iBGP signaling graph, and (3) compute the IGP path costs between each pair of routers. The import policies are used to manipulate attributes of the eBGP routes in the first stage of the algorithm, and the iBGP and IGP information are needed for the third stage.

- *BGP neighbor information:* Because BGP route selection (Table 2-2) depends on the router ID associated with the BGP session announcing the route, our algorithms require knowing the router ID associated with each BGP session. The second stage uses the router IDs of the *eBGP* sessions, while the third stage uses the router IDs for the *iBGP* sessions.

We note that these inputs are easy to either obtain or approximate. First, a network operator can capture all of the necessary data with `telnet` or `ssh` access to each router. Second, many aspects of the input data do not change very often; as such, the prototype is useful even if all of the input data is collected infrequently (*e.g.*, once a day). Finally, because certain inputs can be approximated (*e.g.*, a BGP session's router ID is typically the loopback IP address[3] of the router on the opposite end of the session), the prototype can be effective even with limited input.

The prototype uses a database back-end, which provides efficient access to small subsets of the configuration data and routes and also stores intermediate results, which allow

[3] A router's *loopback address* is an IP address on the router that is reachable via any of the router's interfaces.

us to validate each part of the algorithm separately. Figure 5-16 summarizes how the prototype uses the inputs and intermediate results to generate a table of predicted routes. The three modules shown in Figure 2 correspond to the first two stages from Section 5.3.2; assuming that $b_r \in \gamma(E)$ allows us to break the second stage into two simpler modules. The prototype performs three operations:

Applying import policy to eBGP-learned routes: This operation corresponds to the first step described in Section 5.3.2. Each row of the *import* table specifies how a particular set of rows in the *known routes* table should be modified; the prototype performs the actual modifications on the MODIFIED ROUTES table. For each row in the *import* table, the first operation applies the policy by (1) finding the appropriate routes by selecting the set of routes learned at the corresponding router on that BGP session that match the specified AS path regular expression and (2) setting the other attributes (*e.g.*, local preference) according to the values specified in that row of the *import* table.

Computing the egress routers for a destination: This operation generates the set of best eBGP-learned routes, B, using the algorithm from Section 5.6.2, corresponding to the first half of stage 2 in Section 5.3.2. This part of the algorithm performs "select" statements on the MODIFIED ROUTES table to successively refine the set of candidate routes. The *router ID* table contains the router ID for every BGP session at each router, which is needed for step 7 of the decision process. As the method from Section 5.5 marks "best" routes, these routes are inserted into the EGRESS POINTS table for use by the third operation.

Computing the predicted routes: This operation uses the iBGP signaling graph, IGP path costs, and algorithm from Section 5.7.2 to determine the best BGP route for each prefix at each router. The module uses the iBGP signaling graph to determine which routes are advertised to each router, the IGP path costs between each pair of routers to determine the closest eBGP-speaking router to each ingress router (used in step 6 of the decision process), and the router ID of each iBGP session (step 7) to determine the egress router that each ingress router will select. The *RR clients* table represents the iBGP signaling graph and *IGP path costs* represents the shortest IGP path between each pair of routers in the AS. Each row of *RR clients* specifies a route reflector client for a particular cluster; this provides the partial ordering needed by the algorithm. When applying the IGP tiebreaking step at an ingress router, *IGP path costs* is used to determine the egress router with the shortest IGP path.

To ensure that the prototype operates on reasonable timescales, even with a large number of routes and eBGP sessions, we made the following optimizations: (1) because many routes have the same AS path attribute, store the AS paths in a separate table to accelerate lookups based on AS path regular expressions; (2) because many prefixes are advertised in exactly the same manner (*i.e.*, at the same set of egress routers and with the same attributes), compute the best BGP routes only once for each *group of prefixes*; and (3) upon an incremental policy change, only recompute the routes for prefixes affected by that change.

In the next two sections, we analyze the performance and correctness of our algorithms on real data from a large tier-1 ISP. The analysis focuses on a snapshot of the network state during early morning (EST) on February 4, 2003. We validate the prediction algorithm for the 91,554 prefixes whose eBGP routes are learned at peering points to other large providers, since we have routing tables from all of these locations; we excluded prefixes that were learned at other routers. (Recall that the prediction algorithm relies on knowing

all of the potential egress routers where routes to a prefix are learned.) The initial BGP routing data consists of 1,620,061 eBGP-learned routes with 43,434 distinct AS paths. We apply the tool to these inputs and check whether it produces the same answers that the operational routers selected. In addition to collecting BGP routing tables from the peering routers (where the eBGP routes are learned), we also collect BGP tables from several route reflectors and access routers to verify the results.

5.8.2 Performance Evaluation

In this section, we evaluate the performance of our implementation of the BGP emulator. We do not attempt to perform a complete performance analysis of the prototype. Rather, we conduct experiments that demonstrate the *practicality* of the prediction algorithm.

While our evaluation is preliminary, our performance tests demonstrate that the emulator can operate on timescales that could allow an operator to use a BGP emulator based on our algorithms in a practical setting. Our evaluation demonstrates the following points:

- The emulator computes the best routes for one prefix throughout a large tier-1 ISP network in about one second. Although predicting the best route for all prefixes at all routers in such a network takes several hours, this type of computation does not need to be performed all that frequently in practice.
- Exploiting commonalities among route advertisements to eliminate redundant computation reduces the running time of the emulator by approximately 50%.
- Using the emulator to evaluate the effects of an incremental change to router configuration typically takes only a few seconds. Thus, we believe that the emulator can be practical for tasks such as interdomain traffic engineering.

After briefly discussing the evaluation framework, we examine the emulator's performance. First, we discuss the emulator's performance when it computes the routes for *every* prefix in the routing table from scratch, without any performance optimizations. We then examine how insights from the BGP decision process and previous measurement studies can improve performance. Finally, we describe how the emulator can quickly predict the effects of incremental configuration changes.

Evaluation Framework

We ran the emulator on a Sun Fire 15000 with 192 GB of RAM and 48 900 MHz Ultrasparc-III Copper processors. Because this is a time-shared machine, we ran each of our experiments several times to ensure the accuracy of our measurements. Except where noted, the prototype used only two 900 MHz processors (one for the database process and one for the emulator itself); the combined memory footprint of the database process and the emulator never exceeded 50 MB. Because the emulator did not use more resources than a standard PC, the results of our evaluation should reasonably reflect the emulator's performance on commodity hardware.

Because the emulator's running time depends on many interdependent factors—including the number of neighbor ASes, the number of eBGP sessions, the number of prefixes, and the number of routers—running independent benchmarks for each of these

factors with *realistic routing and configuration data* is extremely difficult. For example, it is difficult to run an experiment that controls every other factor that affects running time while varying the number of eBGP sessions. Similarly, determining a precise running time for the emulator to process an incremental configuration change is difficult because the running time depends on how many routes must be recomputed as a result of that change.

Rather than isolating individual factors that affect performance, which is difficult and may not accurately reflect realistic network conditions, we evaluated the BGP emulator's running time using the actual routing tables and configuration data from a large tier-1 ISP with several hundred routers; we present absolute performance numbers, as well as appropriate averages, to give a rough estimate of the emulator's running time for an arbitrary-sized network. We also use the averages to estimate the running time for computing the effects of incremental routing changes. Most networks have fewer prefixes in their routing tables, fewer routers, and fewer BGP sessions per router. Therefore, the running times we report can be considered conservative: the emulator should have a shorter running time for most other networks.

Route Prediction from Scratch

To get a baseline performance estimate for the algorithm, we first ran the emulator without any performance optimizations. Before the emulator can begin executing the route prediction algorithm, it must load the input data into the database. Loading the configuration data has three separate steps: (1) parsing and loading the routing tables, (2) parsing and loading the import policies, (3) building the database indexes. The first two steps can be parallelized by router since the tables and configuration for each router can be parsed and loaded independently. When loading each routing table in sequence, the prototype parsed and loaded all 1,620,061 eBGP-learned routes from a large tier-1 ISP in just over 90 minutes, at a rate of about 280 routes per second. When loading up to 20 tables in parallel, the emulator finished loading the routing tables in about 520 seconds. The speed of this operation is not critical, since it is likely only performed once per day. The time to parse and load the import policies and router ID information was negligible: the emulator parsed and loaded this information in just over 1 second.

Once all of the data was parsed and loaded into the database, the emulator applied the 486 import policy operations to the eBGP-learned routes in a total of 1,576 seconds, or about 0.31 operations per second (it does not make sense to give a per-prefix performance number for this module, since one import policy applies to many prefixes). The second module computed the set of best eBGP routes at a rate of about 3 prefixes per second, and the third module computed the best route for each prefix and ingress router at approximately 7.3 milliseconds per prefix *per router*.

Although the emulator takes a total of about 5 hours to compute all routes for all routers in a large ISP network, running the emulator is likely to be much faster in most cases. First, depending on the task, a network operator may not need to perform route prediction for every prefix. For example, it is well known that the majority of network traffic is destined for a relatively small fraction of the total prefixes [32]; thus, a network operator would likely to be able to perform effective traffic engineering by emulating route selection for only a small number of prefixes. Similarly, a network operator who is debugging a reachability problem to a single prefix or small group of prefixes will only need to run the

emulator for those prefixes. Second, several performance optimizations can significantly improve the efficiency of the emulator, as we discuss next.

Performance Optimizations

To ensure that the emulator operates on reasonable timescales, even with a large number of routes and eBGP sessions, we designed the emulator around the inherent structure of the input data. In particular, we make three observations that inspire aspects of the design: (1) many BGP routes have the same AS path attribute; (2) neighboring ASes often advertise a large group of prefixes with the same attributes across all eBGP sessions, and they often advertise a large group of prefixes to the same set of eBGP-speaking routers; and (3) incremental router configuration changes typically only affect a small number of routes. These observations allow the BGP emulator to scale to a large number of prefixes and eBGP sessions and halve the emulator's running time.

Store and manipulate each unique AS path only once: Modifying the eBGP-learned routes according to import policies potentially involves sequentially scanning each router's BGP routing table for routes whose AS paths match a given regular expression; performing this operation once per import policy would involve many table scans. Fortunately, many eBGP-learned routes have the same AS path: in our BGP routing tables, each unique AS path appears in twenty eBGP-learned routes, on average. We exploit this observation by having the *known routes* and MODIFIED ROUTES tables store a *pointer* (*i.e.*, a foreign key) into a separate table that contains the distinct AS paths. This level of indirection significantly reduces the overhead of the first module, which repeatedly modifies attributes for some set of routes that match a certain AS path regular expression. The first module (1) searches the relatively small AS path table to find the AS path pointers associated with a regular expression and (2) selects the subset of table entries that must be modified by selecting the entries that have those AS path pointers (on which the table is indexed). By operating on a table of 45,000 AS paths instead of more than 1 million eBGP-learned routes, the first module can apply 1.02 import policy operations per second—more than a factor of 3 improvement over the 0.31 operations per second reported in Section 5.8.2.

Group prefixes with the same eBGP routing choices: The emulator must compute the set of best eBGP-learned routes for each prefix; because an Internet routing table often has more than 100,000 prefixes, performing this prediction once per prefix could be computationally expensive. Fortunately, because a neighboring AS typically advertises a large group of prefixes over the same set of peering sessions, *many prefixes are advertised in exactly the same fashion across all eBGP sessions* with neighboring ASes [32]. This pattern of advertisements typically happens when a single institution announces several prefixes from a single location or a single peer advertises various prefixes with the same AS path length. As such, for many prefixes, the process for computing the set of best routes is exactly the same. For example, if two prefixes have an identical set of MODIFIED ROUTES (*i.e.*, the same attributes for the route from each eBGP neighbor), the second module of the emulator would produce the same egress set. In fact, this is true as long as the two prefixes have routes with the same AS path *length* from each neighbor, since the BGP decision process only considers the length of the path. To exploit this observation, the *known routes* and MODIFIED ROUTES tables store the *length* of the AS path, along with the pointer to the table of unique AS paths. We group prefixes that have routes with the same AS path

length, local preference, origin type, and MED, reducing the total number of predictions from 91,554 (*i.e.*, one per prefix) to 8,291 (*i.e.*, one per group of prefixes). Identifying these groups of prefixes required 1,420 seconds (this time is proportional to the total number of eBGP-learned routes). After grouping the prefixes, the computation in the second module requires only 15,753 seconds, rather than the 30,212 seconds needed when performing the computation separately for each prefix. The speed-up is somewhat limited because of the overhead required to determine whether a new computation can be avoided.

Group prefixes with the same egress set: The best route that the emulator predicts at a particular ingress router ultimately depends on the set of routers in the egress set for that prefix. In theory, the number of distinct sets of egress routers is exponential in the number of routers. Fortunately, because many prefixes are advertised in exactly the same fashion, and because an AS usually applies the same local policies to manipulate and select these routes, many prefixes have exactly the same set of egress routers; the emulator can thus select the best route for each group of prefixes with the same egress set, rather than for each prefix. In our routing data, the 91,554 prefixes have only 290 distinct egress sets. We exploit this observation by applying the algorithm in Section 5.7 only once per ingress router and set of egress routers, rather than once for each ingress router and prefix. Determining whether a prediction has already been computed for an ingress router and a set of egress routers requires an additional database query. Despite this additional overhead, this optimization reduces the running time of the third module from 7.3 to 4.3 milliseconds per prefix per ingress router.

Compute small differences after incremental changes: We envision that network operators would use the BGP emulator as a traffic engineering tool, in order to predict how a configuration change would affect the flow of traffic. These kinds of configuration changes typically *only affect some small subset of the total number of routes*. Thus, in cases of incremental configuration change, the emulator avoids unnecessary recomputation by determining which prefixes are affected by the policy change and recomputing the best routes only for these prefixes. The first phase of the algorithm only reapplies the import policy for the routes learned on the associated eBGP session. The first phase keeps track of the prefixes that are affected by the policy change, which allows the second phase to reevaluate the BGP decision process only for these prefixes. Then, the third phase evaluates the selection of the egress router for these destination prefixes only. In fact, some of these prefixes might have a set of egress routers that the third phase has evaluated before, allowing the emulator to reuse the result of the earlier computation. Together, these optimizations allow the emulator to quickly answer "what if" questions about incremental changes to the network configuration. We find that recomputing the best routes after a single import policy change takes less than one second on average.

■ 5.8.3 Validation

To verify that the emulator produces correct answers, we perform validation using complete routing protocol implementations on production routers in a large operational network. Network simulators do not capture the full details of the standard routing protocols, so it is not useful to compare the emulator's results with those of a simulator. In addition, we may be unaware of vendor-specific variations that could affect the accuracy of our results. Since we cannot make arbitrary changes to deployed configurations on a

	Number	*Attribute Mismatch*	*Unusual Configuration*	*Total Errors*
AS Paths	43,434	3	9	12 (0.028%)
Routes	1,620,061	36	277	313 (0.019%)

Table 5-3: Summary of errors in applying import policy.

live network, we run the emulator on individual snapshots derived from daily dumps of the router configuration files, BGP routing tables, and BGP neighbor information and compare the emulator's route predictions to what was seen in practice. For each phase of the algorithm, we compare our results to actual BGP tables and present a breakdown of any mismatches we encounter. To isolate the sources of inaccuracy, we evaluate each phase in Figure 5-16 independently, assuming perfect inputs from the previous module; we also perform an end-to-end validation.

The emulator generates correct results for more than 99% of the prefixes. Most mismatches can be attributed to the fact the data sets were collected at slightly different times. The analysis focuses on a snapshot of the network state from early morning (EST) on February 4, 2003. We validate the prediction algorithm for the 91,554 prefixes whose eBGP routes are learned at peering points to other large providers, since we have routing tables from all of these locations; we excluded prefixes that were learned at other routers. (Recall that the prediction algorithm relies on knowing all of the potential egress routers where routes to a prefix are learned.) The initial BGP routing data consists of 1,620,061 eBGP-learned routes with 43,434 distinct AS paths. We applied the tool to these inputs and checked whether the emulator produced the same answers that the operational routers selected. In addition to collecting BGP routing tables from the peering routers (where the eBGP routes are learned), we also collected BGP tables from several route reflectors and access routers to verify the results.

Applying Import Policy

To verify that the first phase correctly emulates the application of import policy, we need only compare the route attributes (*i.e.*, local preference, MED, etc.) in the MODIFIED ROUTES table to the actual BGP routing tables. The MODIFIED ROUTES table contains the routes with attributes modified by applying the import policies specified in the *import* table to the initial *known routes* table. Because the prototype uses routing tables to approximate the actual routes received at each router in the AS, we cannot determine what routes were discarded by the import policy. Thus, the emulator cannot emulate the filtering policies specified by import policies, but it can still determine the effects of import policy configurations that set or manipulate route attributes (*e.g.*, for traffic engineering).

We compared the route attributes between the *known routes* and *modified routes* tables for all 1,620,061 eBGP routes with 43,434 distinct AS paths. Table 5-3 summarizes the results of our validation. The emulator's results matched the route attributes seen in the BGP tables for all but 313 eBGP-learned routes on 12 distinct AS paths. We observed 36 attribute mismatches over 3 AS paths, which can likely be attributed to actual policy changes, since the routing tables and the configuration files were captured at slightly different times of day; we verified this conclusion by inspecting the configuration data for the next day. The remaining mismatches involved 9 unique AS paths because the prototype did not handle

	Number	*Different*	*Missing*	*Total Errors*
AS Paths	43,434	66	187	253 (0.582%)
Prefixes	91,554	120	483	603 (0.659%)

Table 5-4: Mispredictions in the set of best eBGP routes.

a complex configuration scenario permitted on Cisco routers. This accounted for 277 of the 313 route mismatches. Overall, the first phase produced successful results for more than 99.97% of the cases.

Computing the Set of Best eBGP Routes

To verify that the second phase correctly computes the set of best eBGP routes, we check that the best route at each eBGP-speaking router as specified by the EGRESS POINTS table corresponds to the route that appears in the routing table of that router's route reflectors. These routes match the vast majority of the time. However, in a few cases, the two routers had different routes (*i.e.*, with different AS paths), even though one router apparently learned the route directly from the other; these results are summarized in the "Different" column in Table 5-4. The "Missing" column highlights cases where the route reflector did not have *any* route for that prefix. Timing inconsistencies can explain both scenarios, which together account for just over 0.5% of the cases.

To verify that the emulator does not incorrectly exclude routes from the set of best eBGP routes, we check that, for each prefix, the best route at each route reflector appears in the set of best eBGP routes as computed by the emulator[4]. In other words, we consider cases where a route reflector's best route would have directed traffic towards some egress router that was not contained in the EGRESS POINTS table. Only 1.11% of best routes at route reflectors for 2% of prefixes fell into this category. Routing dynamics can explain these inconsistencies—through manual inspection, we found that, in many cases, the best route at the RR was clearly worse than the routes in the set of best eBGP routes (*e.g.*, the route reflector's best route had the same local preference but a higher AS path length).

Computing the Best Route at Each Router

To verify that the emulator correctly predicts the best egress router, we examined the best routes in BGP tables at several routers and compared the (destination prefix, next-hop) pair from the routing table with the results in the PREDICTED ROUTES table entry for that router. We performed these comparisons at two access routers that connected directly to customers in different geographic locations to verify that the emulator makes correct predictions at ingress routers. We also analyzed the emulator's predictions at two route reflectors to verify that the algorithm makes correct route predictions as it traverses the signaling graph. The best route matched our prediction for 99.5-99.7% of the cases, as summarized in Table 5-5. At each router, we excluded prefixes if the best egress router was not one of the peering routers included in the *known routes* table (recall that we excluded routers for which we did not have routing tables). In these cases, we cannot evaluate

[4]The reverse is not necessarily true. An egress point may have a larger IGP path cost to each of the top-level route reflectors for certain sets of eBGP routes.

Router	*# Predictions*	*Case 1*	*Case 2*	*Case 3*	*Total Errors*
RR1	88,865	33	322	21	376 (0.423%)
RR2	88,164	33	185	5	223 (0.253%)
AR1	88,165	38	178	5	221 (0.251%)
AR2	76,547	151	170	37	358 (0.468%)

Table 5-5: Errors in predicting the best egress router. Prefixes predicted incorrectly by the second phase and those where the "right" answer was not a peering router are excluded.

Router	*# Predictions*	*Case 1*	*Case 2*	*Case 3*	*Total Errors*
RR1	89,343	40	459	55	554 (0.620%)
RR2	88,647	39	314	41	394 (0.444%)
AR1	88,649	44	307	40	391 (0.441%)
AR2	76,733	157	283	71	511 (0.666%)

Table 5-6: Summary of errors for end-to-end validation.

whether the algorithm would have made the correct prediction because we didn't have the routes from that egress router in the first place.

We classify the errors among the remaining prefixes in terms of three cases: *Case 1:* The route at the ingress router does not appear in the MODIFIED ROUTES table and, as such, does not appear in the egress set. *Case 2:* The route at the ingress router appears in the MODIFIED ROUTES table but does not appear in the EGRESS POINTS table. *Case 3:* The misprediction has no obvious explanation.

Case 1 errors likely result from timing inconsistencies, where the best route at the ingress router did not exist at the egress router when the routing tables were dumped. Timing inconsistencies can also explain Case 2 errors: for example, an ingress router or a route reflector could have a route that is no longer one of the best eBGP-learned routes, which could happen if a better route arrived at an eBGP-speaking router but had not yet propagated to other routers in the AS. We are unable to explain only 0.05% of the errors.

End-to-End Validation

We performed an end-to-end validation to study the effect of error propagation on the best routes ultimately predicted by the emulator. We compared the emulator's prediction with the same four routing tables used for the validation of the third module, with the exception that the input included the errors from the first two modules. At these four routers, the emulator predicted the correct routes for more than 99% of all prefixes, as summarized in Table 5-6. We hypothesized that the majority of the mispredicted routes could be explained by the dynamics of the input data. For example, if the best route at an eBGP-speaking router were temporarily withdrawn at the time that the route reflector table was captured, inconsistencies between routing tables could arise[5].

These results suggest that the algorithm we have proposed is accurate: prediction errors are infrequent and result mainly the dynamics of the inputs. Since most prefixes whose

[5]To evaluate our hypothesis, we analyzed a feed of iBGP update messages collected on the same day. More than 45% of the prefixes with incorrect predictions had a BGP routing change during the data collection period at the same router where the apparent mismatch occurred, and 83% of the prefixes experienced an update at some router in the AS during this period.

routes change frequently do not receive much traffic [120], these inconsistencies would certainly not prevent the emulator from being used for traffic engineering tasks.

5.9 Proposed Improvements to BGP

Thus far, this chapter has focused on predicting BGP route selection inside a single AS. Notably, two artifacts, the MED attribute and route reflection, complicate this process. Not only do these attributes make route prediction difficult, *they also create problems with the operation of BGP itself*. The use of MED, both with and without route reflection has been shown to cause oscillation [4]; route reflection can also prevent convergence and cause forwarding loops [61]. The MED attribute is intended to allow a neighboring AS to dictate preferred exit points on routes advertised at multiple exit points, but it prevents a router from forming a consistent ordering of preferences over routes. Route reflectors were introduced to allow an iBGP topology to scale, but they do so in a way that prevents routers from discovering the complete set of eBGP-learned routes.

In this section, we explore possible solutions to the problems introduced by MED and route reflection. A major lesson one should draw from this section is that a system that had visibility into an AS's topology, configuration, and available BGP routes could actively *control* the BGP route selection process, rather than simply trying to predict its outcome. The Routing Control Platform (RCP) [13, 31], whose design we will discuss in Chapter 7, can thus not only help ensure correctness (as discussed briefly in Section 4.6), but also make Internet routing easier to control (and, hence, predict).

