ABSTRACT

Foundations of Inter-Domain Routing

Vijay Ramachandran

2005

Inter-domain routing protocols establish best-effort connectivity between the independently administered networks that form the Internet. Because of the scale, heterogeneity, and autonomy of the Internet, inter-domain routes are computed using complex routing policies provided locally at each network with little global coordination. The interaction of these local policies has been shown to produce global routing anomalies, *e.g.*, protocol oscillations and nondeterministic routing. Understanding the interaction of local policies is essential in improving Internet stability. Unfortunately, the current inter-domain routing protocol for the Internet, the Border Gateway Protocol (BGP), allows wide latitude in configuring local policies and has evolved without formal guarantees regarding its behavior.

This dissertation develops a theoretical framework for the design and analysis of path-vector protocols, like BGP, used for inter-domain routing. It presents the path-vector policy system (PVPS), which is an abstract representation of the fundamental components of an inter-domain routing protocol. The framework includes an explicit notion of policy languages and global assumptions made about the network; it is shown that, while local constraints on policies alone can prevent routing anomalies, implementation of these constraints infringes on other desirable protocol properties. Thus, any design guaranteeing a reasonable combination of protocol properties requires a nontrivial global constraint on the network. Applications of these design principles are explored, including several constraint-enforcement mechanisms. Unlike many previous models of inter-domain routing, the PVPS framework can be extended to include the complexities of combining inter-domain and intra-domain routing and to include policies that cannot be described as a linear preference ordering on paths.

Foundations of Inter-Domain Routing

A Dissertation
Presented to the Faculty of the Graduate School
of
Yale University
in Candidacy for the Degree of
Doctor of Philosophy

by
Vijay Ramachandran

Dissertation Director: Joan Feigenbaum

December 2005

For My Parents

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Joan Feigenbaum. I enrolled in her first class at Yale, a seminar on e-commerce research, and after giving a presentation on traitor tracing that she must have liked, she asked me about my research interests. For this I am forever grateful, because I apparently said the right things, and Joan magically became my advisor. At the time, I didn't realize exactly what that meant, but now I can wholeheartedly say that, in any capacity, no student could ask for a better mentor.

Joan's support for my research and career development has been truly generous. She encouraged me, early on, to attend conferences and meet other researchers, but even here she went miles beyond what I expected: I remember her taking the time to personally introduce me to many of her colleagues. She has carefully read almost anything I have produced and has given thoughtful feedback, whether it be conference papers or graduate-school paperwork. She imparted other useful job skills: she taught me how to referee journal submissions; she involved me in writing and assembling several grant proposals; and she encouraged my participation in funding-agency workshops and reviews. I know few others who had such a valuable opportunity to be so completely involved in the academic research process as a student.

She approached my teaching fellowship in much the same way. Two weeks before class began, she and I met to plan the class syllabus and to choose readings together. I was involved in preparing almost all the sets of lecture slides and class notes, and she valued my input and skills while doing so. She and I together designed homework and exam questions. And, among my favorite moments as a graduate student are the times she let me give lectures—some of which I wholly prepared on my own.

But Joan's guidance goes far beyond her exemplary role as a research and teaching advisor. She has gone out of her way to help with my post-graduation plans, and is always willing to discuss my concerns. She has given me personal support through frustrating times, be they difficult health or conference rejections. And she has set an example for me by inspiring research enthusiasm and collaboration.

I doubt that I can ever say enough to thank Joan for all that she has done for me, but I will always be truly indebted to her for her care and wisdom.

Next, I want to thank Aaron Jaggard, my colleague or partner-in-crime, depending on the situation. Aaron has worked with me through all of the papers that constitute this dissertation, and none of the results would be possible without him. I will always remember his stack of blank notecards rubber-banded together, always at reach, so that even while gulping down New Haven pizza, we're not far from recording a potentially good idea. Through many long days (and some nights) in front of a blackboard or our respective computers, we've managed to cobble together what I think is a nice set of results. I thank him for fixing all the proofs that went wrong, for writing all the parts I couldn't, and for ignoring all the stupid things that come out of my mouth. (I also thank him for forcing me to be more rigorous than I would have been otherwise; he is a mathematician, after all.) He deserves equal credit for anything in this dissertation. I could never ask for a more devoted collaborator, and I hope our collaboration will continue.

I am also deeply grateful to Tim Griffin, my other collaborator. He is a reservoir of routing knowledge and, as our resident expert, has helped Aaron and me learn everything we know about BGP. Tim is responsible for taking our first results and expanding it into a SIGCOMM paper; that was what finally gave me a dissertation topic. And through the rest of the research, he was always willing to review our ideas, he encouraged us to develop them, and he challenged us to make them better. Tim has been an inspiration—he is someone who knows how to really enjoy his work and convince others that it's fun, too. He reminds me that one is never too old to be a child, and that one is never too young to tackle the next great problem.

There are many other people without whom this dissertation would not be possible. I want to thank the faculty and my fellow graduate students in the Computer Science Department at Yale for pitching in their thoughts, attending my talks, and always giving me good feedback and support. I thank Jonathan Edwards College, which gave me a home and a community for three years, and the Yale Glee Club, which kept me singing happily, even when work was difficult. My friends and family deserve my sincere appreciation for everything they have done, especially: my late grandfather, to whom I attribute any mathematical skills I have; my grandmother, whose love will always keep me smiling; and my parents, to whom I owe everything.

Financial support for my research came from several sources that I would like to explicitly acknowledge here. During my tenure as a graduate student, I received a 2001–2004 National Defense Science and Engineering Graduate (NDSEG) Fellowship from the U.S. Department of Defense (DoD) and a 2000–2001 CTInfoTech Scholarship from the State of Connecticut. In my last year of graduate study, I was partially supported by National Science Foundation (NSF) Grant ITR–0219018. My travel for research meetings and conferences was supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research (ONR) through ONR Grant N00014–01–1–0795.

In fact, my first meetings with Tim and Aaron and the initial results inspiring this dissertation are a result of telephone conversations among the PIs of this URI grant, during which my task (as the helpful graduate student) was to keep written notes. I somehow showed enough interest in one of the problems we had been discussing so that Joan and Andre Scedrov, a professor at the University of Pennsylvania, suggested that Aaron and I work on it off-line. We were then lucky enough to discover that our simple hypothesis, which later became one of the foundational theorems in this dissertation, was true. What happened next is in the following 180 pages.

Vijay Ramachandran
New Haven, CT, USA
May 2005

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The goal of this dissertation is the development of a rigorous theoretical framework for the design and analysis of path-vector protocols, which are primarily used for inter-domain routing on the Internet.

Packets sent across the Internet are directed by routers along a path from source to destination. Some edges on this path occur within a single autonomous system (AS)—a network managed under one administrative domain—while other edges occur between ASes. In the case of *intra-domain routing* (routing within an AS), *link-state protocols* are used to communicate the network information required to compute routes. These protocols flood the domain with status information so that routers can learn the network topology's current state and locally calculate the optimal paths for packets. Because the routers in a domain are managed by the same administrative authority, they presumably have a common routing goal; thus, given the same network-status information, they can compute routes both consistently and independently. In addition, domains are