5.9.1 MED-ication for Late-Exit Semantics

The MED attribute causes problems because it is not comparable across routes from different neighboring ASes, which prevents a router from producing a consistent total ordering over all possible routes. Also, in networks without route reflection, inconsistent preferences between pairs of routes is based on the router ID attribute, an arbitrary tiebreak that carries no meaningful semantics (as in Figure 5-7, for example).

Before we consider solutions to the problems introduced by MED, it is worth noting that MED, as it operates today and when used with route reflection *may not have the intended effect on route selection*. Consider the example shown in Figure 5-6. A neighboring AS sending routes a and b with MED values 10 and 20, respectively, expects that the AS shown would always prefer route a over route b, as long as both existed, causing router X to perform late-exit routing (*i.e.*, send its traffic via route b via router Y). Unfortunately, the AS shown will *not* do so: RR prefers route c, so router X will never learn route b, and it will continue to forward packets via route a.

We observe that if MED values are *remapped* into an explicit ranking across neighboring ASes, rather than arbitrary values, then the MED attribute *can* be compared across all routes at step 4 of the route selection process (as it is today). Comparing an exit-rank across all routes can sometimes result in different outcomes than BGP today, but in many cases the differences do not affect the important semantics of BGP. For example, consider Figure 5-7, but where the MED attribute is compared across all routes. Suppose that the route selection process retains the MED comparison step, but that AS 2's MED values of 10 and 20 are remapped to 1 and 2, and that the highest MED value of any eBGP-learned

route, 2, is added to the MED value on every route learned via iBGP (this transformation guarantees that comparing MEDs across all routes would not cause iBGP-learned routes to be preferred over eBGP-learned routes). In this case, routers X and Y would ultimately select routes c and d, respectively, as opposed to a and d in BGP today. Although X selects c instead of a, its preference between these two routes was based on the arbitrary router ID tiebreak; therefore, having router X select c instead does not destroy any meaningful semantics.

The type of remapping we have described preserves MED's semantics, but implementing an exit-rank requires visibility into the set of available routes that is not available today. Unfortunately, MED values are typically based on dynamic values (*e.g.*, IGP path costs across the network), so an AS that sends routes with MED values cannot simply configure a static ranking. Given today's architectures, neither the sending nor receiving AS could perform a remapping of MED values into an exit-rank, since no single router learns the complete set of routes advertised from a neighboring AS. Performing such a remapping would require either the sending or receiving AS to have complete visibility over all routes being sent or received for a destination. On the other hand, the Routing Control Platform (RCP) [31] or similar recently proposed architectures [9] can perform such a remapping, since RCP has full visibility of routes sent from a neighboring AS (as well as full control over the routes that it sends to a neighboring AS). This modification allow the algorithm from Figure 5-4 to correctly compute the outcome of BGP route selection, and it would also eliminate intra-AS safety problems.

■ 5.9.2 Scalability without Route Reflection

Route reflectors allow iBGP topologies to scale to large number of routers because they obviate the need to have a "full mesh" topology with $O(|R|^2)$ sessions. Unfortunately, they restrict route visibility because they only send a single best route from all of the routes they have learned. In this chapter, we have explained how this restriction complicates predicting the outcome of BGP route selection; previous work has also noted that it can cause persistent oscillation and forwarding loops [4, 61].

To remedy the problems with persistent oscillation, Basu *et al.* proposed that route reflectors forward *all routes* that are equally good up to and including the MED comparison. It turns out that this modification correctly emulates a full mesh iBGP topology; thus, it is possible to model the outcome of their modified protocol with the algorithm from Figure 5-9. Unfortunately, this proposal requires modifications to the routers, since each router readvertises multiple routes instead of a single best route. Additionally, because each router readvertises multiple routes to its neighboring routers, every router must select routes using a *consistent* selection criterion. Otherwise, given multiple routes, some router along the path to an egress router might select a different route, violating route validity (Definition 3.7). This restriction precludes certain policies and configurations (*e.g.*, a router may not manipulate attributes on a route learned via iBGP).

Architectures such as RCP propose separating route selection from the routers and placing this functionality in a system that computes routes on behalf of all of the routers within an AS [31]. Rather than returning only a single best route to all of its clients (as a route reflector does), RCP advertises to each router *the route that it would have selected in a full mesh iBGP configuration*. This architecture allows the network to scale in the same way that route

reflectors do, but it provides some important additional advantages. First, because RCP explicitly assigns routes to all routers in the network, it can *guarantee* that the route assignments satisfy route validity. Second, RCP allows for a more scalable network design. Furthermore, RCP does not have to make the same routing decisions as its clients (as route reflectors do today). As a result, unlike route reflectors, RCP nodes can be replicated at arbitrary places in the IGP topology.

■ 5.10 Summary

To perform everyday network engineering tasks effectively, efficiently, and with minimal unnecessary changes to the live network, operators need a way to predict the behavior of a routing protocol before deploying that configuration. This chapter has presented route prediction algorithms that predict the outcome of BGP route selection based on only a static snapshot of the network state.

In addition to helping network operators accomplish traffic engineering tasks, these algorithms provide useful insight into the subtleties of network-wide BGP route selection and suggest several directions for improvements to the Internet routing system. For instance, network-wide BGP route prediction could be combined with traffic measurements to help network operators select BGP configuration changes that achieve various traffic engineering goals. In addition, the emulator could be combined with higher-level mechanisms that spot misconfiguration or check that other constraints are satisfied [30].

Although the diagram in Figure 5-3 shows only three stages, we envision that network operators could incorporate other phases. For example, another phase could combine the predicted forwarding paths with traffic data to predict the load on each link in the network. Using the model for traffic engineering assumes that traffic volumes are relatively stable, and that they remain stable in response to configuration changes. In previous work, we found that prefixes responsible for large amounts of traffic have relatively stable traffic volumes over long timescales [32]. Operators could use the routing model to test configuration changes on reasonably slow timescales that affect prefixes with stable traffic volumes. A network operator could also combine measurements or estimates of the traffic arriving at each ingress router for each destination prefix [40] with the link-level paths to predict the load on each link in the network. Another phase might evaluate the optimality of the these link-level paths in terms of propagation delay or link utilization and could search for good configuration changes before applying them on a live network.

Finally, we note that modeling BGP routing is more difficult than it should be. In the future, we hope that routing protocol designers will consider predictability as a design goal; as we describe in Section 5.9, some of these simplifications that aid protocol modeling also fix problems with protocol *operation*. Routing protocols that are easy to model and reason about will make everyday network engineering tasks more tractable.

The clock doesn't matter in baseball...
Theoretically, one game could go on forever. Some seem to.
- Herb Caen

CHAPTER 6
Local Conditions for Safe Internet Routing

In Internet routing, independently operated autonomous systems (ASes) must cooperate to exchange global information; nevertheless, this cooperation occurs in a landscape where these independent networks compete to provide Internet service. BGP facilitates this "competitive cooperation" by enabling network operators to express routing policies that are consistent with desired economic, business, and performance goals.

Recall from Section 2.3.1 that *ranking* and *filtering* are the two main mechanisms that operators use to implement their policies. Ranking determines which of many possible routes to a destination should be used, thus providing an AS the flexibility to specify preferences over multiple candidate paths to a destination (*e.g.*, specifying a primary and a backup path). Filtering allows an AS to selectively advertise routes to some ASes and hide routes from others, thereby controlling which neighboring ASes send traffic over its infrastructure.

There are two important characteristics of policy routing: *autonomy* and *expressiveness*. Autonomy is the ability of each AS to set its rankings and filters independent of the others. Expressiveness refers to the flexibility that the routing protocol provides an operator for specifying rankings and filters. Ranking expressiveness determines what classes of rankings over routes are permitted by the protocol, while filtering expressiveness determines the range of route filters that are allowed.

The combination of expressiveness and autonomy has, in large part, been the reason for the success of BGP over the past decade. We contend that both autonomy and filtering expressiveness will be *requirements* for policy routing for the foreseeable future. Previous studies of routing stability assume that ASes are willing to compromise some degree of autonomy, filtering expressiveness, or both (see Section 6.1). However, autonomy preserves each AS's ability to set its policies without coordinating with any other AS. Filtering expressiveness gives an AS flexibility in how it establishes contracts with another AS, a task that should be unconstrained.

Ideally, an interdomain routing system should preserve autonomy, filtering expressiveness, and ranking expressiveness. However, the ability to specify highly expressive rankings comes at considerable cost to system robustness: as has been observed by Varadhan

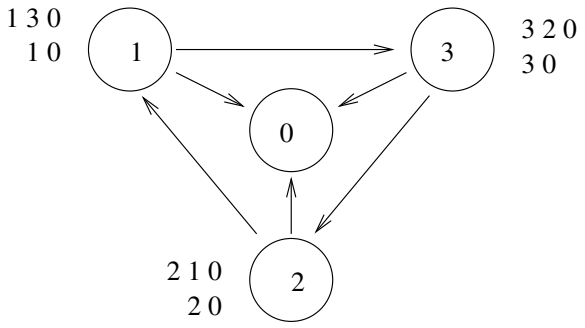


Figure 6-1: Instability can arise when each AS independently specifies rankings [56, 135]. Each circle represents an AS. AS 0 is the destination. The listing of paths beside each node denotes a ranking over paths.

et al. and Griffin *et al.*, among others, if there are no constraints on the rankings that an AS can specify, BGP may violate safety (*i.e.*, oscillate forever) [56, 135].

Example 6.1 Consider Figure 6-1 [56, 135]. ASes 1, 2, and 3 each prefer the indirect path through their neighboring AS in the clockwise direction over the direct path to the destination, 0. All other paths are filtered. This configuration has no stable path assignment (*i.e.*, a path assignment from which no node would deviate). For example, consider the path assignment $(10, 210, 30)$; in this case, AS 1 has a better path available, 130, so it switches paths. This switch breaks the path 210, causing AS 2 to switch to its second choice, path 20. The resulting path assignment, $(130, 20, 30)$, is a permutation of the original path assignment: this time, AS 3 has the path 320 available, so it switches. This oscillation continues forever. ■

As the previous example suggests, full autonomy and expressiveness can have undesirable consequences. Routing protocol update messages should reflect actual reachability changes in the network topology or policy. Unfortunately, in BGP, conflicting policies can cause oscillations that produce endless streams of routing updates that are unrelated to changes in topology or policy. This instability creates numerous performance problems, may cause network partitions, and complicates diagnosis and debugging of problems in the routing system. Worse yet, a network operator has no way to guarantee that any given configuration of rankings and filters will not adversely interact with the policies of other ASes. In light of these issues, developing rigorous conditions on policy expressiveness that guarantee routing stability, while preserving autonomy, is crucial.

This chapter explores the following question: provided that each AS retains complete autonomy and complete filtering expressiveness, how expressive can rankings be while guaranteeing stable routing? This question is important because ranking autonomy and filtering expressiveness reflect the realities of how ASes specify policies today, and little is known (beyond the results surveyed in Section 6.1) about the tradeoffs between autonomy and expressiveness as far as routing stability is concerned, particularly under filtering. In particular, our work is the first to develop necessary conditions for stability under realistic assumptions about autonomy and expressiveness and the first to derive necessary conditions for stability in policy routing.

This chapter makes three main contributions. First, in Section 6.3.1, we show that rankings based solely on the immediate next-hop AS en route to the destination may never

reach a stable path assignment from an arbitrary initial state; *i.e.*, next-hop rankings, which are common in practice, are *not safe*. Moreover, under unrestricted filtering, a routing system with next-hop rankings may have no stable path assignment. In addition to their operational implications, these results are also somewhat surprising, because next-hop rankings with no route filtering always have one stable path assignment. We also observe that although rankings based on a globally consistent weighting of paths are safe under filtering, even minor generalizations of the weighting function compromise safety (Section 6.3.2).

Second, we define a *dispute ring*, a special case of the "dispute wheel" (a group of nodes whose rankings have a particular form) of Griffin *et al.* [56], and show that any routing protocol that has a dispute ring is not safe under filtering (Section 6.4). Using the dispute wheel concept, Griffin *et al.* showed a sufficient condition for safety, proving that if a routing system is unsafe then it must have a dispute wheel. In contrast, to our knowledge, our result is the first known necessary condition for safety under filtering.

Third, we show that, providing for complete autonomy and filtering expressiveness, the set of allowable rankings that guarantee safety is effectively ranking based on variants of weighted shortest paths. In Section 6.5, we prove that any routing system that permits paths of length $n+2$ to be ranked over paths of length n can have a dispute ring, and is thus unsafe under filtering. We also prove that any routing system that permits paths of length $n+1$ to be ranked over paths of length n can have a dispute wheel. In summary, our results indicate that stable policy routing with provider autonomy and expressive filtering requires tight constraints on rankings.

Recent work has observed that routing protocols whose rankings are derived from a "strictly monotonic" algebra are guaranteed to converge [57]; informally, a strictly monotonic algebra is one where a path has a higher cost (*i.e.*, is less preferred) than any of its subpaths. In cases of unrestricted filtering, these strictly monotonic algebras represent a generalization of shortest paths routing, which is consistent with our results. In Section 6.6, we explain how both our results and this algebraic framework lend insight into the design of future policy-based routing protocols.

Our findings may be interpreted in several ways. The optimist will note that checking a set of rankings to ensure safety is trivial, because all it requires is that BGP routers modify the decision process to consult a route's "local preference" attribute only after considering its AS path length. The pessimist, however, may conclude that guaranteeing safe routing while preserving autonomy may yield constraints on expressiveness that are too constraining. In either case, the results proved in this chapter about the fundamental tradeoff between the expressiveness and autonomy may help guide the design of stable interdomain routing protocols in the future; Section 6.6 explores some possibilities.

■ 6.1 Background

Because Internet routing is policy-based, and each AS has the flexibility to define its own policies independently of other ASes, the policies of one AS may interact with those of another to cause the protocol to oscillate. Table 6-1 surveys previous and ongoing work in this area.

Year	Author	Major Results
1996	Varadhan *et al.* [135, 136]	Observed that policy-based routing protocols may never converge (*i.e.*, they may be "unsafe").
1999	Griffin *et al.* [56, 58]	Showed NP-hardness of determining safety, and derived sufficient global conditions for safety.
2001	Gao and Rexford [47]	Derived restrictions on routing configuration and topology that guarantee safety, and observed that these conditions are similar to common practice.
2003	Sobrinho [125]	Developed an algebraic framework for general path vector, policy-based protocols and derived properties that the algebra must have to guarantee convergence.
2003	Griffin *et al.* [54]	Laid out design tradeoffs in policy-based routing protocols: expressiveness, modularity, etc.
2004	Jaggard and Ramachandran [75]	Designed tests for dispute wheels.
2005	Griffin and Sobrinho [57]	Developed a framework called "metarouting", useful for constructing routing protocols that are guaranteed to converge.

Table 6-1: Results from previous work on global routing stability. Chapter 6 builds on many of these results.

A seminal paper by Varadhan *et al.* observed that policy-based interdomain routing protocols could oscillate and defined the concept of *safety* [135, 136]. Varadhan *et al.* also conjectured that routing systems that allow rankings other than those based on next-hop rankings or shortest path routing may be unsafe [135, 136]. In fact, in Section 6.3, we show that even routing systems that only allow next-hop rankings are not safe.

Griffin *et al.* asked how expressive an autonomous, robust routing system can be [54]; we address this question in this chapter. Varadhan *et al.* showed that a routing system with an acyclic topology will have at least one stable path assignment if participants can only express next-hop preferences [135, 136]. We show that when BGP's protocol dynamics are taken into account, restricting each AS to only next-hop rankings does *not* guarantee that the routing system will be safe (even though the routing system always has at least one stable path assignment).

Gao and Rexford derived sufficient constraints on rankings, filtering, and network topology to guarantee routing stability; they also observe that these constraints reflect today's common practice [46, 47]. They showed that if every AS considers each of its neighbors as either a customer, a provider, or a peer, and obeys certain local constraints on rankings and filtering, and if the routing system satisfies certain topological constraints, then BGP is stable.[1] However, their model does not incorporate ranking autonomy, because their proposed topological constraints are global.

[1]Griffin *et al.* noted that analogous sufficient conditions apply to iBGP with route reflection [61], although we show in Chapter 3 that these conditions are unnecessarily strong.

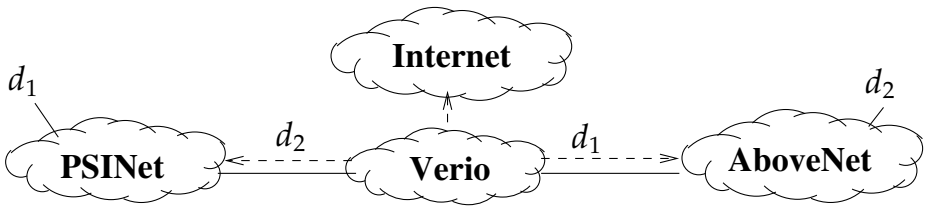


Figure 6-2: Constraints on filtering and topology are not enforceable.

Griffin's sufficient conditions require global knowledge of rankings; those of Gao and Rexford require global knowledge of the AS-level topology. Our goal is to derive constraints that must hold on the configuration of a single AS *without any global knowledge*. Furthermore, Griffin's work does not consider the effects of filtering; the conditions of Gao and Rexford restrict quite severely. The example below illustrates why these restrictions may sometimes be too strict.

Example 6.2 Figure 6-2 shows a situation that occurred in 2001 [10]. When PSINet terminated its peering with AboveNet, AboveNet lost connectivity to PSINet's customers, d_1. To restore connectivity, AboveNet bought "transit" service from Verio (already a peer of PSINet), but only for routes to PSINet and its customers.

Verio does not filter d_1 (or any of PSINet's prefixes) from AboveNet, which is only possible if Verio treats AboveNet as a customer. The constraints imposed by Gao and Rexford state that an AS *must* prefer customer routes over peering routes.[2] This constraint requires Verio to rank AboveNet's route to d_2 over any other available routes to d_2 in order to guarantee stability, which restricts Verio's flexibility in how it can select routes. Establishing a new business relationship (and, hence, altering its filtering policies) *requires* Verio to change its rankings as well. ■

The framework of Gao and Rexford is also too strict because it assumes that a pair of ASes has only a single type of business relationship. For multinational ISPs, this assumption is constantly violated: two ASes may have a peering relationship over sessions in one geographic region, but one may purchase transit from the other in another geographic region.

Example 6.3 Consider Figure 6-3; there are hundreds of similar real-world examples [10]. AS 1 and AS 2, two ISPs, peer in North America, but AS 1 buys service from AS 2 in Europe (in practice, this arrangement may occur if AS 1 does not have a European backbone). AS 1 will typically learn *all* destinations (*i.e.*, European and North American) over its customer link, but just the North American destinations over its peering link. Suppose that AS 2 peers with AS 3 in North America, and AS 1 peers with AS 3 in Europe. The router in AS 2 that has a peering relationship with AS 3 will advertise the European routes from its

[2]Gao and Rexford present a weaker constraint that allows an AS to rank routes learned from customers and peers over those from providers, but does *not* require customer routes to be strictly preferred over routes from peers. This relaxed condition requires that there are no instances where an AS's customer is also a peer of another one of the AS's peers. Of course, the example shown in Figure 6-2 could also violate this constraint on the topology: PSINet is Verio's customer for d_1, but it would be reasonable for PSINet to peer with another of Verio's peers, since all are "tier-1" ISPs.

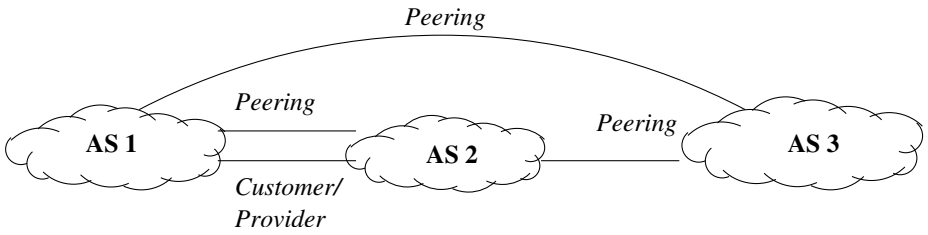


Figure 6-3: Pairs of ASes may have different business relationships in different geographic regions.

customer link; AS 3 also learns those via its peering session with AS 1. This arrangement is precisely the "peer-provider" cycle that is prohibited some of the convergence conditions of Gao and Rexford (*e.g.*, Guideline A, [47]). This scenario mandates that AS 2 prefer the route to the destination via AS 1, rather than another peer through which it may have a route to the same destination. ■

Various previous work has studied *global* conditions to guarantee the safety of routing systems; global conditions presume that the routing system does not preserve local choice of rankings (*i.e.*, ranking autonomy). Griffin *et al.* showed that, if the rankings of the ASes in a routing system do not form a *dispute wheel* (a concept that describes global relationship between the rankings of a set of ASes), then the routing system is safe [56]. Griffin *et al.* also examined *robustness*, the property that safety is guaranteed even if arbitrary nodes or edges are removed from the graph. We view robustness as a special case of filtering: removing an edge can be achieved if the ASes incident to that edge filter all routes through that edge; removing a node entails having all ASes filter all routes through that node.

Griffin *et al.* also showed how to modify a BGP-like path vector protocol to detect the existence of a dispute wheel but left unspecified how the ASes should resolve the dispute wheel [59]. Machiraju and Katz defined a new global invariant for determining safety when at most one AS deviates from the conditions of Gao and Rexford [83]. Govindan *et al.* proposed a routing architecture where ASes coordinate their policies [52, 53] using a standardized policy specification language [2]. Jaggard and Ramachandran presented global conditions that guarantee safety of routing systems that allow ASes to express only next-hop preferences over routes, and designed centralized and distributed algorithms to check these global conditions [75].

More recent work has attempted to design policy-based protocols that are guaranteed to converge without imposing any global conditions. Sobrinho defined new concepts that describe global relationships between preferences and incorporate several previous results (including those of both Griffin *et al.* [56] and Gao and Rexford [47]) into a single algebraic framework [125]. He examined requirements for convergence and asserted that any vectoring protocol that preserves a property called "strict monotonicity" is guaranteed to converge. Recent work on "metarouting" exploits this algebraic framework to allow protocol designers to specify new routing protocols that are guaranteed to converge by requiring algebras that preserve strict monotonicity [57]. In this chapter, we will see that the set of routing protocols that are guaranteed to converge is apparently not much more permissive than shortest path routing, if ASes have complete liberty in setting filters. Metarouting may allow the protocol designer to better explore the tradeoffs between filtering expres-

siveness, ranking expressiveness, and safety (*i.e.*, compromising filtering expressiveness to allow for more expressive rankings).

In contrast to these studies of global conditions for safety, we study the conditions under which a policy-based interdomain routing protocol can be safe if it preserves the autonomy of each AS. Our results suggest that, allowing for complete autonomy and filtering expressiveness, guaranteeing safety requires restricting ranking independence essentially to preferences based on consistent weightings of the edges in the graph.

Gouda and Schneider study classes of routing protocol metrics for which each node in a routing tree has its most preferred path, but do not address routing stability [51].

6.2 Routing Model and Definitions

We now define our routing model. After introducing some basic terminology, we formally define two notions of good behavior for routing protocols: *stability* and *safety*. Finally, we extend each of these two definitions to handle filtering expressiveness.

6.2.1 Preliminaries

We consider a model consisting of N ASes (nodes)[3], labeled $1, \ldots, N$. Each of these nodes wishes to establish a path (defined below) to a single destination, labeled 0.

Definition 6.1 (Path) *A* path *from i to j is a sequence of nodes $P = ii_1i_2 \ldots i_m j$ with no repetition;* i.e., *such that $i_u \neq i_v$ if $u \neq v$, and $i_u \neq i, j$ for all u.*

Note that we have slightly altered the definition of a *path* from Chapter 3 (Definition 3.1), although the concept is conceptually the same: we have eliminated commas and subscripts for notational convenience, and we have defined that the path to be loop-free, as BGP typically discards paths that have loops in the AS-level path. Definition 6.1 also precludes the practice of AS-path prepending (described in Section 2.2.2), as it does not affect the results in this chapter.

We denote the number of hops in a path P as $length(P)$. In addition, given an AS k, we will write $k \in P$ if node k appears in P. For clarity, given a path P from i to j, we will often denote P by iPj; furthermore, if P is a path from i to j, and Q is a path from j to k, then we will denote the concatenation of P and Q by $iPjQk$.

We denote the set of *all* paths from i to 0 (*i.e.*, all paths on the complete graph) using the nodes $1, \ldots, N$ by $\mathcal{P}_i^N$. Given the set of nodes $\{1, \ldots, N\}$, each AS i will choose a *ranking* $\prec_i$ over the set of all paths $\mathcal{P}_i^N$, defined as follows.

Definition 6.2 (Ranking) *A* ranking $\prec_i$ *for node i is a total ordering over the set of all paths $\mathcal{P}_i^N$; thus, given any two paths $P, Q \in \mathcal{P}_i^N$, either $P \prec_i Q$ (i prefers Q to P) or $P \succ_i Q$ (i prefers P to Q).*

An AS may always choose the *empty path*, ε, which is equivalent to total disconnection from the destination node 0. Thus, we have $\varepsilon \in \mathcal{P}_i^N$ for all i and N. Furthermore, we assume that every AS strictly prefers connectivity to disconnectivity, so that $P \succ_i \varepsilon$ for all $P \in \mathcal{P}_i^N$.

[3] In this chapter, we use the terms "AS" and "node" interchangeably.

All paths may not be available to node i, due to both topological constraints and filtering by other nodes. We will use $\mathcal{F}_i \subseteq \mathcal{P}_i^N$ to denote the set of paths actually available for use by node i. The empty path is always available; *i.e.*, $\varepsilon \in \mathcal{F}_i$.

A *routing system* is specified by the rankings of the individual nodes, together with the paths available to the individual nodes. Observe that we have decoupled the "routing policy" of each AS i into two components: the rankings $\prec_i$ of AS i over route advertisements received; and a determination of which paths are filtered from other ASes. The filtering decisions of all nodes, together with physical constraints on the network, yield the sets $\mathcal{F}_1, \ldots, \mathcal{F}_N$. We thus have the following formal definition of a routing system.

Definition 6.3 (Routing system) *A* routing system *is a tuple* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$*, where node* i *has ranking* $\prec_i$ *over the set* $\mathcal{P}_i^N$*, and* $\mathcal{F}_i$ *is the set of paths available to node* i*.*

A routing system specifies the input to any interdomain routing protocol we might consider. Given this input, the protocol should converge to a "routing tree": that is, an assignment of a path to each AS, such that the routes taken together form a spanning tree rooted at 0. To formalize this notion, we must define path assignments and consistent paths.

Definition 6.4 (Path assignment) *A* path assignment *for the routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is a vector of paths* $\boldsymbol{P} = (P_1, \ldots, P_N)$ *such that, for all* i*,* $P_i \in \mathcal{F}_i$*.*

Thus, a path assignment is an assignment of a feasible path to each node i, where feasibility is determined by the set of paths $\mathcal{F}_i$. Even though each node has a path assigned, these paths may not be *consistent*: node i may be assigned a path $P_i = ij\hat{P}_j0$, where j is the first node traversed on P_i, and where $\hat{P}_j$ is a path from j to 0. However, the path $\hat{P}_j$ may not be the same as the path P_j assigned to j in the path assignment $\boldsymbol{P}$; in fact, $\hat{P}_j$ may not even be in the set of feasible paths $\mathcal{F}_j$. For example, a node or link along the path $\hat{P}_j$ may experience a failure, causing the routing protocol to withdraw the path; if j has heard such a withdrawal but i has not, then it is possible that $P_i = ij\hat{P}_j0$ until node i learns that $\hat{P}_j$ no longer exists. To formally capture such situations, we define consistent paths and consistent path assignments. The definition of consistent path is a simplification of Definition 3.6 from Chapter 3, since we do not require the notion of an induced path in this chapter.

Definition 6.5 (Consistent path) *Given a path assignment* $\boldsymbol{P}$*, a path* $\hat{P}_i$ *for node* i *is* consistent *with* $\boldsymbol{P}$ *if one of the following holds:*

1. $\hat{P}_i = \varepsilon$*; or*
2. $\hat{P}_i = i0$*; or*
3. $\hat{P}_i = ijP_j0$*, for some* $j \neq i$*.*

Definition 6.6 (Consistent path assignment) *A* consistent path assignment *for the routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is a path assignment vector* $\boldsymbol{P} = (P_1, \ldots, P_N)$ *such that for all* i*,* P_i *is consistent with* $\boldsymbol{P}$*.*

A routing protocol where packets are forwarded solely on destination should ultimately assign paths that are consistent with each other.

6.2.2 Stability and Safety

Informally, a path assignment is *stable* if it is consistent, and no node has a more preferred consistent path available.

Definition 6.7 (Stable path assignment) *Given a routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$*, and a consistent path assignment* $\boldsymbol{P}$*, we say that* $\boldsymbol{P}$ *is* stable *if for all nodes* i*, and all paths* $\hat{P}_i \in \mathcal{F}_i$ *that are consistent with* $\boldsymbol{P}$*,* $\hat{P}_i \prec_i P_i$*.*

Definition 6.8 (Stable routing system) *The routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is* stable *if there exists at least one stable path assignment* $\boldsymbol{P}$*.*

The stability of a routing system does not indicate whether a routing protocol will converge *regardless* of the initial path assignment. The *safety* property, which states that a protocol eventually converges, regardless of the initial path assignment and ordering of the routing messages, captures this notion.

In defining safety, we will consider a simplified abstraction of BGP. We model the process by which nodes receive route advertisements from other nodes and subsequently update their own route decisions. We will consider a protocol dynamic where at each time step only a single AS is *activated*; when activated, an AS immediately processes all pending incoming route advertisements, and then makes a route decision. Formally, this model will translate into a path assignment sequence where exactly one node (the "activated" node) changes its route at any given time step.

A routing system is safe if no oscillation occurs regardless of the order in which nodes are activated.

Definition 6.9 (Fair activation sequence) *The sequence* $i_1, i_2, \ldots$ *is a* fair activation sequence *if each node* $i = 1, \ldots, N$ *appears infinitely often in the sequence.*

This definition of fair activation sequence is similar to that presented by Gao and Rexford [47], except that in our definition we only activate one node at a time. This distinction is minor: we can interpret the Gao and Rexford dynamics as a model where outstanding routing messages may be in flight when a particular node is activated.

We now define our simplified model of the routing protocol dynamics: that is, starting from an initial path assignment $\boldsymbol{P}_0$, and given a fair activation sequence of nodes $i_1, i_2, \ldots$, what is the resulting observed sequence of path assignments $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots$? To formalize the dynamics of our model, we consider an abstraction of the BGP decision process described in Figure 6-4. At each time t, a node i_t is activated, and chooses its most preferred available path consistent with the path assignment $\boldsymbol{P}_{t-1}$. All other nodes' paths remain unchanged. It is clear that this decision process yields a sequence of path assignments $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots$.

After any given activation step t, the overall path assignment $\boldsymbol{P}_t$ may not be consistent. Inconsistencies reflect the fact that a node only updates its path assignment in response to the receipt of a route advertisement. If, at time t_0, a node i is using a path that traverses some other node j that has since changed paths, then node i would obliviously continue to use (and advertise) that *inconsistent* path until it receives a routing update that reflects that the path through j has disappeared or changed. When activated, say, at time $t > t_0$, node i would discover that the path it was using was inconsistent with $\boldsymbol{P}_t$ and would then select

Routing protocol dynamics
At time $t-1$, the current path assignment is $\boldsymbol{P}_{t-1}$; *i.e.*, each node i has currently selected path $P_{i,t-1}$ to the destination 0. At time t:

1. A given node i_t is activated.
2. Node i_t updates its path to be the *most preferred path* (according to $\prec_{i_t}$) consistent with $\boldsymbol{P}_{t-1}$. That is,
   (a) $P_{i_t,t} \in \mathcal{F}_{i_t}$ is consistent with $\boldsymbol{P}_{t-1}$, and
   (b) $P_{i_t,t} \succ_{i_t} \hat{P}_{i_t} \; \forall \; \hat{P}_{i_t} \in \mathcal{F}_{i_t}$ consistent with $\boldsymbol{P}_{t-1}$.
3. All other nodes leave their paths unchanged.

Figure 6-4: The routing protocol dynamics, given an activation sequence $i_1, i_2, \ldots$. The process starts from an initial path assignment P_0.

its highest-ranked path that was consistent with $\boldsymbol{P}_t$. The activation of a node at some time t corresponds to that node receiving all available routing information in the system up to that time.

With the definition of our protocol dynamics in hand, we can define protocol *safety*. Given a routing system and an activation sequence, we say that the system has converged if, after some finite time, the path assignment remains invariant for all future time. A protocol is *safe* if it converges to a stable path assignment, regardless of the initial path assignment and fair activation sequence.

Definition 6.10 (Safety) *A routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is* safe *if for any initial path assignment* $\boldsymbol{P}_0$ *and fair activation sequence* $i_1, i_2, \ldots$, *there exists a finite* T *such that* $\boldsymbol{P}_t = \boldsymbol{P}_T$ *for all* $t \geq T$.

Because the activation sequences are fair in the preceding definition, if a routing system converges to $\boldsymbol{P}_t$, then the resulting path assignment to which the system converges must be both consistent and stable. If not, at least one node would change its path assignment eventually.

■ 6.2.3 Filtering

We are interested in the stability and safety of systems that result when nodes are allowed to filter routes from other nodes. We thus require conditions stronger than stability and safety, known as *stability under filtering* and *safety under filtering*. Informally, a routing system is stable (respectively, safe) under filtering if, under any choices of filters made by the ASes, the resulting routing system is always stable (respectively, safe).

Definition 6.11 (Stable under filtering) *The routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is* stable under filtering *if, for all choices of available paths* $\hat{\mathcal{F}}_i \subseteq \mathcal{F}_i$ *for* $i = 1, \ldots, N$, *the routing system* $(N, \prec_1, \ldots, \prec_N, \hat{\mathcal{F}}_1, \ldots, \hat{\mathcal{F}}_N)$ *is stable.*

Definition 6.12 (Safe under filtering) *The routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is* safe under filtering *if, for all choices of available paths* $\hat{\mathcal{F}}_i \subseteq \mathcal{F}_i$ *for* $i = 1, \ldots, N$*, the routing system* $(N, \prec_1, \ldots, \prec_N, \hat{\mathcal{F}}_1, \ldots, \hat{\mathcal{F}}_N)$ *is safe.*

We interpret these definitions as follows. The set of available paths $\mathcal{F}_i$ gives the set of paths that are physically possible for node i to use, given the current network topology. Once all nodes have chosen their route filters, $\hat{\mathcal{F}}_i$ gives the set of paths that can ever be used by node i in route advertisements. Because we allow arbitrary choice of filters, the resulting routing system should be stable and safe regardless of the choices of $\hat{\mathcal{F}}_1, \ldots, \hat{\mathcal{F}}_N$ that are made.

■ 6.3 Ranking Classes and Safety

In this section, we study two natural ranking classes under which ASes retain autonomy in setting rankings over paths. First, in Section 6.3.1, we study the rankings where each AS is allowed to rank paths solely based on the immediate next-hop AS, called "next-hop rankings". We show that (1) there are routing systems where each node has only a next-hop ranking that are not safe; and (2) even though all routing systems where nodes have next-hop rankings are stable, there exist some routing systems of this form that are not stable under filtering.

In Section 6.3.2, we study the properties of routing systems where each node is allowed to choose a weight for all its outgoing links, and rankings are derived from a "total" weight associated to each path. The total weight of a path is defined as the weight of the first link on that path, plus a discounted sum of the weights of all remaining links on that path. We show that if the discount factor is anything other than 1 (which corresponds to shortest path routing), then there exist weight configurations that yield a routing system that is not safe.

■ 6.3.1 Next-Hop Rankings

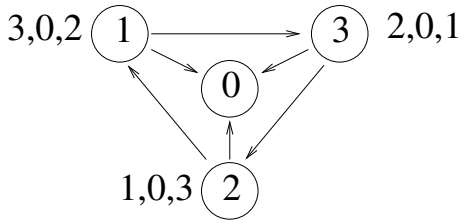
One natural set of rankings for a routing system is one where each AS can express rankings over paths solely based on the next-hop AS in the path. Such a class of rankings makes sense because an AS establishes bilateral contracts with its immediate neighbors and, as such, will most often wish to configure its rankings based on the immediate next-hop AS en route to the destination. For example, an AS will typically prefer sending traffic via routes through its neighboring customer ASes over other ASes, since those customer ASes are paying based on traffic volume. We formally define next-hop rankings as follows:

Definition 6.13 (Next-hop ranking) *Given* N*,* $\prec_i$ *is a* next-hop ranking *if, for all nodes* j, k *with* i, j, k *distinct, we have:*

$$ijP_j0 \prec_i ikP_k0 \Rightarrow ijP'_j0 \prec_i ikP'_k0, \tag{6.1}$$

for all $P_j, P'_j \in \mathcal{P}_j^N$*, and* $P_k, P'_k \in \mathcal{P}_k^N$*. (Here we interpret* $\mathcal{P}_0^N = \{\varepsilon\}$*.) Thus,* $\prec_i$ *ranks paths based only on the first hop of each path.*

Such a restriction on policy would still be sufficiently rich to achieve most traffic engineering goals, since most policies are based on the immediate next-hop AS [32]. Addi-



(a) Routing system

Activate	1	2	3
—	(1 0)	(2 0)	(3 2 0)
2	(1 0)	**(2 1 0)**	(3 2 0)
1	**(1 3 2 0)**	(2 1 0)	(3 2 0)
3	(1 3 2 0)	(2 1 0)	**(3 2 1 0)**
2	(1 3 2 0)	**(2 0)**	(3 2 1 0)
1	**(1 0)**	(2 0)	(3 2 1 0)
3	(1 0)	(2 0)	(3 2 0)

(b) Activation sequence

Figure 6-5: Next-hop rankings are not safe in this routing system. AS 1 prefers all paths through AS 3 over the direct path to the destination 0 (with ties broken deterministically) and prefers the direct path over all paths through AS 2. Similarly, AS 3 prefers all paths via AS 2, and so forth.

tionally, this class of rankings is expressive enough for most current policy goals, because most current routing policies are dictated according to the AS's business relationship with its immediate neighbor. In this section, we show that while systems with next-hop rankings are generally stable, there exist examples that are unsafe, as well as systems that are unstable under filtering.

In the following proposition, we routing systems with next-hop rankings, provided that no filtering is employed. The proof is straightforward, using a construction due to Feigenbaum *et al.* [38].

Proposition 6.1 *Suppose* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is a routing system such that* $\prec_i$ *is a next-hop ranking for each* i*, and* $\mathcal{F}_i = \mathcal{P}_i$ *for all* i*. Then there exists a stable path assignment* $\boldsymbol{P}$ *for this routing system.*

We now show that there may exist $\hat{\mathcal{F}}_1 \ldots \hat{\mathcal{F}}_N$, where $\hat{\mathcal{F}}_i \subseteq \mathcal{F}_i$ for all i, such that even though the system $(N, \prec_1 \ldots \prec_N, \mathcal{F}_1 \ldots \mathcal{F}_N)$ is stable, the filtered system $(N, \prec_1 \ldots \prec_N, \hat{\mathcal{F}}_1 \ldots \hat{\mathcal{F}}_N)$ is unstable. That is, there exist routing systems with next-hop rankings for which a stable path assignment exists, but introducing filtering can yield a system where *no* stable path assignment exists.

Observation 6.1 *A routing system where each node has only a next-hop ranking may not be safe.*

Example 6.4 Consider Figure 6-5. In this example, each AS ranks every one of its neighboring ASes. For example, AS 1 prefers all paths that traverse AS 3 as the immediate next hop over all paths that traverse AS 0 as the immediate next hop, regardless of the number of ASes each path traverses; similarly, AS 1 prefers paths that traverse AS 0 as the immediate next hop over paths that traverse AS 2. Each AS readvertises its best path to the destination to all of its neighbors (*i.e.*, the system has no filtering). Now consider the activation sequence in Figure 6-5(b); if infinitely repeated, this activation sequence would be fair, and the routing system would oscillate forever. Thus, the routing system is not safe.

This system is not *safe*, but it is *stable*: for example, the path assignment $(10, 210, 3210)$ is stable. Nodes 2 and 3 are using paths through their most preferred nodes. Node 1's most preferred node, node 3, is using a path that already goes through node 1, so node 1 is also

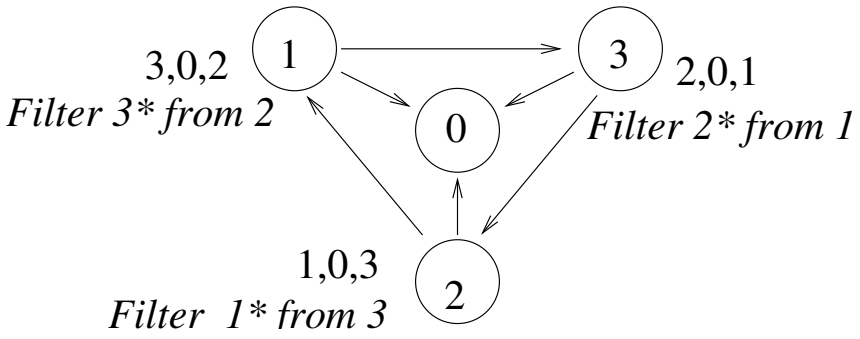


Figure 6-6: This routing system is stable without filtering but unstable under filtering. The figure shows a routing system with next-hop rankings and filtering that is equivalent to the unstable routing system with the rankings over paths shown in Figure 6-1.

using its most preferred consistent path. As every node is using its most preferred consistent path, no node will change paths when activated, so the path assignment is stable. ■

A routing system where each node has a next-hop ranking may not be safe, but Feigenbaum *et al.* showed that there is always guaranteed to be at least one stable path assignment for such routing systems [38]. However, allowing nodes to filter paths from each other can create routing systems for which there is *no* stable path assignment.

Observation 6.2 *There exist routing systems with next-hop rankings for which a stable path assignment exists, but introducing filtering can yield a system where* no *stable path assignment exists.*

Example 6.5 Consider Figure 6-6. As before, each AS ranks every one of its neighboring ASes. Additionally, each AS may also declare arbitrary filtering policies. In this example, each AS (other than the destination) does *not* readvertise any indirect path to the destination. For example, AS 1 does not advertise the path 130 to AS 2, and thus the path 2130 is not available to AS 2. Formally, we define $\mathcal{F}_1 = \{130, 10\}$, $\mathcal{F}_2 = \{210, 20\}$, and $\mathcal{F}_3 = \{320, 30\}$.

The resulting routing system is equivalent to the system in Figure 6-1, once the filtered paths are removed from each node's ranking. Thus, the filtered routing system is unstable by the same reasoning as that from the example shown in Figure 6-1: for any path assignment in this routing system, at least one AS will have a higher ranked consistent path (and, hence, has an incentive to deviate from the path assignment). ■

Using a construction similar to that from the example in Figure 6-2, it is possible to show how this example could arise in practice. The example demonstrates the complex interaction between filtering and rankings—a class of rankings that guarantees stability without filtering can be unstable under certain filtering conditions.

■ 6.3.2 Edge Weight-Based Rankings

There exists at least one routing system that preserves autonomy and yet ensures safety under filtering: if each provider is allowed to choose edge weights for its outgoing links,

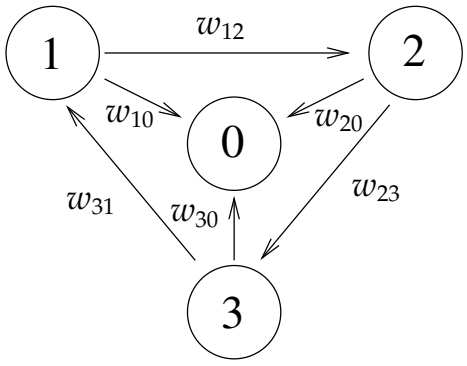


Figure 6-7: Routing system with edge weight-based rankings.

and each provider ranks paths based on the sum of edge weights, the resulting "shortest paths" routing system is guaranteed to be safe [56]. Since this result holds for any $\mathcal{F}_1, \ldots, \mathcal{F}_N$, any routing system built in this way is guaranteed to be safe under filtering. In this section, we will formulate a generalized model of such *edge weight-based rankings*, with both next-hop rankings and shortest path routing as special cases. Such rankings do not allow providers to directly specify their ranking; rather, the rankings of each provider are *derived* from the strategic choices made by all providers, namely, the choices of outgoing link weights that each provider sets. This notion of "derived" rankings is a potentially useful method for ensuring autonomy in interdomain routing protocols.

Definition 6.14 (Edge weight-based rankings)
$(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ *is a routing system with* edge weight-based rankings *if there exists an assignment of edge weights w_{ij} to each ordered pair of ASes i, j, together with a parameter $\alpha \in [0,1]$, such that for each AS i and paths $P_i, \hat{P}_i \in \mathcal{P}_i^N$ with $P_i = ii_1 \ldots i_n 0$ and $\hat{P}_i = ij_1 \ldots j_m 0$, there holds:*

$$P_i \prec_i \hat{P}_i \quad \text{if and only if} \quad w_{ii_1} + \alpha \left(\sum_{k=1}^{n-1} w_{i_k i_{k+1}} + w_{i_n 0} \right) > w_{ij_1} + \alpha \left(\sum_{\ell=1}^{m-1} w_{j_\ell j_{\ell+1}} + w_{j_m 0} \right).$$

The interpretation of this definition is as follows. Each node chooses edge weights for all possible outgoing links; *i.e.*, node i chooses a weight w_{ij} for each node j. Next, node i determines its rankings by ordering all paths $P_i = ii_1 \ldots i_n 0$ in increasing order according to their weight $w_{ii_1} + \alpha(\sum_{k=1}^{n-1} w_{i_k i_{k+1}} + w_{i_n 0})$, where α is a global parameter used to weight the tail of the path. The parameter α allows us to compare two extreme points: $\alpha = 1$, corresponds to shortest path routing based on the matrix of edge weights $\boldsymbol{w}$, while $\alpha = 0$ corresponds to next-hop rankings. A natural question to ask is whether a routing system using edge weight-based rankings can be safe for intermediate values of α. It turns out that the *only* edge weight-based ranking class that can guarantee safety (and safety under filtering), regardless of the weights chosen by each provider, is the scheme defined by $\alpha = 1$; *i.e.*, shortest path routing.

Observation 6.3 *A routing system with edge weight-based rankings may be unstable for any α where $0 < \alpha < 1$.*

Example 6.6 Consider the routing system shown in Figure 6-7. If the system is such that each node prefers the two-hop path to the destination, followed by the one-hop (*i.e.*, direct) path, followed by the three-hop path, then the system will be unstable because its behavior will correspond to that shown in Figure 6-1. The routing system will be unstable if the following conditions are satisfied, for all $i = 1, 2, 3$: $w_{i,i+1} + \alpha w_{i+1,0} < w_{i,0} < w_{i,i+1} + \alpha(w_{i+1,i+2} + w_{i+2,0})$ (for addition modulo 3). If $\alpha = 1$, these inequalities cannot be simultaneously satisfied for any nonnegative choice of the edge weight vector $\boldsymbol{w}$, which is expected, since $\alpha = 1$ corresponds to shortest path routing. On the other hand, if $0 < \alpha < 1$, then there are many vectors $\boldsymbol{w}$ that satisfy the inequalities above. For example, we can choose $w_{10} = w_{20} = w_{30} = 1$, and let $w_{12} = w_{23} = w_{31} = x$, for any x such that $(1 - \alpha)/(1 + \alpha) < x < 1 - \alpha$. For this definition of $\boldsymbol{w}$, all three inequalities above will be satisfied, and thus the rankings of each node will lead to the same oscillation shown in Figure 6-1. ■

The results in this section imply that routing protocols maybe unstable if individual nodes may rank paths in ways that deviate even slightly from rankings based on shortest paths routing. Of course, the reader should not interpret these results as saying that shortest paths routing is the *only* routing protocol that will converge, since there is a larger class of routing protocols based on strictly monotonic algebras (of which shortest paths routing is one instance) that are safe [57]. Rather, the results imply that a routing protocol where nodes may rank paths in ways that violate the monotonicity of shortest paths routing—even if only slightly—are may be unstable.

■ 6.4 Dispute Wheels and Dispute Rings

Our goal is to study the classes of rankings for which the routing system is guaranteed to be safe under filtering. Griffin *et al.* have shown that checking whether a particular routing system is safe is NP-hard [56]. To simplify our study of safety, we introduce a useful concept developed by Griffin *et al.* [56], known as a *dispute wheel*. Informally, a dispute wheel gives a listing of nodes, and two path choices per node, such that one path is always preferred to the other. If a routing system oscillates, then it is possible to construct a dispute wheel whereby each node in the wheel selects its more preferred path (via the node in the clockwise direction) over its less preferred path. Griffin *et al.* showed that if a routing system with no filtering does not have a dispute wheel, then it is safe.

The dispute wheel is a useful concept because it allows us to analyze dynamic properties such as safety by simply looking at the rankings of each node in the routing system. In this section, we formally define a dispute wheel and show the relationship of Griffin's routing model, which simulates messages being passed between nodes, to the model we use in this chapter, which uses fair activation sequences. This relationship allows us to study safety in terms of the routing model in this chapter. We then introduce a special type of dispute wheel called a *dispute ring* and show that, if any routing system has a dispute ring, then it is not safe under filtering. Finally, we relate dispute wheels to dispute rings and show that, although the presence of a dispute ring guarantees that a routing system is not safe under filtering, it does not necessarily imply that a routing system is not safe without filtering. Figure 6-8 summarizes the results of this section and how they relate to results from previous work [56].

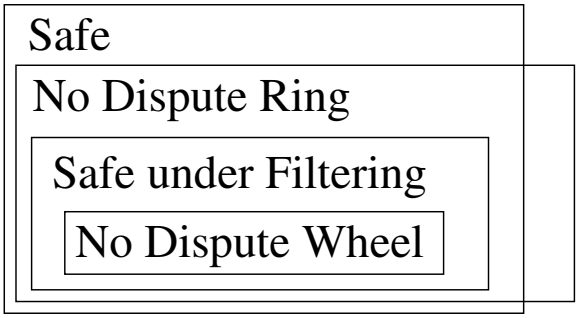


Figure 6-8: Relationships between safety and dispute rings and wheels. Previous work showed that a routing system with no dispute wheel is safe [56]. Section 6.4 presents all other relationships shown in this figure.

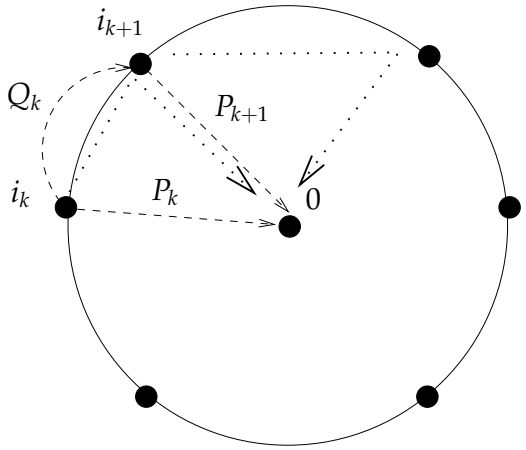


Figure 6-9: Illustration of a dispute wheel. Dotted lines show preferred (indirect) paths to the destination. The nodes $i_1, \ldots, i_m$ are pivots.

■ 6.4.1 Dispute Wheels and Safety

Definition 6.15 (Dispute wheel) *Given a routing system* $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$*, a* dispute wheel *is a collection of distinct nodes* $i_1, \ldots, i_m$*, called* pivots*, together with two sets of paths* $P_1, \ldots, P_m$ *and* $Q_1, \ldots, Q_m$*, such that the following conditions all hold (where we define* $i_{m+1} = i_1$ *for notational convenience):*

1. $P_k \in \mathcal{F}_{i_k}$ *for all* $k = 1, \ldots, m$*;*
2. Q_k *is a path from* i_k *to* i_{k+1} *for all* $k = 1, \ldots, m$*;*
3. *The path* $\hat{P}_k = i_k Q_k i_{k+1} P_{k+1} 0$ *is feasible,* i.e., $\hat{P}_k \in \mathcal{F}_{i_k}$*,*
4. $\hat{P}_k \succ_{i_k} i P_k 0$*.*

Thus, each node i_k prefers the path $i_k Q_k i_{k+1} P_{k+1} 0$ to the path $i_k P_k 0$, as shown in Figure 6-9.

We now show that safety in the Simple Path Vector Protocol (SPVP) defined by Griffin *et al.* [56] implies safety in our model, which allows us to use dispute wheels to analyze safety.

Proposition 6.2 *Given a routing system, a fair activation sequence, and an initial path assignment $\boldsymbol{P}_0$, let $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots$ be the resulting sequence of path assignments according to the dynamics described in Figure 6-4. Then there exists a sequence of messages in the Simple Path Vector Protocol (SPVP) such that the same sequence of path assignments is observed.*

Thus, in particular, if a routing system is safe under SPVP, then it is safe according to Definition 6.10.

Proof Sketch. The key difference between SPVP and the dynamics we have defined is that SPVP is *asynchronous* (*i.e.*, at any time step, messages may be in flight), so different nodes may have different assumptions about the global path assignment at any time. SPVP is *nondeterministic* with respect to the timing of messages; the delay between a routing update at node j and the receipt of the new route advertisement from node j at node i can be arbitrary. We use this fact to construct, inductively, a sequence of messages such that at time t, the current set of paths available to node i_t in SPVP corresponds exactly to $\boldsymbol{P}_{t-1}$. Furthermore, we time the delivery of routing updates to node i_t in SPVP so that any updates that occurred since the last time i_t was activated arrive exactly at the start of time step t. In SPVP, this will initiate a routing update at node i_t, which corresponds exactly to the activation of i_t in our model (see Figure 6-4).

Thus, the sequence of path assignments seen in this realization of SPVP matches the sequence of path assignments seen in our dynamics. We conclude that if SPVP is guaranteed to be safe for the given routing system (*i.e.*, if eventually no further routing updates occur, regardless of the initial path assignment), then the routing system is safe according to Definition 6.10 as well. ■

Corollary 6.1 *If a routing system $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ has no dispute wheel, then it is safe under filtering (and hence safe).*

Proof. Choose subsets $\hat{\mathcal{F}}_i \subseteq \mathcal{F}_i$. Then, any dispute wheel for the routing system $\hat{S} = (N, \prec_1, \ldots, \prec_N, \hat{\mathcal{F}}_1, \ldots, \hat{\mathcal{F}}_N)$ is also a dispute wheel for the original routing system $S = (N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$. Thus, the result follows from Proposition 6.2 and the results of [56]. ■

If no dispute wheel exists, the routing system is safe under filtering, but, unfortunately, this condition is not a necessary condition for safety, and thus not much can be said about a system that does have a dispute wheel. Furthermore, there exist routing systems that have a dispute wheel but which are safe under filtering.

Observation 6.4 *The existence of a dispute wheel does* not *imply that the routing system is unsafe, nor that the routing system is not safe under filtering.*

Example 6.7 See Figure 6-10. The first two most preferred paths in each node's ranking form a dispute wheel, but the system is safe: the system converges to $\boldsymbol{P} = (10, 20, 30)$. Furthermore, *no combination of filters can create an oscillation*. The two-hop paths are not part of the stable path assignment, so filtering those paths has no effect on the protocol dynamics. Filtering a three-hop path would simply result in a node selecting the direct

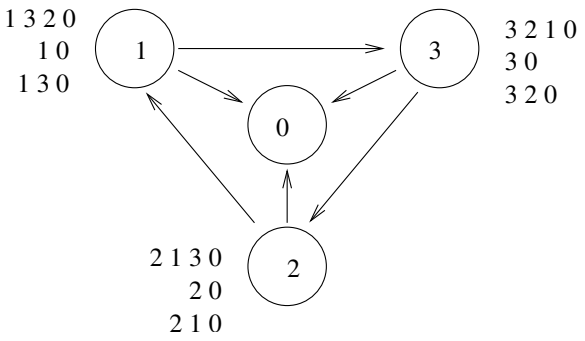


Figure 6-10: A routing system that is safe for any choice of filters.

path to the destination, and the node would never deviate from that path. If one direct path is filtered, then the other two nodes will take direct paths to the destination and the node whose direct path is filtered will take its most preferred three-hop path. If two direct paths are filtered, then $\boldsymbol{P}$ is simply a chain to the destination: the node that has the direct path takes it, and the other two nodes will take two and three-hop paths. ■

■ 6.4.2 Dispute Rings and Safety

In this section, we extend the dispute wheel notion to understand the relationship between ranking expressiveness and safety under filtering. We define a relationship between rankings called a *dispute ring*, a special case of a dispute wheel where each node appears at most once. The dispute ring is a useful concept because it allows us to prove a *necessary* condition for safety under filtering.

Definition 6.16 (Dispute ring) *A* dispute ring *is a dispute wheel—a collection of nodes $i_1, \ldots, i_m$ and paths $P_1, \ldots, P_m$, $Q_1, \ldots, Q_m$ satisfying Definition 6.15—such that $m \geq 3$, and no node in the routing system appears more than once in the wheel.*

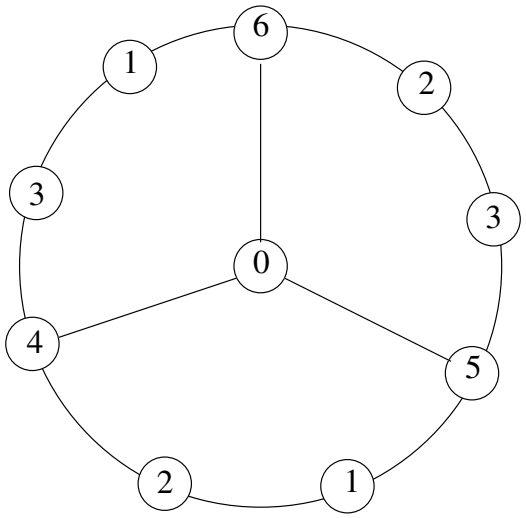
Proposition 6.3 *If a routing system has a dispute ring, then it is not safe under filtering.*

Proof. Assume that a routing system has a dispute ring, defined by $i_1, \ldots, i_m$, and paths $Q_1, \ldots, Q_m$, $P_1, \ldots, P_m$. Then, construct filters such that $\mathcal{F}_i$ contains *only* the paths in that dispute ring. Specifically, $\mathcal{F}_i$ contains the following paths from $\mathcal{P}_i^N$ (where we define $i_{m+1} = i_1$). (1) If i is not in the dispute ring, then $\mathcal{F}_i = \emptyset$. (2) If i is a pivot node on the dispute ring, say $i = i_k$, then $\mathcal{F}_i$ contains exactly two paths: P_k, and $i_k Q_k i_{k+1} P_{k+1} 0$. (3) If i is not a pivot node, but $i \in Q_k$ for some k, then we can write $Q_k = i_k Q_k^1 i Q_k^2 i_{k+1}$. In this case $\mathcal{F}_i$ consists of the single path $i Q_k^2 i_{k+1} P_{k+1} 0$. (4) If i is not a pivot node, but $i \in P_k$ for some k, then we can write $P_k = i_k P_k^1 i P_k^2 0$. In this case, $\mathcal{F}_i$ consists of the single path $i P_k^2 0$. Since each node appears at most once on the dispute ring, the preceding definition uniquely defines $\mathcal{F}_i$ for all nodes i.

There exists at least one consistent path assignment $\boldsymbol{P}_t$ such that some pivot node i_{k-1} uses its most preferred path, $i_{k-1} Q_{k-1} i_k P_k 0$, every other pivot node i_j uses path $i_j P_j 0$, and every other non-pivot node i uses its only available path consistent with this assignment. Then, the following activation sequence will result in an oscillation:

Node	*Ranking*
1	$160 \succ 1240$
2	$240 \succ 2350$
3	$350 \succ 3160$
4	$43160 \succ 40$
5	$51240 \succ 50$
6	$62350 \succ 60$

(a) Routing system



(b) Dispute wheel

Figure 6-11: System that (1) has no dispute ring and (2) is not safe.

1. *Activate node i_k.* Node i_k then switches to its more preferred path, $i_k Q_k i_{k+1} P_{k+1} 0$.
2. *Activate nodes along Q_{k-1} in reverse order, from the node immediately preceding i_k, to i_{k-1}.* All nodes along Q_{k-1} switch to the empty path, ε.
3. *Activate node i_{k-1}.* The path $i_{k-1} Q_{k-1} i_k P_k 0$ is now inconsistent, so i_{k-1} must switch to the path $i_{k-1} P_{k-1} 0$.
4. *Return to Step 1 with k replaced by $k+1$, and iterate again.*

By the fourth step of the iteration above, the new path assignment is "isomorphic" to the initial configuration: now node i_k is using the path $i_k Q_k i_{k+1} P_{k+1} 0$, and every other pivot node i_j is using path $i_j P_j 0$. Thus, as this iteration repeats, the dynamics will ultimately reach node i_k once again with the original path assignment. Note that all paths in this activation sequence are guaranteed to be available and consistent, by the definition of $\mathcal{F}_i$. To make this activation sequence fair, we must also activate the nodes that are not in $P_i \cup Q_i$ for any i in the dispute ring; and the non-pivot nodes in P_i for all i in the dispute ring. The nodes that are not in $P_i \cup Q_i$ for any i have only the path ε available, and each non-pivot node in P_i (for all i) has only one path to the destination available. Therefore, these nodes will never change paths, and do not affect the oscillation. ■

We emphasize that, for simplicity, we reduced the set of filters, $\mathcal{F}_i$, to include only the set of paths that are involved in an oscillation. We note that there will typically be more permissive sets $\mathcal{F}_i$ that will also result in oscillation, because the dispute ring is present in the underlying set of rankings. Our intent is to highlight the most basic case of filtering that can cause an oscillation, given the existence of a dispute ring.

Despite the fact that systems that are safe under filtering are guaranteed not to have a dispute ring, testing for a dispute ring is not sufficient to guarantee that the routing system is safe, because of the following observation

	Path Assignment					
Act.	1	2	3	4	5	6
—	(1 2 4 0)	(2 4 0)	(3 5 0)	(4 0)	(5 0)	(6 0)
5	(1 2 4 0)	(2 4 0)	(3 5 0)	(4 0)	**(5 1 2 4 0)**	(6 0)
1	**(1 6 0)**	(2 4 0)	(3 5 0)	(4 0)	(5 1 2 4 0)	(6 0)
3	(1 6 0)	(2 4 0)	**(3 1 6 0)**	(4 0)	(5 1 2 4 0)	(6 0)
4	(1 6 0)	(2 4 0)	(3 1 6 0)	**(4 3 1 6 0)**	(5 1 2 4 0)	(6 0)
5	(1 6 0)	(2 4 0)	(3 1 6 0)	(4 3 1 6 0)	**(5 0)**	(6 0)
3	(1 6 0)	(2 4 0)	**(3 5 0)**	(4 3 1 6 0)	(5 0)	(6 0)
2	(1 6 0)	**(2 3 5 0)**	(3 5 0)	(4 3 1 6 0)	(5 0)	(6 0)
6	(1 6 0)	(2 3 5 0)	(3 5 0)	(4 3 1 6 0)	(5 0)	**(6 2 3 5 0)**
4	(1 6 0)	(2 3 5 0)	(3 5 0)	**(4 0)**	(5 0)	(6 2 3 5 0)
2	(1 6 0)	**(2 4 0)**	(3 5 0)	(4 0)	(5 0)	(6 2 3 5 0)
1	**(1 2 4 0)**	(2 4 0)	(3 5 0)	(4 0)	(5 0)	(6 2 3 5 0)
5	(1 2 4 0)	(2 4 0)	(3 5 0)	(4 0)	**(5 1 2 4 0)**	(6 2 3 5 0)
6	(1 2 4 0)	(2 4 0)	(3 5 0)	(4 0)	(5 1 2 4 0)	**(6 0)**

Figure 6-12: Activation sequence for unsafe system from Figure 6-11.

Observation 6.5 *Routing systems that have a dispute wheel but do not have a dispute ring may not be safe.*

Example 6.8 Consider the routing system described by Figure 6-11(a) and the corresponding dispute wheel in Figure 6-11(b). Suppose that nodes 1, 2, and 3 prefer two-hop paths over three-hop paths, and the only paths available to nodes are those depicted in the figure. This system is not safe; for example, suppose $\boldsymbol{P}_0 = (1240, 240, 350, 40, 50, 60)$. The system then oscillates as shown in Figure 6-12. However, the system has no dispute ring; in particular, the dispute wheel depicted in Figure 6-11(b) cannot be reduced to a dispute ring. ■

■ 6.5 Autonomy and Safety

In this section, we determine necessary and sufficient constraints on the allowable classes of rankings, such that if each AS autonomously sets its ranking while filtering is unrestricted, the protocol is guaranteed to be safe. We do so by characterizing whether a routing system where rankings are independently specified by each AS can induce either a dispute ring or a dispute wheel.

Any protocol's configurable parameters implicitly constrain the rankings ASes can express. For example, in BGP, the set of protocol parameters is rich enough to allow providers to express essentially any possible ranking over paths. In Section 6.5.1, we axiomatically formulate two properties that should be satisfied by any protocol that respects autonomy: *permutation invariance* and *scale invariance*. The first requires the rankings allowed by the protocol to be independent of node labeling, while the second requires the allowed rankings to scale gracefully as nodes are added to the system. We abstract protocols satisfying these two conditions using the notion of an *autonomous ranking constraint* (ARC) function;

such a function takes the ranking of a single AS as input, and accepts it if that ranking is allowed by the protocol. Observe that *any* protocol that respects the ability of ASes to autonomously choose rankings can be represented by a corresponding ARC function.

In Section 6.5.2, we give two examples of such functions: the shortest hop count ARC function (which only accepts rankings where shorter paths are preferred to longer paths), and the next-hop ARC function (which only accepts next hop rankings). We then determine the class of ARC functions such that, as long as each node independently chooses an acceptable ranking, the resulting *global* routing system will be safe under filtering. In Section 6.5.3, we show that the only ARC functions that are safe under filtering are nearly equivalent to the shortest hop count ARC function.

■ 6.5.1 ARC Functions

In this section, we define an *autonomous ranking constraint* (ARC) function, which serves as an abstraction of the protocol's constraints on allowed rankings over routes. We start by defining a local ranking constraint (RC) function, which takes as input the ranking of a single AS i, $\prec_i^N$ and determines whether that ranking is allowable.

Definition 6.17 (Local RC function) *Given N nodes, a* local ranking constraint (RC) function $\pi(\prec_i)$ *takes as input the ranking of a single AS i over all paths in $\mathcal{P}_i^N$, and returns "accept" if $\prec_i$ is allowed by π, and returns "reject" otherwise. If $\pi(\prec_i) =$ "accept", we will say that $\prec_i$ is π-accepted. If we are given a routing system $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$ where each $\prec_i$ is π-accepted, we will say the routing system is π-accepted.*

Because we are restricting attention to protocols that respect the ability of ASes to choose rankings autonomously, a first condition that must be satisfied is that constraints on rankings should be "local": that is, an AS should not face constraints on allowable rankings due to the rankings chosen by other ASes. For this reason, *local* RC functions act only on the ranking of a single AS. More generally, protocols might place system-wide constraints on the vector of rankings chosen by all ASes; such protocols should be represented by "global" RC functions. Of course, such protocols do not respect autonomy, and so we do not consider them here.

We now define two natural conditions any local RC function that preserves autonomy should satisfy. First, the local RC function's conditions on rankings should provide consistent answers to different ASes, regardless of the *labeling* of the ASes. That is, for the local RC function to preserve uniformity, each AS should be subject to the same constraints on routing policies, and those constraints should not depend on the particular assignment of AS numbers to ASes. For example, suppose the routing system consists of three ASes, and AS 1 has an accepted ranking where it prefers 1230 over 120, and 120 over 10. Then we expect the same ranking should be accepted, even if the labels of nodes are *permuted*. For example, suppose we permute the node labels that $1 \to 2$, $2 \to 3$, and $3 \to 1$. Then node 2 should also have an accepted ranking where it prefers 2310 over 230, and 230 over 20 (because 2310, 230, and 20 are the new paths that result after applying the permutation to 1230, 120, and 10, respectively). If this property were not satisfied, then the set of accepted rankings determined by a local RC function would depend on the global assignment of AS numbers to nodes, not on the characteristics of the individual rankings themselves. We

call this notion *permutation invariance*; to define it precisely, we must proceed through a sequence of definitions, starting with *path permutation*.

Definition 6.18 (Path permutation) *Given N nodes, let σ be a permutation of the nodes $1, \ldots, N$. Then σ induces a* path permutation *on any path $P = ii_1i_2 \ldots i_m j$ from i to j, yielding a new path $\sigma(P) = \sigma(i)\sigma(i_1)\sigma(i_2) \ldots \sigma(i_m)\sigma(j)$ from $\sigma(i)$ to $\sigma(j)$. We always define $\sigma(0) = 0$.*

Definition 6.19 (Ranking permutation) *Given N nodes, let σ be a permutation of the nodes $1, \ldots, N$. Then σ induces a* ranking permutation *on a ranking $\prec_i$ for node i over the paths in $\mathcal{P}_i^N$, yielding a new ranking $\sigma(\prec_i)$ over the paths in $\mathcal{P}_{\sigma(i)}^N$, as follows: If $P_1, P_2 \in \mathcal{P}_i^N$, and $P_1 \prec_i P_2$, then $\sigma(P_1)\sigma(\prec_i)\sigma(P_2)$ (where $\sigma(P_i)$ is the path permutation of path P_i under σ).*

Note that a permutation does not modify the routing system any substantive way, except to *relabel* the nodes, and to relabel the paths and rankings and in a way that is consistent with the relabeling of nodes.

Definition 6.20 (Permutation invariance) *A local RC function π is* permutation invariant *if, given N and a ranking $\prec_i$ for an AS i over all paths in $\mathcal{P}_i^N$, the relation $\prec_i$ is π-accepted if and only if $\sigma(\prec_i)$ is π-accepted, for any permutation σ of $1, \ldots, N$.*

Second, a local RC function should specify conditions for acceptance or rejection of rankings that "scale" appropriately with the number of nodes in the system; we call this property *scale invariance*. Suppose, for example, that a local RC function accepts a ranking $\prec_i$ over $\mathcal{P}_i^N$, when N nodes are in the system. Now suppose that we add nodes to the system, so the total number of nodes is $\hat{N} > N$. As additional nodes are added to the system, additional paths become available as well, and each node i must specify its rankings over the larger set $\mathcal{P}_i^{\hat{N}}$. Informally, scale invariance of the local RC function requires that i should be able to "extend" the ranking $\prec_i$ to an accepted ranking over $\mathcal{P}_i^{\hat{N}}$, without having to modify its existing ranking over $\mathcal{P}_i^N$; otherwise, the properties of allowed rankings would depend on the number of nodes in the global system.

To formalize this concept, we first define a subranking.

Definition 6.21 (Subranking) *Given N nodes, let $\prec_i$ be a ranking for AS i over all paths in $\mathcal{P}_i^N$. Given $\hat{N} > N$, let $\hat{\prec}_i$ be a ranking for AS i over all paths in $\mathcal{P}_i^{\hat{N}}$. Note that $\mathcal{P}_i^N \subset \mathcal{P}_i^{\hat{N}}$. We say that $\prec_i$ is a* subranking *of $\hat{\prec}_i$ if $P_1 \prec_i P_2$ implies $P_1 \hat{\prec}_i P_2$, for all $P_1, P_2 \in \mathcal{P}_i^N$.*

We now define scale invariance.

Definition 6.22 (Scale invariance) *A local RC function π is* scale invariant *if the following condition holds: given any π-accepted ranking $\prec_i$ for AS i over $\mathcal{P}_i^N$, and given any $\hat{N} > N$, there exists at least one π-accepted ranking $\hat{\prec}_i$ over $\mathcal{P}_i^{\hat{N}}$ that has $\prec_i$ as a subranking.*

Permutation invariance guarantees that relabeling nodes does not affect allowed rankings; scale invariance ensures that even as the set of nodes in the network increases, the rankings over previously existing paths should remain valid. Local RC functions that satisfy both permutation invariance and scale invariance correspond to protocols that respect the ability of ASes to autonomously choose rankings; we call such functions *autonomous ranking constraint functions*.

Definition 6.23 (ARC function) *A local RC function is an* autonomous ranking constraint (ARC) function *if it is both permutation invariant and scale invariant.*

We want to derive the conditions under which protocols are guaranteed to be safe under filtering. Given that we use an ARC function as an abstraction of the constraints placed by a protocol on rankings, we would thus like to characterize ARC functions that can ensure safety under filtering of the entire routing system (a global property). For this reason, we extend the definition of "safety under filtering" to cover local RC functions.

Definition 6.24 *Let π be a local RC function. We say that π is* safe under filtering *if all π-accepted routing systems are safe under filtering.*

■ 6.5.2 Examples of ARC Functions

We now present two simple examples of ARC functions: the shortest hop count ARC function, which is guaranteed to be safe, but is not expressive; and the next hop ARC function, which is expressive, but not safe.

Example 6.9 Our first example is the *shortest hop count RC function*, π^{shc}. Given the number of nodes N, the RC function π^{shc} accepts a ranking $\prec_i$ for node i if and only if the relation $\prec_i$ strictly prefers shorter paths (in terms of hop count) over longer ones. Formally, it accepts $\prec_i$, if, for any two paths $P_i, \hat{P}_i \in \mathcal{P}_i^N$ such that length$(P_i) <$ length$(\hat{P}_i)$, $P_i \succ_i \hat{P}_i$. Ties may be broken arbitrarily.

It is not hard to verify that π^{shc} is an ARC function. To check permutation invariance, note that if $\prec_i$ is allowed for node i, then of course for any permutation σ, the ranking $\sigma(\prec_i)$ will also be allowed for node $\sigma(i)$, as $\sigma(\prec_i)$ will also prefer shorter paths to longer paths. Scale invariance is natural: given any shortest hop count ranking $\prec_i$ over $\mathcal{P}_i^N$, and given $\hat{N} > N$, there obviously exists at least one shortest hop count ranking over $\mathcal{P}_i^{\hat{N}}$ that has $\prec_i$ as a subranking. ■

π^{shc} forces all providers to use shortest AS path length, effectively precluding each AS from having any policy expressiveness in choosing rankings (other than when breaking ties). A more flexible set of rankings is allowed by the *next hop RC function* of the next example.

Example 6.10 The *next hop RC function*, π^{nh}, accepts a ranking $\prec_i$ for node i if and only if $\prec_i$ satisfies Equation (6.1) in Section 6.3.1; that is, if $\prec_i$ is a next hop ranking.

π^{nh} is clearly permutation invariant: if $\prec_i$ is a next hop ranking for node i, then clearly $\sigma(\prec_i)$ is a next hop ranking for node $\sigma(i)$. Furthermore, note that any next hop ranking $\prec_i$ is determined entirely by the rankings of node i over each possible next hop, together with tiebreaking choices among routes with the same next hop. Thus, for $\hat{N} > N$, $\prec_i$ can be extended to a next hop ranking over $\mathcal{P}_i^{\hat{N}}$, by extending node i's rankings over each possible next hop, and determining tiebreaking rules for any routes with next hop $N+1, \ldots, \hat{N}$. We conclude that π^{nh} is scale invariant as well, and thus it is an ARC function.

π^{nh} grants greater flexibility in choosing routing policies than under the shortest hop count RC function, π^{shc}, albeit at some cost. With π^{nh}, each AS i will choose a next hop ranking $\prec_i$ without any *global* constraints on the composite vector of next hop rankings

$(\prec_1, \ldots, \prec_N)$ chosen by the nodes. We have shown earlier in Section 6.3.1 that there exist configurations of next hop rankings that may not be stable or safe under filtering; thus, the ARC function π^{nh} can lead to a lack of safety. ■

Next, we use dispute rings and dispute wheels to characterize the class of ARC functions that are safe under filtering. We will prove that this class is closely related to the ARC function π^{shc}.

■ 6.5.3 Impossibility Results

We prove two main results in this section. Informally, the first result can be stated as follows: suppose we are given an ARC function and an accepted ranking such that some n hop path is *less preferred* (*i.e.*, ranked lower) than another path of length at least $n+2$ hops. Then, we can construct an accepted routing system with a dispute ring; *i.e.*, one that is not safe under filtering. The second result states that if some n-hop path is *less preferred* than another path of length at least $n+1$ hops, then there exists a routing system with a dispute wheel (though not necessarily a dispute ring). Note that this result is weaker than our first result, because a dispute wheel does not necessarily imply that the system is not safe under filtering.

We interpret these results as follows: if we are searching for ARC functions that are safe under filtering, we are very nearly restricted to considering only the shortest hop count ARC function, because all paths of n hops *must be more preferred* than paths of at least $n+2$ hops to guarantee safety under filtering, and all paths of n hops must be more preferred than paths of at least $n+1$ hops to prevent dispute wheels.

Our first lemma, which is crucial to proving both of our results, uses permutation invariance to construct a dispute wheel from a single node's rankings. We use a permutation to "replicate" pieces of the dispute wheel until the entire wheel is completed.

To state the lemma, we will require the definition of *period* of a node with respect to a permutation, as well as the period of a permutation. Given a permutation σ on the nodes $1, \ldots, N$, let σ^k denote the permutation that results when σ is applied k times; *e.g.*, $\sigma^2(j) = \sigma(\sigma(j))$, where σ^0 is defined to be σ.

Definition 6.25 (Period) *Given a permutation σ on the nodes $1, \ldots, N$, we define the* period of *i under σ as* $\text{period}_i(\sigma) = \min\{k \geq 1 : \sigma^k(i) = i\}$.

Thus, the period of i is the minimum number of applications of σ required to return to i.

Definition 6.26 (Permutation period) *Given a permutation σ on the nodes $1, \ldots, N$, we define the* period of the permutation *σ as* $\text{period}(\sigma) = \min\{k \geq 1 : \sigma^k(i) = i \text{ for all } i\}$.

Thus, $\text{period}(\sigma)$ *is the minimum number of applications of σ required to recover the original node labeling, and can be computed as the least common multiple of* $\text{period}_i(\sigma)$*, for $1 \leq i \leq N$.*

The following result establishes the conditions under which we can apply a permutation to a π-accepted ranking to obtain a dispute wheel. We use this lemma as a building block for both of the theorems in this section.

Lemma 6.1 *Let π be an ARC function. Suppose there exists a node i with a ranking $\prec_i$ over $\mathcal{P}_i^N$, two paths $R_i, \hat{P}_i \in \mathcal{P}_i^N$, and a permutation σ on $1, \ldots, N$ such that: (1) $\prec_i$ is π-accepted; (2)*

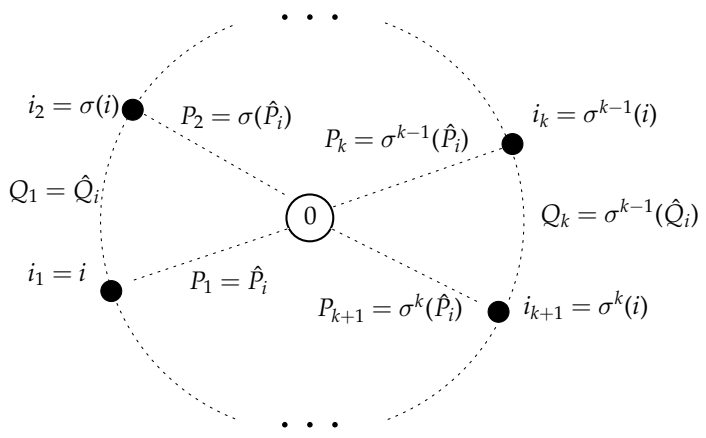


Figure 6-13: Dispute wheel construction for Lemma 6.1.

$R_i \succ_i \hat{P}_i$*; (3)* $\text{period}_i(\sigma) = \text{period}(\sigma)$*; and (4) there exists a path* $\hat{Q}_i$ *from* i *to* $\sigma(i)$ *such that:*

$$R_i = i\hat{Q}_i\sigma(i)\sigma(\hat{P}_i)0. \tag{6.2}$$

Then there exists a π*-accepted routing system with a dispute wheel.*

This dispute wheel is defined by pivot nodes $i_1, \ldots, i_m$*, and paths* $P_1, \ldots, P_m$ *and* $Q_1, \ldots, Q_m$*, where* $m = \text{period}(\sigma)$*, and where for* $k = 1, \ldots, m$*, we have* $i_k = \sigma^{k-1}(i)$*,* $P_k = \sigma^{k-1}(\hat{P}_i)$*, and* $Q_k = \sigma^{k-1}(\hat{Q}_i)$*.*

Proof. Refer to Figure 6-13. The key idea of the proof is that, since $\text{period}_i(\sigma) = \text{period}(\sigma)$, we can repeatedly apply σ to the paths $\hat{Q}_i$ and $\hat{P}_i$ and apply permutation invariance to construct a π-accepted routing system with a dispute wheel.

Let $m = \text{period}(\sigma)$. Define the sequence $i_1, i_2, \ldots, i_m$ by $i_k = \sigma^{k-1}(i)$ for $k = 1, \ldots, m$. Since $\text{period}(\sigma) = \text{period}_i(\sigma)$, the nodes $i_1, \ldots, i_m$ are all distinct. For $k = 1, \ldots, m$, define $\prec_{i_k} = \sigma^{k-1}(\prec_i)$; since the nodes $i_1, \ldots, i_m$ are all distinct, this assignment of rankings to nodes is well defined (*i.e.*, no node is assigned two different rankings). By permutation invariance, since $\prec_i$ is π-accepted, we conclude $\prec_{i_k}$ is π-accepted for all k. For all other nodes j, choose any π-accepted ranking $\prec_j$. Let $\mathcal{F}_j = \mathcal{P}_j^N$ for all nodes j.

This permutation defines a π-accepted routing system $(N, \prec_1, \ldots, \prec_N, \mathcal{F}_1, \ldots, \mathcal{F}_N)$. We now construct a dispute wheel for this system. Define $Q_k = \sigma^{k-1}(\hat{Q}_i)$, and $P_k = \sigma^{k-1}(\hat{P}_i)$, for $k = 1, \ldots, m$. We claim that these definitions yield a dispute wheel.

Since $\mathcal{F}_j = \mathcal{P}_j^N$ for all j, all paths are feasible. Next, since $\hat{Q}_i$ is a path from $i_1 = i$ to $i_2 = \sigma(i)$, we conclude that Q_k is a path from i_k to i_{k+1} for all k (where we define $i_{m+1} = i_1$). We now observe that:

$$\begin{aligned}\sigma^{k-1}(R_i) &= \sigma^{k-1}(i)\sigma^{k-1}(\hat{Q}_i)\sigma^k(i)\sigma^k(\hat{P}_i)0 \\ &= i_k Q_k i_{k+1} P_{k+1} 0.\end{aligned}$$

Finally, since $\prec_{i_k} = \sigma^{k-1}(\prec_i)$ and $R_i \succ_i \hat{P}_i$, we have $\sigma^{k-1}(R_i) \succ_{i_k} \sigma^{k-1}(\hat{P}_i)$. Using the preceding derivation and the fact that $P_k = \sigma^{k-1}(\hat{P}_i)$, we conclude that $i_k Q_k i_{k+1} P_{k+1} 0 \succ_{i_k} i_k P_k 0$, as

required.

Thus, we have established that $i_1, \ldots, i_m$, together with $Q_1, \ldots, Q_m$ and $P_1, \ldots, P_m$, constitute a dispute wheel. ■

The preceding lemma reduces the search for a dispute wheel to a search for a permutation and a π-accepted ranking with the stated properties. Observe from Equation (6.2) that the permutation σ maps the path $\hat{P}_i$ into the "tail" of the path R_i; in applying Lemma 6.1, we will construct a partial permutation by mapping a path $\hat{P}_i$ into the "tail" of R_i as in (6.2), and then we will complete the permutation by adding nodes to the system if necessary so that $\text{period}_i(\sigma) = \text{period}(\sigma)$. We use this approach to prove two theorems; the first states that if an ARC function accepts at least one ranking that prefers an n-hop path less than a path of at least $n+2$ hops, then the ARC function is not safe under filtering.

Theorem 6.1 *Let π be an ARC function. Suppose there exists a node i with π-accepted ranking $\prec_i$, and two paths $R_i, \hat{P}_i \in \mathcal{P}_i^N$ such that* $\text{length}(R_i) > \text{length}(\hat{P}_i) + 1$ *and $R_i \succ_i \hat{P}_i$. Then, π is not safe under filtering.*

Proof. The proof relies on Lemma 6.1 to build a dispute wheel. First, using scale invariance of the ARC function, we show that the stated conditions of the theorem ensure that there exist two paths R_i', $\hat{P}_i'$ such that: $\text{length}(R_i') \geq \text{length}(\hat{P}_i') + 1$; R_i' is more preferred than $\hat{P}_i'$ for some π-accepted ranking; and R_i' and $\hat{P}_i'$ have no nodes in common, other than i and 0. Lemma 6.2 then completes the proof of the theorem through two steps: first, once we have found the paths R_i' and $\hat{P}_i'$, we use them to build a permutation σ such that the conditions of Lemma 6.1 are satisfied; and second, we show that the dispute wheel given by Lemma 6.1 is in fact a dispute ring, by checking that no nodes are repeated around the wheel.

We first construct the paths R_i' and $\hat{P}_i'$ as described in the previous paragraph. Let i, $\prec_i$, R_i, and $\hat{P}_i$ be given as in the theorem. Let $\ell = \text{length}(\hat{P}_i)$; *i.e.*, $\hat{P}_i = iu_1u_2 \ldots u_{\ell-1}0$. We add ℓ new nodes to the routing system, and label them $v_1, \ldots, v_\ell$; let $N' = N + \ell$. By scale invariance, there exists a π-accepted ranking $\prec_i^{N'}$ on the set of paths $\mathcal{P}_i^{N'}$ with $\prec_i$ as a subranking. For such a ranking $\prec_i^{N'}$ we have $R_i \succ_i^{N'} \hat{P}_i$.

But now consider the path $T_i = iv_1 \ldots v_\ell 0$; note that $\text{length}(T_i) = \ell + 1$. Since $R_i \succ_i^{N'} \hat{P}_i$, either $T_i \succ_i^{N'} \hat{P}_i$, or $R_i \succ_i^{N'} T_i$. In the former case, let $R_i' = T_i$, $\hat{P}_i' = \hat{P}_i$; and in the latter case, let $R_i' = R_i$, and $\hat{P}_i' = T_i$. Then $\text{length}(R_i') \geq \text{length}(\hat{P}_i') + 1$, $R_i' \succ_i^{N'} \hat{P}_i'$, and R_i' and $\hat{P}_i'$ have no nodes in common other than i and 0.

The following lemma uses Lemma 6.1 to construct a dispute wheel.

Lemma 6.2 *Let π be an ARC function. Suppose there exists a node i with π-accepted ranking $\prec_i$ over $\mathcal{P}_i^N$, and two paths $R_i, \hat{P}_i \in \mathcal{P}_i^N$ such that:*

1. $\text{length}(R_i) \geq \text{length}(\hat{P}_i) + 1$;
2. *$R_i \succ_i \hat{P}_i$; and*
3. *R_i and $\hat{P}_i$ have no nodes in common other than i and 0.*

Then there exists a π-accepted routing system for which there exists a dispute ring.

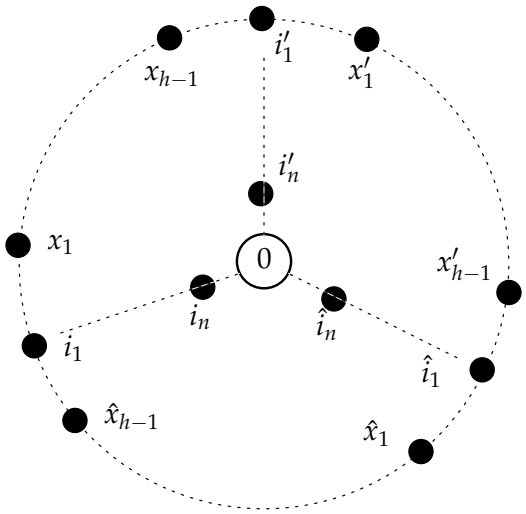


Figure 6-14: Dispute ring construction for Lemma 6.2.

Proof of Lemma. The proof of this lemma proceeds by using scale invariance: we add enough new nodes to the system to allow us to build a permutation such that the conditions of Lemma 6.1 are satisfied. The key insight is that we initially construct the permutation σ by mapping the path $\hat{P}_i$ into the "tail" of the path R_i. We then add enough nodes so that when we complete the definition of σ, we have $\text{period}_i(\sigma) = \text{period}(\sigma)$.

Let $\text{length}(\hat{P}_i) = n$, and let $h = \text{length}(R_i) - n$; note that, by Condition 1 in the statement of the lemma, we know $h \geq 1$. Define $i_1 = i$. We label the nodes so that $\hat{P}_i = i_1 i_2 \dots i_n 0$, and $R_i = i_1 x_1 x_2 \dots x_{h-1} \hat{i}_1 \hat{i}_2 \dots \hat{i}_n 0$. We want to define a permutation σ that will map the path $i_1 \dots i_n 0$ to the tail of R_i, *i.e.*, to the path $\hat{i}_1 \dots \hat{i}_n 0$. However, this does not completely define a permutation, so we must add additional nodes to ensure that $\text{period}_i(\sigma) = \text{period}(\sigma)$.

We add $2(h-1) + n$ additional nodes to the system, labeled $\hat{x}_1, \dots, \hat{x}_{h-1}$, and $i'_1, \dots, i'_n, x'_1, \dots, x'_{h-1}$. By scale invariance, we know there exists at least one π-accepted ranking $\hat{\succ}_i$ over all paths using this larger set of nodes, such that $\hat{\succ}_i$ has $\succ_i$ as a subranking. In particular, since $R_i \succ_i \hat{P}_i$, we have $R_i \hat{\succ}_i \hat{P}_i$. We now define a permutation σ according to the following maps:

$$\begin{aligned} &i_k \to \hat{i}_k \to i'_k \to i_k, \quad k = 1, \dots, n; \\ &x_k \to \hat{x}_k \to x'_k \to x_k, \quad k = 1, \dots, h-1. \end{aligned}$$

That is, $\sigma(i_k) = \hat{i}_k$, $\sigma(\hat{i}_k) = i'_k$, etc. For all nodes j not listed, we define $\sigma(j) = j$. Note that the period of σ is $\text{period}(\sigma) = 3$, and of course $\text{period}_i(\sigma) = \text{period}_{i_1}(\sigma) = 3 = \text{period}(\sigma)$. Finally, note that by definition of σ, we have $R_i = i\hat{Q}_i \sigma(i) \sigma(\hat{P}_i) 0$, where $\hat{Q}_i = i x_1 \dots x_{h-1} \hat{i}_1$.

Thus, the conditions of Lemma 6.1 have been satisfied by the ranking $\hat{\succ}_i$, the paths R_i and $\hat{P}_i$, and the permutation σ; so we know there exists a π-accepted routing system for which there exists a dispute wheel. To complete the proof, we need only check that the dispute wheel is a dispute ring. Note that the wheel has three pivot nodes. Furthermore, to check that no nodes are repeated around the wheel, we simply enumerate the elements of our dispute wheel: $\hat{Q}_i = i_1 x_1 \dots x_{m-1} \hat{i}_1$; $\sigma(\hat{Q}_i) = \hat{i}_1 \hat{x}_1 \dots \hat{x}_{m-1} i'_1$; $\sigma^2(\hat{Q}_i) = i'_1 x'_1 \dots x'_{m-1} i_1$

; $\hat{P}_i = i_1 \ldots i_n 0$; $\sigma(\hat{P}_i) = \hat{i}_1 \ldots \hat{i}_n 0$; and $\sigma^2(\hat{P}_i) = i'_1 \ldots i'_n 0$. It is straightforward to check that these paths constitute a dispute ring: in Figure 6-14, note that the dispute wheel constructed from these paths has no repeated nodes. ■

Lemma 6.2 completes the proof of the theorem: we have shown that if some π-accepted ranking exists satisfying the conditions of the theorem, then using only permutation invariance and scale invariance we can build a π-accepted routing system with a dispute ring. This routing system is then unsafe under filtering, by Proposition 6.3. ■

The preceding theorem suggests that ARC functions that are safe under filtering are closely related to the shortest hop count ARC function, because no rankings can be accepted where n hop paths are less preferred than $(n+k)$-hop paths, for $k \geq 2$. The next theorem draws this relationship even closer, by proving that there exists a dispute wheel if an ARC function accepts any ranking where an n-hop path is less preferred than an $(n+1)$-hop path.

Theorem 6.2 *Let π be an ARC function. Suppose there exists a node i with π-accepted ranking $\prec_i$, and two paths $R_i, \hat{P}_i \in \mathcal{P}_i^N$ such that* $\text{length}(R_i) = \text{length}(\hat{P}_i) + 1$ *and $R_i \succ_i \hat{P}_i$. Then there exists a π-accepted routing system with a dispute wheel.*

Proof. As in the proof of Lemma 6.2, our basic approach is to map the path $\hat{P}_i$ into the "tail" of the path R_i. This partially defines a permutation σ. Using a graphical approach, we show how to add nodes to the system and complete the permutation σ so that $\text{period}_i(\sigma) = \text{period}(\sigma)$. We then apply Lemma 6.1 to conclude there exists a π-accepted routing system with a dispute wheel.

To begin, write $R_i = i i_1 \ldots i_n 0$, and $\hat{P}_i = i v_1 \ldots v_{n-1} 0$. We will partially define a permutation σ, and then add nodes and "complete" the permutation so that σ satisfies the conditions of Lemma 6.1. For all nodes $j \notin R_i \bigcup \hat{P}_i$, we define $\sigma(j) = j$. Let $V = R_i \bigcup \hat{P}_i \setminus \{0\} = \{i, i_1, \ldots, i_n, v_1, \ldots, v_{n-1}\}$; *i.e.*, V is the set of the nonzero nodes in $R_i \bigcup \hat{P}_i$. We define a *directed* graph on the vertex set V, by defining the set of arcs A as follows: $A = \{(i, i_1)\} \bigcup \{(v_k, i_{k+1}) : k = 1, \ldots, n-1\}$. Define the graph $G = (V, A)$.

We can immediately make the following observations about G: (1) each node in V has either exactly one outgoing link and no incoming links; or exactly one incoming link and no outgoing links; or exactly one incoming link and exactly one outgoing link; and (2) from the definition of A, node i has exactly one outgoing link and no incoming links. We interpret the graph G as a partial representation of the permutation σ, by defining $\sigma(j) = k$ if $(j, k) \in A$.

Of course, this only partially defines σ, and we now consider how we should complete the definition of σ. Let $T_1, \ldots T_m$ be the disjoint connected components of G; we write $T_k = (V_k, A_k)$. By the definition of "connected component", we know $V_k \cap V_{k'} = A_k \cap A_{k'} = \emptyset$ for $k \neq k'$. We assume without loss of generality that $i \in V_1$.

Our approach is to first define σ for all the nodes in each connected component T_k, for $k = 2, \ldots, m$. From the observations above, we can enumerate the nodes in V_k as $V_k = \{u_1, u_2, \ldots, u_\ell\}$, such that each u_r has a link to u_{r+1}, for $r = 1, \ldots, \ell - 1$; and either u_ℓ has no outgoing links (in which case T_k is just a "segment") or u_ℓ has a link to u_1 (in which case T_k is a "cycle"). We define $\sigma(u_r) = u_{r+1}$, where we interpret $u_{\ell+1}$ as u_1. Thus, in a segment

or cycle, each node is mapped to its successor; in addition, in a segment, the last node is mapped to the first node. This defines the permutation σ for all nodes, except those in V_1.

To complete the proof, we will add enough nodes and extend the definition of σ so that $\text{period}_i(\sigma) = \text{period}(\sigma)$; we can then apply Lemma 6.1. Note that for all nodes $j \in V_2 \cup \cdots \cup V_m$, we can compute $\text{period}_j(\sigma)$ based on the preceding definition. Let K be the least common multiple of $\text{period}_j(\sigma)$, over all $j \in V_2 \cup \cdots \cup V_m$. We then *add* nodes to the system, and in particular to the set V_1, until $|V_1|$ (*i.e.*, the number of nodes in V_1) is a *multiple* of K. Let the nodes added be $z_1, \ldots, z_s$; these nodes will eventually become the pivots of the dispute wheel. We know that A_1 must be of the form $\{(i, i_1), (i_1, u_1), \ldots, (u_{\ell-1}, u_\ell)\}$ for some nodes $u_1, \ldots, u_\ell \in V$. We define σ as follows: $\sigma(i) = i_1$; $\sigma(i_1) = u_1$; $\sigma(u_r) = u_{r+1}$, for $r = 1, \ldots, \ell - 1$; $\sigma(u_\ell) = z_1$; $\sigma(z_r) = z_{r+1}$, for $1 \leq r \leq s - 1$; and $\sigma(z_s) = i$. Thus, it is as if we added the nodes $z_1, \ldots, z_s$, and turned the segment T_1 into a cycle. Since the length of this cycle is a multiple of K, it is clear that $\text{period}(\sigma)$ is a multiple of K, and $\text{period}_i(\sigma) = \text{period}(\sigma)$.

By scale invariance, even though we have added nodes to the system, we can extend $\prec_i$ to a π-accepted ranking over the resulting larger set of paths, while maintaining the preference of R_i over $\hat{P}_i$ for node i. Furthermore, recalling our initial definition of the arc set A, it is clear that we have $R_i = ii_1 \ldots i_n 0 = i\sigma(i)\sigma(v_1) \ldots \sigma(v_{n-1})0 = i\sigma(i)\sigma(\hat{P}_i)0$. Thus, we can apply Lemma 6.1, with $\hat{Q}_i = \emptyset$, to conclude there exists a π-accepted routing system with a dispute wheel. ■

The preceding results should not be interpreted as suggesting that we cannot find a routing system that is safe under filtering, where nodes prefer $(n + k)$-hop paths over n-hop paths. Indeed, from Figure 6-10, there are routing systems where nodes prefer 3-hop paths over 1-hop paths, and yet the system is safe under filtering. However, checking whether such systems are safe under filtering requires global verification; the theorems we have presented suggest that safety under filtering cannot be guaranteed through local verification alone, if some nodes are allowed to prefer $(n + k)$-hop paths over n-hop paths.

Furthermore, the two results in this section highlight the importance of dispute rings. Theorem 6.1 gives the strong result that an ARC function that allows some $(n + k)$-hop path to be more preferred than an n-hop path cannot guarantee safety under filtering, if $k \geq 2$. Theorem 6.2 only guarantees the existence of a dispute wheel if an ARC function that allows some $(n + 1)$-hop path to be more preferred than an n-hop path—we cannot draw conclusions regarding the stability or safety of a routing system on the basis of the existence of a dispute wheel (see Figure 6-10).

■ 6.6 Implications: Possibilities for Guaranteeing Safety

In light of the results in this chapter, a natural question to ask is whether they are positive or negative. In one sense, our results are grim, because they suggest that if BGP remains in its current form and each AS retains complete autonomy and filtering expressiveness (*i.e.*, the only realistic scenario for the foreseeable future), then the routing system cannot be guaranteed to be safe unless each AS constrains its rankings over available paths to those that are consistent with shortest hop count (or, alternatively, preferences that are based on consistent edge weights).

On the other hand, our results suggest several clear directions for designing a policy-based routing protocol that is guaranteed to be safe. Although we presented the ARC function as a proof technique, such a function could be implemented in practice to restrict the rankings that operators specify in operational networks. We foresee a few possibilities: (1) the routing protocol remains as it is today, and the constraints derived in Theorems 6.1 and 6.2 are checked by a tool that statically detects configuration faults (*e.g.*, *rcc*, as described in Chapter 4); (2) the routing protocol is modified to prevent operators from expressing rankings that violate these constraints in the first place; (3) ASes use dynamic analysis to detect when a safety violation has taken place (*e.g.*, [59]); or (4) a protocol is designed whereby ASes exchange enough information about their rankings to enable detection or prevention of safety violations, but not enough to reveal their strategic business decisions. We have not yet fully evaluated the merits of each approach in this work, but we briefly speculate on their advantages and disadvantages.

6.6.1 Static Fault Detection with *rcc*

Enforcing the ranking constraints from Section 6.5.3 locally at each AS using a static analysis tool like *rcc* (Chapter 4) would require no changes to the existing routing protocols and configuration languages. Unfortunately, the results in Theorems 6.1 and 6.2 only provide global guarantees if *every* AS abides by the constraints: as a result, there may be little incentive for one AS to constrain its policies if other ASes do not abide by the same constraints. As such, while using *rcc* to verify the necessary conditions for safety under filtering could guarantee safety without requiring any modifications to the protocol, it would require ASes to abide by constraints that are not enforceable, since an AS has no incentive to restrict its filtering in this way if other ASes do not.

6.6.2 A New Routing Protocol that Restricts Expressiveness

Implementing the ranking constraints by restricting the protocol "knobs" requires wholesale changes to both the configuration language and the routing protocol, but it would provide absolute guarantees for safety while still preserving the autonomy of each AS. The difficult design question involves determining exactly *how* the knobs should be restricted. HLP [129], a recent proposal for replacing BGP, proposes that these knobs constrain the policy constraints to conform to those suggested by Gao and Rexford [47], but the examples in Section 6.1 suggest that this prescription is too restrictive. Our results suggest another possibility: a protocol that stipulates that rankings be consistent with weighted shortest paths but affords each AS some flexibility in actually setting those edge weights. A recently developed framework called "metarouting" may also be useful in designing a protocol that is safe under filtering [57]. We now explore each of these possibilities, which are not mutually exclusive.

Option #1: Implement Rankings with Shortest Paths and "Knob Tweaking"

Our results in Section 6.3.2 suggest one possible direction: there we show that routing using preferences derived from edge weights is guaranteed to be stable. Suppose each AS ranks paths based on the sum of edge weights to the destination and adjusts weights on its incident outgoing edges to neighboring ASes. Rankings would then be derived from

the total path cost, but an AS might still retain enough flexibility to control the next-hop AS en route to the destination. Such an approach could ensure that the protocol is safe on short timescales, while allowing "policy disputes" to occur on longer timescales, out-of-band from the routing protocol. Of course, we must explore whether this apparently more restrictive language could still implement the policies that operators want to express. Furthermore, a more restrictive policy language would guarantee safety, but would likely cause routing to oscillate on a slower timescale as operators observe the routing protocol converging to undesirable paths. We believe that this design decision is exactly the right one: the routing protocol *should* converge on a fast timescale and accurately reflect network topology, while policy conflicts should be resolved on slower, "human" timescales.

One possible problem with routing protocol whose rankings are derived from consistent edge weights is its lack of modularity. For example, a failure or policy change in a remote part of the network may require a network operator to re-tweak the edge weights on the edges incident to his AS simply to maintain a desired rankings over paths. Of course, in practice, an operator would not find such continual knob-tweaking acceptable. These adjustments could be performed automatically (*i.e.*, the weights required to achieve some ranking could be automatically computed and reconfigured). Whether such a solution would introduce too many routing messages into the global routing system, as well as what the precise technique should be for adjusting these weights, are areas for future exploration.

Option #2: Use "Metarouting" to Design a Strictly Monotonic Protocol

Metarouting [57] is a recently developed framework based on Sobrinho's path algebras [125] that allows a routing protocol designer to design a routing protocol using a "metalanguage" that specifies an algebra. Informally, an *algebra* is a specification of the labels that links and paths can assume, the output of composing these labels with some operator, and a ranking function that specifies which labels are preferred over others. Sobrinho's work observed that any *strictly monotonic* algebra (*i.e.*, where the ranking function prefers a path $P_i = (v_i, \ldots, d)$ over any path PP_i) is guaranteed to be safe [125]; metarouting shows how to compose multiple algebras to obtain a resulting algebra that is strictly monotonic and provides a language for constructing these algebras.

Metarouting is a framework for *analyzing* the safety routing protocols, but it remains to be seen whether it can be used to *design* any expressive policy-based routing protocol. For example, because next-hop rankings are not monotonic, a routing protocol that is based on an algebra whose ranking function uses next-hop rankings as the primary criterion is not safe. Additionally, a strictly monotonic routing protocol whose ranking function is not derived from consistent edge weights still requires restrictions on filtering. For the reasons we discussed earlier in this chapter, it is doubtful that operators would accept any routing protocol that hard-wires filtering restrictions into the protocol itself. Further, metarouting only explores sufficient conditions for safety; it is unclear whether it captures all interesting routing protocols that satisfy safety (particularly since it excludes protocols based on next-hop rankings, for example).

6.6.3 Restrict Autonomy by Exposing Rankings

Another possibility for guaranteeing safety is to impose restrictions on autonomy. Some previous work detects safety violations using out-of-band distributed detection mechanisms [75, 83]. Recent work notes that safety may still be satisfied if only a single AS deviates from the ranking and filtering constraints of Gao and Rexford (as summarized in Table 2-3) and presents a distributed computation that securely determines the number of ASes that violate the constraints without revealing exactly which ASes are actually in violation [83]. Jaggard and Ramachandran also proposed mechanisms for out-of-band distributed detection of a dispute wheel [75].

A reasonable area for future work would be to further explore these types of approaches. For one, these distributed detection algorithms do not offer any recourse for *resolving* the conflicting policies that give rise to the safety violation, which would ultimately require restrictions on autonomy. For example, suppose every AS were permitted to express next-hop rankings. In this case, what information must each AS reveal (and to whom) about its rankings so that the resulting system could detect and resolve safety violations? In a similar vein, it may be reasonable to exploit the trust relationships among groups of ASes (*e.g.*, one AS may trust one of its neighbors, but not others) to design a protocol that guarantees safety but only requires partial revelation of rankings to some subset of all ASes.

6.6.4 Detect Safety Violations using Dynamic Analysis

Rather than restricting rankings and filters to prevent safety violations, the routing protocol could rely on analysis of routing protocol dynamics to detect when safety has been violated. Previous work has focused on such techniques. For example, Griffin *et al.* proposed a "safe path vector protocol" [59]; this chapter suggests a modification to BGP where each AS includes an explanation of why it changed routes (*e.g.*, because the route it changed to was more or less preferred than the route it was previously using). The protocol facilitates detection of safety problems, but, in the process, it exposes the potentially private ranking functions of each AS and provides no mechanism for resolving conflicts in rankings. More generally, analyzing routing protocol dynamics could help identify safety violations, even without modifications to the routing protocol itself, although such identification would likely require analysis of the dynamics from multiple vantage points in the AS graph.

6.7 Summary

This chapter explored the fundamental tradeoff between the expressiveness of rankings and routing safety, presuming that each AS retains complete autonomy and filtering expressiveness, and presented the first study of the effects of filtering on safety. We make the following contributions.

1. We showed that next-hop rankings are not safe; we also observed that although rankings based on a globally consistent weighting of paths are safe under filtering, even minor generalizations of the weighting function compromise safety.
2. We defined a *dispute ring* and show that any routing system that has a dispute ring

is not safe under filtering. Our results are the first necessary conditions concerning safety.

3. We showed that, providing for complete autonomy and filtering expressiveness, the class of allowable rankings that guarantee safety is effectively ranking based on variants of weighted shortest paths. We also explored the implications of these findings for the design of future interdomain routing protocols.

This chapter continued the theme of exploring the dynamic properties of Internet routing by analyzing static properties of the routing configuration. The fact that we can determine *anything* about the Internet routing dynamics by analyzing the static rankings of each AS independently is rather remarkable. It is even more noteworthy that we can guarantee safety, a property of the *global* routing system, by analyzing the static configurations of each AS independently. Of course, guaranteeing safety does in fact require placing very strict constraints on each AS's rankings, which is why we advocated relaxing these constraints to some degree. An altogether different approach would be to try to detect these properties at runtime, by analyzing the routing messages themselves (*i.e.*, *dynamic analysis*). The next chapter briefly explores how other techniques, such as dynamic analysis, can complement static analysis.

Why does everyone talk about the past? All that counts is tomorrow's game.

- Roberto Clemente

CHAPTER 7
Conclusion

Despite the fact that communication on the Internet relies on correct and predictable operation of the routing protocols, today's Internet routing infrastructure is surprisingly fragile. The behavior of Internet routing depends heavily on how today's *de facto* standard Internet routing protocol, BGP, is configured. Although BGP's configuration is precisely what allows operators to realize complex economic and policy goals, the resulting "programmability" of the protocol creates the potential for incorrect and unpredictable behavior. This dissertation has presented *proactive* techniques for improving the correctness and predictability of Internet routing. In this chapter, we briefly review the causes for incorrect and unpredictable behavior. After summarizing the contributions of this dissertation in Section 7.2, we return in Section 7.3 to the lessons learned from this work (see Section 1.6) and propose some possible steps forward based on these lessons. Section 7.4 concludes.

■ 7.1 Reasons for Correctness and Predictability Problems

Routing on the Internet involves the interaction between tens of thousands of independently operated networks, or autonomous systems (ASes), each of which may have anywhere from one to hundreds of independently configured routers. Although a network operator configures each router independently, the behavior of the protocol may in fact depend on the configurations of multiple routers, as well as the dependencies between these configurations. These inter-router dependencies make configuring a network of routers akin to writing a very large distributed program. In light of the fact that, until now, there have been no tools or techniques to help network operators reason about the configuration of the network *as a whole*, it is not surprising that network operators make mistakes configuring their routers.

Even if network operators could be assured that the protocol would always behave "correctly", they would still have no assurance regarding what would actually happen to traffic in response to a particular routing configuration. Various protocol artifacts (*e.g.*, the MED attribute and route reflection), as well as the interaction of BGP with interior routing protocols make it hard to predict route assignments and traffic flow. As such, to determine the effects of a potential configuration change on the flow of traffic (and,

hence, to determine whether such a configuration change meets traffic engineering goals), an operator currently has no choice but to test that configuration change on a running network.

Additionally, the operation of Internet routing depends on the interactions of configurations across multiple administrative domains, but network operators typically do not have access to the configurations of these neighboring domains. As a result of this opacity, a network operator has no way to guarantee that the policies configured in his own network will not conflict with those in neighboring networks; worse yet, that operator has no way to debug a problem caused by conflicting configurations when one does arise. For example, as described in Chapters 3 and 6, ASes with conflicting policies can violate safety: unlike a shortest paths routing protocol, policy-based routing protocols may never converge to a stable outcome if the policies in neighboring ASes conflict [135].

7.2 Summary of Contributions

This dissertation posited that the complexity of routing protocol configuration is a major impediment to correct and predictable Internet routing and has presented *proactive* techniques for improving the correctness and predictability of Internet routing. These techniques are based on a rigorous correctness specification for routing, which we presented in Chapter 3. This specification has three aspects: route validity, path visibility, and safety. Although we have explored the correctness specification in the context of today's Internet routing system, we hope that it will prove useful for evaluating the behavior of any routing protocol, particularly ones involving complex routing policies (*e.g.*, modifications to or replacements for BGP).

Using this correctness specification as a guide, we have developed tools and techniques that allow a network operator to detect problems and predict routing protocol behavior *before* the configuration is deployed by analyzing the static configuration files of the routers within a single AS. In particular, we presented two such tools:

- *rcc* , presented in Chapter 4, detects faults in the router configurations within a single AS (*i.e.*, violations of route validity, path visibility) by analyzing the static configuration files. *rcc* has been downloaded by over seventy network operators to date and is available for download at `http://nms.lcs.mit.edu/rcc/`.
- The route prediction algorithms presented in Chapter 5 (and the associated prototype, the *routing sandbox*) allows an operator to determine which routes each router will ultimately select, given only a static snapshot of the router configuration and the routes learned via eBGP. We developed a prototype of this tool and determined that our route computation algorithms are both accurate and fast enough to be used to compute routes for a large tier-1 ISP.

In addition to providing these tools to help network operators configure the Internet routing system, we also derive the constraints that each AS's rankings must be subject to in order for the global routing system to satisfy safety, presuming that each AS retains complete autonomy in how it specifies both rankings and filters (Chapter 6). While the constraints we derived could certainly be implemented in *rcc*, it turns out that they are too restrictive: guaranteeing safety under such circumstances essentially requires that each AS

constrain its rankings to be consistent with shortest paths routing. We also speculated on the implications of these results for the design of future Internet routing systems.

7.3 Moving Forward from the Lessons Learned

Some researchers have lamented the "ossification" of the Internet routing system and have argued that it is difficult to have real impact in this area [108]. Indeed, much networking research either analyzes the problems with the routing system or proposes solutions that work within (or around) the limitations of BGP. Arguably, the growing research interest in overlay networks reflects the community's general frustration with the difficulty of effecting real, substantive change in the Internet's underlying routing system. However, fixing the routing system is an important and challenging goal that the networking research community needs to address. We believe that there are many open and *important* problems related to these challenges where networking researchers can have impact.

We now revisit the lessons learned (Section 1.6) and discuss possible steps forward in terms of the two philosophies posed at the end of Chapter 1: (1) leaving the routing infrastructure largely unchanged and focusing on tools and techniques to make the existing infrastructure more correct and predictable; or (2) modifying the existing architecture and infrastructure so that it is inherently more correct and predictable. In addition to revisiting the lessons from Section 1.6, in Section 7.3.5, we consider imminent open problems in Internet routing security (*i.e.*, how to guarantee correctness and predictability in the face of adversaries).

7.3.1 Static configuration analysis detects many faults

One of the important lessons to take away from our experience with *rcc* is that static configuration analysis detects many faults in deployed routing configurations. Of course, this lesson also begs the question of identifying and detecting the types of faults that cannot be detected with static analysis alone. In general, there appear to be many classes of events that are of interest to network operators that will not be apparent from static analysis alone but can nevertheless exploit static analysis to help network operators drill down on the source of an observed problem:

- **Shifts in traffic flow.** Operators can use existing monitoring infrastructure and anomaly detection tools to help them detect shifts in traffic flow, but identifying the cause of a traffic shift typically requires knowledge of many factors, including the configuration. For example, to determine whether a traffic shift was caused by a change in demand or a change in routing, an operator may want to see how the routes advertised from neighboring ASes have changed or determine whether the configuration has changed. The techniques described in Chapter 5 can help determine how these changes would likely affect the flow of traffic.
- **Contract violations.** Statically detecting faults in one's own network provides no guarantees that neighboring networks will behave correctly or scrupulously. In these cases, dynamic analysis is critical, but static analysis also plays a crucial role in helping operators identify perpetrators. For example, recent work has developed an algorithm to detect inconsistent route advertisements from neighboring ASes [35]; this

technique exploits static analysis of import policies to discount cases where an apparent inconsistent route advertisement was in fact caused by the AS's own import policies.

- **Performance degradations.** While static analysis alone cannot detect performance degradations, it can help detect (or eliminate) possible causes for these degradations. For example, the checks for path visibility in iBGP (*e.g.*, Theorem 3.4) can help a network operator quickly determine the source of dropped packets and can also eliminate some potential causes of the problem. In the case of routing failures that affect end-to-end performance, the fastest way to determine that a problem exists in the first place is to observe the end-to-end performance of Internet paths, but static analysis may still play a useful diagnostic role in many cases.
- **Route instability.** In other cases, such as protocol oscillation cause by interaction between MED and route reflectors or by interaction of policies between ASes, static analysis is only useful for determining whether a route oscillation *might* happen. Careful analysis of the routing announcements themselves, in conjunction with analysis of local policies, may help determine when such oscillations actually arise and merit attention.
- **Security problems.** Static analysis can help defend against some routing security problems (*e.g.*, advertising private address space into the public Internet), but detecting route hijacks or otherwise invalid route announcements in general is a challenging problem. Real-time analysis and detection of suspicious routing announcements will play an important role in Internet routing security.

Dynamic analysis may play an important complementary role in solving each of these problems, but an important open question is recognizing that, for each case, a different "signal" may be necessary. Existing work that analyzes traffic patterns for anomalies faces a somewhat less challenging problem because the timeseries to analyze is obvious (*i.e.*, amount of traffic per unit time). With routing, the timeseries is less obvious. For example, when is it appropriate to analyze number of prefixes advertised per unit time versus the total number of prefixes announced by an AS versus the AS path length of a prefix over time, and what techniques are most appropriate for analyzing these signals?

■ 7.3.2 Distributed configuration causes faults

Because *distributed* configuration is responsible for many configuration faults, a natural possible step forward is to consider ways to configure the network from a single, centralized point. Refactoring configuration in this fashion would allow a network operator to configure the *network*, rather than configuring routers. This proposal seems appealing, but several challenges stand in its way:

- What type of language should such a centralized scheme use?
- What types of techniques would network operators be most likely to adopt?
- How can the expression of high-level policies be separated from the low-level mechanisms that implement those policies?

Approaches to fault detection fall into two general categories: those that take the existing configuration languages as given and analyze deployed configuration for faults (analysis), and those that try to design higher-level constructs to automatically generate low-level router configuration (synthesis). Both approaches can benefit from better configuration languages that more closely correspond to the network operator's intent. For example, analysis techniques could greatly benefit if they could check the actual routing configuration against a specification of the intended behavior of the routing protocol. Such a specification could serve as a high-level language for automatically *generating* low-level configuration.

Most network operators are trained on existing configuration languages, and learning a new process for configuring routers has a high cost. This situation suggests that the best way to spur adoption would be to design a configuration language that allows an operator to more easily construct configuration fragments from templates, which could be customized according to higher-level macros.

In some sense, these two questions are closely related: network operators will most likely be willing to adopt a configuration language that is similar to the ones they use today. In light of this propensity, one possible approach is to generate a system that gives an operator the semblance of configuring each independent router but is, in reality, a centralized system that is responsible for pushing configuration to the routers but can automatically check for invariants that must hold network-wide. Such a system is actually not that different from the *rcc* paradigm: the main difference is that the network would be checked *before* it is deployed on running routers and the operator would alter configurations in a centralized repository, rather than on the individual routers. Once configuration were centralized in this fashion, however, the system could recognize commonalities ("design patterns") that existed across various routers and BGP sessions and assist an operator in automatically generating these common configurations.

■ 7.3.3 Safety + Autonomy ⇒ Tight restrictions on expressiveness

Our results concerning safety in Chapter 6 suggest that, if ASes are given both complete freedom to establish business contracts (*i.e.*, set filters) and complete autonomy (both reasonably design requirements for the foreseeable future), then the only Internet routing protocol that is guaranteed to converge on a fast timescale is one where paths are ranked according to consistent path costs (*e.g.*, shortest path routing).

This result suggests that, moving forward, future Internet routing protocols should either: (1) constrain rankings so that they are derived from consistent path costs but still provide operators the latitude they need to implement business relationships; (2) relax autonomy, designing the protocol in such a way that reveals the fact that a policy conflict exists without divulging sensitive information; or (3) devise more reasonable constraints on filtering that might permit more flexible rankings. In any case, it is clear that the Internet routing protocol must both converge on a fast timescale, for reasons of both performance and diagnostics, and somehow make it possible for ASes to express preferences over the next-hop AS to which they send traffic (and make it possible to recognize whether such rankings conflict).

The work in this dissertation examines one end of this spectrum: that is, we explore how expressiveness can rankings be, providing for unlimited autonomy and unrestricted filters.

Our results suggest that, in this regime, ASes have relatively little expressiveness over rankings, which implies that exploring how to relax requirements such as autonomy may be worthwhile. In Section 6.7, we discussed possible several future directions. We now revisit the strawman proposal for a new routing protocol that we proposed in Section 6.6 and pose some open questions related to this proposal.

Suppose that every edge in the inter-AS graph had a corresponding weight, that the cost of a path was the sum of all inter-AS edge weights from source to destination, and that each AS was required to prefer the path with lowest path cost. To give operators some latitude in setting rankings, further suppose that each AS had the freedom to set the edge weights for only those edges that were incident to its own AS in the graph (*i.e.*, those edges from it to the next-hop AS towards the destination). Now, we know that such a protocol is guaranteed to converge on a fast timescale, because all rankings are based on shortest paths routing. On the other hand, this strawman proposal introduces two obvious problems:

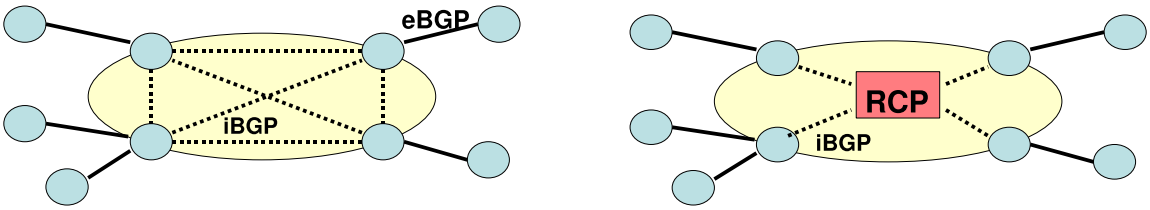
- **Potentially poor isolation.** When an inter-AS edge fails, an operator might have to re-tune the weights on the edges incident to his AS to guarantee that traffic will continue flowing through the chosen next-hop AS for that destination. Is it possible to either design a scheme that automatically tunes these weights for the operator and is still guaranteed to be stable or, alternatively, a way of setting edge weights that is relatively robust to these types of failures?
- **Policy disputes.** This strawman proposal does not eliminate policy disputes entirely; rather, it *moves* the policy disputes so that the protocol oscillates on a slower timescale. Of course, there is another advantage to this slower-timescale oscillation, in that the policy dispute will be apparent from the continually increasing path costs. Can the Internet routing infrastructure incorporate a protocol that automatically detects these disputes and facilitates renegotiation of business relationships to resolve them?

■ 7.3.4 Protocol design should consider correctness and predictability

This dissertation focuses on techniques for improving correctness and predictability within the current Internet routing architecture, but a more fruitful approach in the long term is to design the routing protocol or architecture with an eye towards preventing incorrect and unpredictable behavior in the first place. The routing architecture should be designed to explicitly provide correct and predictable behavior. Correctness and predictability should not depend on the routing protocol's configuration; rather, they should be *intrinsic* to the protocol's design.

Separate routing from forwarding

Our work on detecting faults in router configuration suggests that many router configuration faults result from distributed nature of the configuration. The configuration of some policies introduces dependencies between the configurations of multiple routers: the correct operation of a filter on one router may be based on a label that a different router attaches to the route. Additionally, many of the problems with BGP's correctness and predictability result from today's techniques for disseminating routes within an AS (*i.e.*,



(a) In a conventional iBGP topology, every router in the AS participates in IGP and iBGP and selects routes independently.

(b) RCP receives the iBGP routes and the IGP topology from the routers in the AS and computes routes on behalf of every router in the AS, and assigns those routes using iBGP [31].

Figure 7-1: Overview of the Routing Control Platform (RCP) architecture.

iBGP route reflection). Existing techniques for disseminating BGP routes were designed as scalable, easily deployable extensions to BGP, but did not consider correctness or predictability as first-order concerns. Route reflectors [5] eliminate the need for a full mesh between iBGP speakers, but they do not correctly emulate a full mesh iBGP configuration and, as a result, may cause the protocol to violate basic correctness properties, as described in Chapter 3. After first surveying some techniques that make minor modifications to iBGP to improve correctness and predictability, we explore how more radical modifications to iBGP—specifically, those that separate Internet routing from the individual routers—might help network operators and protocol designers cope with the complexity of Internet routing's techniques for achieving policy and scalability.

Previous work has suggested making small modifications to the way that route reflectors advertise routes to guarantee various correctness properties. RFC 1863 proposed that route servers forward *all* routes to clients, rather than just a single best route [66] and suggested using an "advertiser" attribute to allow recipients to know who advertised the routes. Basu *et al.* proposed a small modification to guarantee safety: instead of having route reflectors only a single route to their clients, this work proposes that route reflectors should advertise all routes that are equally good up to the MED step in the selection process (Table 2-2, Chapter 2) [4]. In Chapter 3 (Section 3.2.1), we also note that a similar modification would also guarantee route validity. Work on BGP scalable transport (BST) has proposed other changes to iBGP that prevent oscillations [74]. Unfortunately, because these proposals require modifying BGP (which implies either convincing router vendors to implement the change, effecting the change through a standards body such as the Internet Engineering Task Force, or both), they have not been widely adopted.

Rather than modifying the deployed routing infrastructure, the functions of Internet routing could reside in a separate system that gathers information about the topology (*e.g.*, via iBGP and IGP), computes the route that each router should use on behalf of each router, and assigns the appropriate route to each router. The idea of having a control system that is separate from the infrastructure that is responsible for forwarding packets (*i.e.*, the routers) is the central idea of the Routing Control Platform (RCP) [13, 31] (see Figure 7-1), as well as work on a "more versatile route reflector" that computes different routing decisions on behalf of its client routers [9]. The IETF ForCES working group has also proposed a frame-

work that separates an individual network element into separate control and forwarding elements, which can communicate over a variety of media (*e.g.*, a backplane, Ethernet, etc.) [42]. The framework dictates that routing protocols be implemented in the control elements [144].

Because RCP has a view of the AS-wide iBGP and IGP topologies, it can assign routes to each router in a way that guarantees that the routing protocol satisfies various properties, such as those from the correctness specification in Chapter 3. RCP may also simplify routing configuration, because configuration can be done on an AS-wide basis, rather than router-by-router. Rather than implementing these policies with specifications of mechanism that are distributed across the routers themselves, RCP could allow an operator to specify a policy *for the entire AS* and be agnostic to the mechanisms that actually implement the policy on the routers themselves.

Separating routing state from the routers can potentially introduce additional robustness, scalability, speed, and consistency problems. The RCP architecture must address these challenges to be viable. We have implemented a preliminary prototype of RCP as an extension to Quagga [111] to examine the feasibility of having RCP control route selection for all of the routers in a large tier-1 ISP [13]. To study the other feasibility issues, we are presently implementing RCP as extensions to the XORP software router [65]; we plan to make this prototype available to the research community.

Ultimately, RCP can facilitate innovation because it allows the logic of route selection to be located in a system that is separate from the infrastructure that is responsible for forwarding data traffic. Moving routing into a logically centralized (albeit replicated) system creates more flexibility in how routes are exchanged between RCP nodes in neighboring ASes: RCP nodes in adjacent ASes could use BGP to exchange these routes, although they could conceivably use any protocol. RCP could also expose more information about the routing topology to applications such as overlay networks, as well as give these networks more control over the IP-level routes that are actually used.

Moving routing functionality into a system like RCP is not the only way to separate routing from forwarding: others have advocated removing routing from the AS entirely by moving routing complexity to end hosts, which query route servers to discover routes [80, 145]. Although these projects share our goal of separating routing complexity from the infrastructure, RCP has the added benefit of simplifying intra-AS routing. Recent work has also proposed working around the existing infrastructure, using an overlay to improve BGP's robustness [1, 50].

Provide direct control over traffic

Much research, both in Chapter 5 and in previous work [39], has focused on developing predictive models of network-wide routing, typically to assist network operators in traffic engineering. A system like RCP can provide *real-time control* of BGP routes rather than modeling the BGP routes in today's routing system. That is, rather than trying to infer the outcome of BGP route selection or otherwise monitor the routing protocol (as many existing systems do [73, 105, 124]), RCP can *control* the outcome, thus directly controlling how traffic flows.

Although RCP provides direct control over traffic, it does not provide any facilities for helping operators determine *how* that traffic should be controlled. Similarly, Chapter 5 pre-

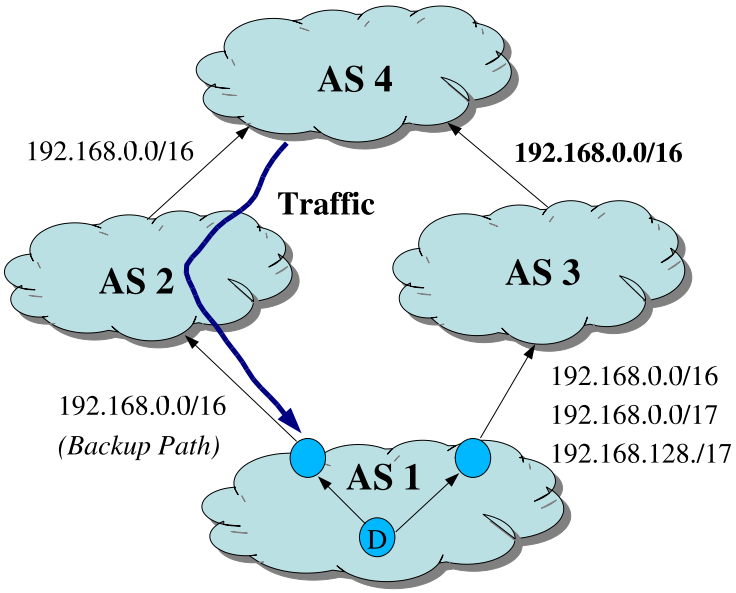


Figure 7-2: Aggregation can interfere with an AS's attempt to control inbound traffic. In this example, AS 1 announces more specific routes to AS 3 in an attempt to have inbound traffic enter via that AS. AS 3, on the other hand may aggregate (or simply filter) those more specific routes to save routing table space, thus interfering with AS 1's intent.

sented algorithms that help network operators predict the effects of configuration changes on the flow of traffic through the network, but these algorithms do not help network operators actually discover a routing configuration that achieves their intended objective (*e.g.*, relieving persistent congestion, minimizing congestion on peering links, etc.). Unlike *intra*domain routing optimization, for which there are existing tools and algorithms to help operators tune parameters [15, 39], *inter*domain routing is much more difficult to optimize: the search space is much larger, and local configuration changes can have global effects that alter the traffic volumes coming *to* that AS from neighboring ASes. Our previous work proposes some guidelines for helping network operators make configuration changes that are less likely to cause unpredictable effects [32], but the design techniques for efficiently finding optimal (or even "good") configuration settings remain open.

Avoid manipulable routing objects

Recall from Chapter 3 that we considered a destination d to be an immutable "handle" for a route that referred to a set of endpoints; we cited IP prefixes as the destination d in the case of Internet routing. Unfortunately, prefixes are *not* immutable: a router in one AS can *aggregate* IP prefixes that are contiguous in the address space taking two routes, $(d_1 \rightarrow v_i)$ and $(d_2 \rightarrow v_j)$ and combining them into a single route, $(d \rightarrow v_k)$, such that any packets destined for endpoints in either d_1 or d_2 will be forwarded according to the route $(d \rightarrow v_k)$. This creates complications for the definition of an induced path (Definition 3.3), since a route to any endpoint by be induced by routes for different destination handles at different routers.

Not only do these manipulable routing objects create problems for reasoning about

properties of the routing protocol, but they also create practical problems. For one, they create a tension between control over inbound traffic and scalability. ASes typically exploit the fact that routers will forward traffic based on the longest matching IP prefix in the routing table and send more specific routing information along links where it wishes to attract traffic. This technique allows an AS to control how traffic reaches its AS from other places on the Internet. On the other hand, to control routing table size, some ASes may combine multiple more specific routes into a single shorter route using a process called aggregation. Aggregation dramatically reduces routing table size because Internet addressing is typically hierarchical. Unfortunately, aggregating more specific prefixes can also interfere with an ASes attempt to control how traffic reaches its network, as shown in Figure 7-2 [142]. Thus, ASes must either aggregate routes at the risk of interfering with the traffic engineering goals of other ASes (the common practice today [11]) or maintain larger routing tables without receiving compensation for the incurring the extra overhead.

Finally, recent work has observed that, because IP prefixes can be manipulated by routers in other ASes, they often do not accurately reflect a set of destinations that is atomically either reachable or unreachable [43]. This characteristic can cause violations of route validity (Definition 3.7) when a portion of endpoints contained within some destination become unreachable but the route advertising the IP prefix for those endpoints is not withdrawn.

In light of these problems presented by aggregation, one reasonable design principle for future Internet routing systems may be to make the destination that is referred to by a route an immutable property of the route, as a recent proposal has suggested [138].

■ 7.3.5 Consider the effects of adversaries on correctness and predictability

This dissertation addresses the challenges for correctness and predictability that are introduced unintentionally. A growing threat to the Internet routing infrastructure, however, is that posed by adversaries. For example, malicious parties may attempt to inject false routing information in an attempt to divert or blackhole traffic. Routing security has been studied in some detail [107], but *interdomain* routing security is particularly difficult because interdomain routing must support complex policy. In addition to securing routing information, the infrastructure should also ultimately provide guarantees about where the traffic actually goes—in other words, whether the *induced path* (Definition 3.3, Chapter 3) actually corresponds to the information about the path that is carried in the route.

Control-Plane Security

Attacks against the control plane have received considerable attention in recent years. The IETF has established a working group in routing protocol security, RPSEC [122]; recent drafts have addressed threats to the Internet routing system and BGP and outlined security requirements [93]. BGP does not provide any support for controlling route announcements. Securing the control plane boils down to two tasks: *origin authentication*, which verifies that the AS that is announcing ("originating") the prefix actually has legitimate ownership over the address space for that prefix; and *path authentication*, which verifies that the sequence of ASes that the routing advertisement traversed corresponds to the sequence of ASes in the AS path.

Various proposals provide either origin authentication [141], path authentication [70], or both [77]. Unfortunately, these schemes require the existence of a public key infrastructure (PKI), a central authority that manages key distribution, ownership, and revocation; PKIs are cumbersome and difficult to maintain. Thus, developing a routing protocol that automatically verifies the ownership of some address space without requiring a centralized, trusted verification or lookup infrastructure remains an important unsolved problem.

Even an Internet routing architecture that provides both origin and path authentication still does not enable an AS to verify that the route it receives is one that it *should be receiving*. That is, while they allow an AS to verify that a route announcement traversed a particular sequence of ASes, they do not provide any mechanism to allow an AS to verify that the sequence of ASes is one that is sensible. Answering this question is critical for routing security. If an AS has no way to determine whether a sequence of ASes is sensible, then it is possible for a single malicious organization to establish a new AS and have that AS buy transit from the upstream AS of a source and destination, respectively. An important unsolved problem involves characterizing the types of bogus routes that an AS can detect with the limited information available today, as well as determining the additional information that should be added to the routing protocol that could assist this detection without revealing sensitive business relationships.

Data-Plane Security

Even if an AS could verify that the routes it receives were authentic and policy-compliant, it still has no way to verify that packets actually traverse the same ASes as those in the route's AS path [88]. We note that the AS path was never intended as an indication of the ASes that traffic will actually traverse. Rather, the AS path was only ever intended for loop detection (*i.e.*, so that an AS would not readvertise a route that it had already learned) and as a coarse metric for choosing shorter routes over longer ones. Nevertheless, providing some guarantees over where traffic will (or won't) travel appears to be a desirable goal for Internet routing. Allowing an AS or end host to verify that a route's AS path matches the actual forwarding path, or, more generally, allowing it to ascertain the sequence of ASes that traffic to or from some destination traverses are important capabilities that any future Internet routing system should provide.

■ 7.4 Concluding Remarks

This dissertation has (1) presented a correctness specification for Internet routing; (2) developed tools and techniques to check for violations of this correctness specification in real-world routing configuration; (3) exploited the correctness specification to design algorithms that predict the outcome of BGP route selection for the set of routers within an AS; and (4) derived the first conditions for guaranteeing a *global* correctness property, safety, that do not require global knowledge of business relationships or topology.

The current version of the Internet's routing protocol, BGP, has been deployed for nearly a decade and has been modified and extended numerous times (*e.g.*, with route reflection). More striking is the plethora of proposals that have not been deployed. In light of these proposals, the key challenge is to be able to separate the good protocol modifications from

the bad ones and to determine how these various modifications interact with each other. The correctness specification presented in this dissertation facilitates this type of reasoning. Furthermore, the proactive techniques that this dissertation has advocated and developed can help resolve the tension between flexibility and complexity that exists in any complex routing protocol. While we have presented the properties of the correctness specification in the context of BGP, it is our hope that this specification can serve as a guide for both evaluating and designing new routing protocols that explicitly provide correctness and predictability.

References

[1] S. Agarwal, C. Chuah, and R. H. Katz. OPCA: Robust Interdomain Policy Routing and Traffic Control. In *Proc. 6th International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, NY, Apr. 2003. (Cited on page 176.)

[2] C. Alaettinoglu et al. *Routing Policy Specification Language (RPSL)*. Internet Engineering Task Force, June 1999. RFC 2622. (Cited on page 140.)

[3] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Experience with an Evolving Overlay Network Testbed. *ACM Computer Communications Review*, 33(3):13–19, July 2003. (Cited on page 46.)

[4] A. Basu et al. Route Oscillations in IBGP with Route Reflection. In *Proc. ACM SIGCOMM*, pages 235–247, Pittsburgh, PA, Aug. 2002. (Cited on pages 60, 75, 132, 133 and 175.)

[5] T. Bates, R. Chandra, and E. Chen. *BGP Route Reflection - An Alternative to Full Mesh IBGP*. Internet Engineering Task Force, Apr. 2000. RFC 2796. (Cited on pages 42 and 175.)

[6] I. V. Beijnum. *BGP*. O'Reilly and Associates, Sept. 2002. (Cited on pages 34 and 75.)

[7] BGP++ Home Page. `http://www.ece.gatech.edu/research/labs/MANIACS/BGP++/`, 2005. BGP simulator for ns-2. (Cited on pages 24, 50 and 100.)

[8] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal Verification of Standards for Distance Vector Routing Protocols. *Journal of the ACM*, 49(4):538–576, July 2002. (Cited on page 49.)

[9] O. Bonaventure, S. Uhlig, and B. Quoitin. *The Case for More Versatile BGP Route Reflectors*. Internet Engineering Task Force, July 2004. `http://totem.info.ucl.ac.be/publications/draft-bonaventure-bgp-route-reflectors-00.html` Work in progress, expired November 2004. (Cited on pages 60, 64, 133 and 175.)

[10] Private communication with Randy Bush, May 2004. (Cited on page 139.)

[11] R. Bush. Re: Verio decides which parts of the Internet to drop. http://www.merit.edu/mail.archives/nanog/1999-12/msg00022.html, Dec. 1999. (Cited on page 178.)

[12] R. Bush, T. Griffin, Z. M. Mao, E. Purpus, and D. Stutsbach. Happy Packets: Some Initial Results. http://www.nanog.org/mtg-0405/pdf/bush.pdf, May 2004. NANOG 31. (Cited on page 45.)

[13] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and K. van der Merwe. Design and Implementation of a Routing Control Platform. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–28, Boston, MA, May 2005. (Cited on pages 51, 60, 64, 93, 132, 175 and 176.)

[14] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The Cutting EDGE of IP Router Configuration. In *Proc. 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, pages 21–26, Cambridge, MA, Nov. 2003. (Cited on page 49.)

[15] Cariden multiprotocol automation and traffic engineering. http://www.cariden.com/products/marcom/mate_overview_0401.pdf, 2005. (Cited on pages 47, 50 and 177.)

[16] C-BGP. http://cbgp.info.ucl.ac.be/, 2005. (Cited on pages 50, 51 and 100.)

[17] Cisco BGP Best Path Selection Algorithm. http://www.cisco.com/warp/public/459/25.shtml. (Cited on page 35.)

[18] How BGP Routers Use the Multi-Exit Discriminator for Best Path Selection. http://www.cisco.com/warp/public/459/37.html. (Cited on page 101.)

[19] Cisco IOS Master Commands List, Release 12.3. http://cisco.com/univercd/cc/td/doc/product/software/ios123/123mindx/crgindx.htm. (Cited on pages 22, 33 and 39.)

[20] Software Systems to Manage Large Networks: A Challenge and Opportunity, Apr. 2005. http://www.csail.mit.edu/events/eventcalendar/calendar.php?show=event&id=375. (Cited on page 49.)

[21] Team Cymru bogon route server project. http://www.cymru.com/BGP/bogon-rs.html. (Cited on pages 78 and 92.)

[22] B. Davie and Y. Rekhter. *MPLS: Technology and Applications*. Academic Press, San Diego, CA, 2000. (Cited on page 54.)

[23] R. Dube. A Comparison of Scaling Techniques for BGP. *ACM Computer Communications Review*, 29(3):44–46, July 1999. (Cited on pages 61 and 75.)

[24] Cisco IOS IP Command Reference, ebgp-multihop. http://www.cisco.com/en/US/products/sw/iosswrel/ps1835/products_command_reference_chapter09186a00800ca79d.html, 2005. (Cited on page 41.)

[25] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 57–72, Banff, Canada, Oct. 2001. (Cited on page 83.)

[26] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. *Generic Routing Encapsulation (GRE)*. Internet Engineering Task Force, Mar. 2000. RFC 2784. (Cited on page 54.)

[27] N. Feamster. Practical Verification Techniques for Wide-Area Routing. In *Proc. 2nd ACM Workshop on Hot Topics in Networks (Hotnets-II)*, pages 87–92, Cambridge, MA, Nov. 2003. (Cited on page 49.)

[28] N. Feamster, D. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the Effects of Internet Path Faults on Reactive Routing. In *Proc. ACM SIGMETRICS*, pages 126–137, San Diego, CA, June 2003. (Cited on pages 45 and 46.)

[29] N. Feamster and H. Balakrishnan. Towards a Logic for Wide-Area Internet Routing. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 289–300, Karlsruhe, Germany, Aug. 2003. (Cited on pages 9, 75 and 81.)

[30] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 43–56, Boston, MA, May 2005. (Cited on pages 9 and 134.)

[31] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and K. van der Merwe. The Case for Separating Routing from Routers. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 5–12, Portland, OR, Sept. 2004. (Cited on pages 9, 51, 60, 64, 93, 132, 133 and 175.)

[32] N. Feamster, J. Borkenhagen, and J. Rexford. Guidelines for Interdomain Traffic Engineering. *ACM Computer Communications Review*, 33(5):19–30, Oct. 2003. (Cited on pages 17, 37, 51, 99, 126, 127, 134, 145 and 177.)

[33] N. Feamster, R. Johari, and H. Balakrishnan. The Implications of Autonomy for Stable Policy Routing. In *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005. (Cited on pages 9 and 103.)

[34] N. Feamster, J. Jung, and H. Balakrishnan. An Empirical Study of "Bogon" Route Advertisements. *ACM Computer Communications Review*, 35(1):63–70, Nov. 2004. (Cited on page 92.)

[35] N. Feamster, Z. M. Mao, and J. Rexford. BorderGuard: Detecting Cold Potatoes from Peers. In *Proc. ACM SIGCOMM Internet Measurement Conference*, pages 213–218, Taormina, Sicily, Italy, Oct. 2004. (Cited on pages 75, 84, 91 and 171.)

[36] N. Feamster and J. Rexford. Network-Wide BGP Route Prediction for Traffic Engineering. In *Proc. SPIE ITCom*, volume 4868, pages 55–68, Boston, MA, Aug. 2002. (Cited on pages 9 and 21.)

[37] N. Feamster, J. Winick, and J. Rexford. A Model of BGP Routing for Network Engineering. In *Proc. ACM SIGMETRICS*, pages 331–342, New York, NY, June 2004. (Cited on pages 9, 51, 75 and 98.)

[38] J. Feigenbaum, R. Sami, and S. Shenker. Mechanism Design for Policy Routing. In *ACM Symposium on Principles of Distributed Computing*, pages 11–20, 2004. (Cited on pages 146 and 147.)

[39] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. NetScope: Traffic Engineering for IP Networks. *IEEE Network*, 14(2):11–19, Mar. 2000. (Cited on pages 50, 176 and 177.)

[40] 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 Transactions on Networking*, 9(3):257–270, June 2001. (Cited on page 134.)

[41] A. Feldmann and J. Rexford. IP Network Configuration for Intradomain Traffic Engineering. *IEEE Network*, 15(5):46–57, Sept. 2001. (Cited on pages 20 and 49.)

[42] Forwarding and Control Element Separation (ForCES) Charter. `http://www.ietf.org/html.charters/forces-charter.html`. (Cited on page 176.)

[43] M. Freedman, M. Vutukuru, N. Feamster, and H. Balakrishnan. Geographic Locality of IP Prefixes. In *Proc. ACM SIGCOMM Internet Measurement Conference*, New Orleans, LA, Oct. 2005. (Cited on page 178.)

[44] V. Fuller, T. Li, J. Yu, and K. Varadhan. *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*. Internet Engineering Task Force, Sept. 1993. RFC 1519. (Cited on page 54.)

[45] L. Gao. On Inferring Automonous System Relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, Dec. 2001. (Cited on page 34.)

[46] L. Gao, T. G. Griffin, and J. Rexford. Inherently Safe Backup Routing with BGP. In *Proc. IEEE INFOCOM*, pages 547–556, Anchorage, AK, Apr. 2001. (Cited on pages 75 and 138.)

[47] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, pages 681–692, Dec. 2001. (Cited on pages 21, 50, 51, 103, 120, 138, 140, 143 and 164.)

[48] R. Gao, C. Dovrolis, and E. Zegura. Interdomain Ingress Traffic Engineering through Optimized AS-Path Prepending. In *Proceedings of IFIP Networking conference*, Waterloo, Canada, May 2005. (Cited on page 37.)

[49] P. Godefroid. Model Checking for Programming Languages using VeriSoft. In *Proc. ACM Symposium on Principles of Programming Languages*, pages 174–186, Paris, France, 1997. (Cited on page 49.)

[50] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around BGP: An Incremental Approach to Improving Security and Accuracy in Interdomain Routing. In *Proc. NDSS*, San Diego, CA, Feb. 2003. (Cited on page 176.)

[51] M. Gouda and M. Schneider. Maximizable Routing Metrics. *IEEE/ACM Transactions on Networking*, 11(4):663–675, Aug. 2003. (Cited on page 141.)

[52] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W. Lee. An Architecture for Stable, Analyzable Internet Routing. *IEEE Network*, 13(1):29–35, January/February 1999. (Cited on page 140.)

[53] R. Govindan, C. Alaettinoglu, K. Varadhan, and D. Estrin. Route Servers for Inter-Domain Routing. *Computer Networks and ISDN Systems*, 30:1157–1174, 1998. (Cited on page 140.)

[54] T. Griffin, A. Jaggard, and V. Ramacandran. Design Principles of Policy Languages for Path Vector Protocols. In *Proc. ACM SIGCOMM*, pages 61–72, Karlsruhe, Germany, Aug. 2003. (Cited on page 138.)

[55] T. Griffin and B. J. Premore. An Experimental Analysis of BGP Convergence Time. In *IEEE International Conference on Network Protocols (ICNP)*, Riverside, CA, Nov. 2001. (Cited on pages 45 and 46.)

[56] T. Griffin, F. B. Shepherd, , and G. Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking*, 10(1):232–243, 2002. (Cited on pages 17, 26, 50, 81, 136, 137, 138, 140, 148, 149, 150 and 151.)

[57] T. Griffin and J. L. Sobrhino. Metarouting. In *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005. (Cited on pages 137, 138, 140, 149, 164 and 165.)

[58] T. Griffin and G. Wilfong. An Analysis of BGP Convergence Properties. In *Proc. ACM SIGCOMM*, pages 277–288, Cambridge, MA, Sept. 1999. (Cited on pages 75 and 138.)

[59] T. Griffin and G. Wilfong. A Safe Path Vector Protocol. In *Proc. IEEE INFOCOM*, pages 490–499, Tel Aviv, Israel, Mar. 2000. (Cited on pages 63, 72, 140, 164 and 166.)

[60] T. Griffin and G. Wilfong. Analysis of the MED Oscillation Problem in BGP. In *IEEE International Conference on Network Protocols (ICNP)*, Paris, France, Nov. 2002. (Cited on page 51.)

[61] T. Griffin and G. Wilfong. On the Correctness of IBGP Configuration. In *Proc. ACM SIGCOMM*, pages 17–29, Pittsburgh, PA, Aug. 2002. (Cited on pages 42, 45, 51, 70, 77, 79, 94, 102, 105, 117, 120, 132, 133 and 138.)

[62] K. Hafner and M. Lyon. *Where Wizards Stay Up Late: The Origins of the Internet*. Simon and Schuster, 1996. (Cited on page 38.)

[63] J. Hajek. Automatically Verified Data Transfer Protocols. In *Proc. ICCC*, pages 749–756, 1978. (Cited on page 49.)

[64] S. Halabi and D. McPherson. *Internet Routing Architectures*. Cisco Press, second edition, 2001. (Cited on page 34.)

[65] M. Handley, O. Hudson, and E. Kohler. XORP: An Open Platform for Network Research. In *Proc. 1st ACM Workshop on Hot Topics in Networks (Hotnets-I)*, pages 53–57, Princeton, NJ, Oct. 2002. (Cited on page 176.)

[66] D. Haskin. A BGP/IDRP Route Server alternative to a full mesh routing. RFC 1863, Oct. 1995. (Cited on page 175.)

[67] C. Hedrick. *Routing Information Protocol*. Internet Engineering Task Force, June 1988. RFC 1058. (Cited on page 33.)

[68] *HP OpenView Products*, 2005. (Cited on page 48.)

[69] *HP Route Analytics Management System*, 2005. (Cited on pages 47 and 48.)

[70] Y.-C. Hu, A. Perrig, and M. Sirbu. SPV: Secure Path Vector Routing for Securing BGP. In *Proc. ACM SIGCOMM*, pages 179–192, Portland, OR, Aug. 2004. (Cited on page 179.)

[71] IANA. *Special Use IPv4 Addresses*. Internet Engineering Task Force, Sept. 2002. RFC 3330. (Cited on page 93.)

[72] Intelliden R-Series. `http://www.intelliden.com/page.asp?id=Products`, 2005. (Cited on pages 47 and 48.)

[73] Ipsum Networks RouteDynamics. `http://www.cisco.com/warp/public/732/partnerpgm/docs/ipsum_routedynamics.pdf`, 2005. (Cited on pages 47, 48 and 176.)

[74] V. Jacobson, C. Alaettinoglu, and K. Poduri. BST—BGP Scalable Transport. In *NANOG 27*, Phoenix, AZ, Feb. 2003. `http://www.nanog.org/mtg-0302/ppt/van.pdf`. (Cited on page 175.)

[75] A. D. Jaggard and V. Ramachandran. Robustness of Class-Based Path Vector Systems. In *IEEE International Conference on Network Protocols (ICNP)*, Berlin, Germany, Nov. 2004. (Cited on pages 138, 140 and 166.)

[76] JUNOS 7.3 Routing Protocols and Policies Command Reference. `http://www.juniper.net/techpubs/software/junos/junos73/swcmdref73-protocols/frameset.htm`. (Cited on page 33.)

[77] S. Kent, C. Lynn, and K. Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications (J-SAC)*, 18(4):582–592, Apr. 2000. (Cited on page 179.)

[78] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian. Delayed Internet Routing Convergence. *IEEE/ACM Transactions on Networking*, 9(3):293–306, June 2001. (Cited on pages 17, 45, 46, 50 and 75.)

[79] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Wide-Area Network Failures. In *Proc. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing*, page 278, Washingon, DC, June 1999. (Cited on page 102.)

[80] K. Lakshminarayanan, I. Stoica, and S. Shenker. Routing as a Service. Technical Report UCB-CS-04-1327, UC Berkeley, 2004. (Cited on page 176.)

[81] Level3 Hit. `http://www.broadbandreports.com/shownews/39381`, Feb. 2004. (Cited on page 22.)

[82] D. Linsalata. 12/8 Problems? `http://www.merit.edu/mail.archives/nanog/2005-09/msg00295.html`, Sept. 2005. (Cited on page 22.)

[83] S. Machiraju and R. Katz. Verifying Global Invariants in Multi-Provider Distributed Systems. In *Proc. 3nd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, pages 149–154, San Diego, CA, Nov. 2004. (Cited on pages 30, 140 and 166.)

[84] O. Maennel, A. Feldmann, C. Reiser, R. Volk, and H. Boehm. Network-Wide Inter-Domain Routing Policies: Design and Realization. In *NANOG 34*, Seattle, WA, May 2005. (Cited on page 49.)

[85] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP Misconfiguration. In *Proc. ACM SIGCOMM*, pages 3–17, Pittsburgh, PA, Aug. 2002. (Cited on pages 17, 44, 45, 75 and 87.)

[86] Z. M. Mao, R. Govindan, G. Varghese, and R. Katz. Route Flap Damping Exacerbates Internet Routing Convergence. In *Proc. ACM SIGCOMM*, pages 221–233, Pittsburgh, PA, Aug. 2002. (Cited on pages 45 and 46.)

[87] Z. M. Mao, T. Griffin, and R. Bush. BGP Beacons. In *Proc. ACM SIGCOMM Internet Measurement Conference*, pages 1–14, Miami, FL, Oct. 2003. (Cited on page 46.)

[88] Z. M. Mao, J. Rexford, J. Wang, and R. Katz. Towards an Accurate AS-Level Traceroute Tool. In *Proc. ACM SIGCOMM*, pages 365–378, Karlsruhe, Germany, Aug. 2003. (Cited on pages 35 and 179.)

[89] D. McPherson, V. Gill, D. Walton, and A. Retana. *Border Gateway Protocol (BGP) Persistent Route Oscillation Condition*. Internet Engineering Task Force, Aug. 2002. RFC 3345. (Cited on page 67.)

[90] How, Why Microsoft Went Down. `http://wired-vig.wired.com/news/print/0,1294,41412,00.html`, Jan. 2001. (Cited on page 22.)

[91] J. Moy. *OSPF Version 2*, Mar. 1994. RFC 1583. (Cited on pages 33 and 99.)

[92] Multiprotocol Label Switching (MPLS). `http://www.ietf.org/html.charters/mpls-charter.html`. (Cited on page 54.)

[93] S. Murphy, A. Barbir, and Y. Yang. *Generic Threats to Routing Protocols*. Internet Engineering Task Force, Oct. 2004. http://www.ietf.org/internet-drafts/draft-ietf-rpsec-routing-threats-07.txt, expired April 2005. (Cited on pages 17 and 178.)

[94] M. Musuvathi and D. Engler. Some Lessons from Using Static Analysis and Software Model Checking for Bug Finding. In *Workshop on Software Model Checking*, Boulder, CO, July 2003. (Cited on pages 49 and 80.)

[95] M. Musuvathi and D. Engler. Model Checking Large Network Protocol Implementations. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, pages 155–168, San Francisco, CA, Mar. 2004. (Cited on page 49.)

[96] The North American Network Operators' Group mailing list archive. http://www.cctec.com/maillists/nanog/. (Cited on pages 22 and 75.)

[97] S. Narain. Network Configuration Management via Model Finding. In *Proc. 19th USENIX Large Installation Systems Administration Conference*, San Diego, CA, Dec. 2005. (Cited on pages 30 and 49.)

[98] Network Configuration (netconf). http://www.ietf.org/html.charters/netconf-charter.html. (Cited on page 49.)

[99] W. Norton. Internet Service Providers and Peering. http://www.equinix.com/pdf/whitepapers/PeeringWP.2.pdf. (Cited on pages 34 and 75.)

[100] Opnet NetDoctor. http://opnet.com/products/modules/netdoctor.htm, 2005. (Cited on pages 47 and 50.)

[101] Opnet NetDoctor White Paper. http://www.opnet.com/products/brochures/Netdoctor.pdf, 2005. (Cited on page 47.)

[102] Opnet SP Guru. http://www.opnet.com/products/spguru/SPGuru_brochure.pdf, 2005. (Cited on pages 47 and 50.)

[103] Opsware Network Automation System. http://www.opsware.com/products/networkautomation/, 2005. (Cited on pages 47 and 48.)

[104] D. Oran. *OSI IS-IS intra-domain routing protocol*. Internet Engineering Task Force, Feb. 1990. RFC 1142. (Cited on pages 33 and 99.)

[105] Packet Design Route Explorer. http://www.packetdesign.com/products/rex.htm, 2005. (Cited on pages 47 and 176.)

[106] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997. (Cited on pages 45 and 46.)

[107] R. Perlman. *Network Layer Protocols with Byzantine Robustness*. PhD thesis, Massachusetts Institute of Technology, Oct. 1988. MIT-LCS-TR-429. http://www.lcs.mit.edu/publications/specpub.php?id=997. (Cited on page 178.)

[108] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse through Virtualization. In *Proc. 3nd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, pages 23–28, San Diego, CA, Nov. 2004. (Cited on page 171.)

[109] J. B. Postel. *Transmission Control Protocol*. Internet Engineering Task Force, Sept. 1981. RFC 793. (Cited on page 34.)

[110] X. Qie and S. Narain. Using Service Grammar to Diagnose BGP Configuration Errors. In *Proc. 17th USENIX Large Installation Systems Administration Conference*, pages 237–246, San Diego, CA, Oct. 2003. (Cited on page 48.)

[111] Quagga software router. `http://www.quagga.net/`. (Cited on page 176.)

[112] B. Quoitin, C. Pelsser, O. Bonaventure, and S. Uhlig. A Performance Evaluation of BGP-based Traffic Engineering. *International Journal of Network Management*, 15(3), May 2005. (Cited on page 37.)

[113] Really Awesome New Cisco ConfIg Differ (RANCID). `http://www.shrubbery.net/rancid/`, 2004. (Cited on pages 46 and 85.)

[114] Redcell Netconfig. `http://www.doradosoftware.com/products/netConfig.jsp`, 2005. (Cited on pages 47 and 48.)

[115] Voyence VoyenceControl! `http://www.voyence.com/solutions/index.shtml`, 2005. (Cited on pages 47 and 48.)

[116] Reef Point. `http://www.reefpoint.com/site/content/firewalls.asp`, 2005. (Cited on page 49.)

[117] Y. Rekhter and T. Li. *An Architecture for IP Address Allocation with CIDR*. Internet Engineering Task Force, Sept. 1993. RFC 1518. (Cited on page 54.)

[118] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, Mar. 1995. RFC 1771. (Cited on pages 19, 21, 33, 34 and 35.)

[119] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, Oct. 2004. `http://www.ietf.org/internet-drafts/draft-ietf-idr-bgp4-26.txt` Work in progress, expired April 2005. (Cited on pages 33, 34 and 35.)

[120] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, pages 197–202, Marseille, France, Nov. 2002. (Cited on pages 102 and 132.)

[121] Router Glitch Cuts Net Access. `http://news.com.com/2100-1033-279235.html`, Apr. 1997. (Cited on pages 22 and 91.)

[122] Routing Protocol Security (RPSEC) Charter. `http://www.ietf.org/html.charters/rpsec-charter.html`. (Cited on page 178.)

[123] A. Shaikh and A. Greenberg. A Case Study of OSPF Behavior in a Large Enterprise Network. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, pages 217–230, Marseille, France, Nov. 2002. (Cited on page 20.)

[124] A. Shaikh and A. Greenberg. OSPF Monitoring: Architecture, Design, and Deployment Experience. In *Proc. First Symposium on Networked Systems Design and Implementation (NSDI)*, pages 57–70, San Francisco, CA, Mar. 2004. (Cited on pages 20 and 176.)

[125] J. L. Sobrinho. Network Routing with Path Vector Protocols: Theory and Applications. In *Proc. ACM SIGCOMM*, pages 49–60, Karlsruhe, Germany, Aug. 2003. (Cited on pages 50, 138, 140 and 165.)

[126] N. Spring, R. Mahajan, and T. Anderson. Quantifying the Causes of Path Inflation. In *Proc. ACM SIGCOMM*, pages 113–124, Karlsruhe, Germany, Aug. 2003. (Cited on page 91.)

[127] SSFNet. `http://www.ssfnet.org/`, 2003. (Cited on pages 24, 50, 51 and 100.)

[128] J. Stewart. *BGP4*. Addison-Wesley, Reading, MA, 1998. (Cited on page 34.)

[129] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica. HLP: A Next-generation Interdomain Routing Protocol. In *Proc. ACM SIGCOMM*, Philadelphia, PA, Aug. 2005. (Cited on page 164.)

[130] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of Hot-Potato Routing in IP Networks. In *Proc. ACM SIGMETRICS*, pages 307–319, New York, NY, June 2004. (Cited on page 38.)

[131] P. Traina, D. McPherson, and J. Scudder. *Autonomous System Confederations for BGP*. Internet Engineering Task Force, Feb. 2001. RFC 3065. (Cited on page 42.)

[132] Tripwire for network devices. `http://www.tripwire.com/products/network_devices/index.cfm`, 2005. (Cited on pages 47 and 48.)

[133] S. Uhlig and O. Bonaventure. Designing BGP-based Outbound Traffic Engineering Techniques for Stub ASes. *ACM Computer Communications Review*, 34(5):89–106, 2004. (Cited on page 51.)

[134] BGP config donation. `http://www.cs.washington.edu/research/networking/policy-inference/donation.html`. (Cited on page 88.)

[135] K. Varadhan, R. Govindan, and D. Estrin. Persistent Route Oscillations in Inter-Domain Routing. Technical Report 96-631, USC/ISI, Feb. 1996. (Cited on pages 25, 26, 136, 138 and 170.)

[136] K. Varadhan, R. Govindan, and D. Estrin. Persistent Route Oscillations in Inter-Domain Routing. *Computer Networks*, 32(1):1–16, 2000. (Cited on pages 75 and 138.)

[137] C. Villamizar, R. Chandra, and R. Govindan. *BGP Route Flap Damping*. Internet Engineering Task Force, Nov. 1998. RFC 2439. (Cited on page 103.)

[138] M. Vutukuru, N. Feamster, M. Walfish, H. Balakrishnan, and S. Shenker. Revisiting Internet Addressing: Back to the Future! Unpublished manuscript. (Cited on page 178.)

[139] M. Vutukuru, P. Valiant, S. Kopparty, and H. Balakrishnan. How to Construct a Correct and Scalable iBGP Configuration. Technical Report MIT-LCS-TR-996, MIT Computer Science and Artificial Intelligence Laboratory, Aug. 2005. `http://www.lcs.mit.edu/publications/specpub.php?id=1787`. (Cited on pages 62 and 73.)

[140] Wide-Area Network Design Laboratory (WANDL) IP Analysis Tools (IPAT). `http://wandl.com/html/ipat/IPAT_new.cfm`, 2005. (Cited on page 47.)

[141] R. White. *Architecture and Deployment Considerations for Secure Origin BGP (soBGP)*. IETF, May 2005. `http://www.ietf.org/internet-drafts/draft-white-sobgp-architecture-01.txt`. Work in progress, expires November 2005. (Cited on page 179.)

[142] R. White, B. Akyol, and N. Feamster. *Considerations in Validating the Path in Routing Protocols*. IETF, June 2005. `http://www.ietf.org/internet-drafts/draft-white-pathconsiderations-05.txt` Work in progress, expires December 2005. (Cited on page 178.)

[143] WorldCom suffers widespread Internet outage. `http://www.usatoday.com/tech/news/2002-10-03-net-outage_x.htm`, Oct. 2002. (Cited on page 22.)

[144] L. Yang et al. Forwarding and Control Element Separation (ForCES) Framework. RFC 3746, Apr. 2004. (Cited on page 176.)

[145] X. Yang. NIRA: A New Internet Routing Architecture. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 301–312, Karlsruhe, Germany, Aug. 2003. (Cited on page 176.)

[146] T. Ye and S. Kalyanaraman. A Recursive Random Search Algorithm for Large-Scale Network Parameter Configuration. In *Proc. ACM SIGMETRICS*, pages 196–205, San Diego, CA, June 2003. (Cited on pages 28 and 50.)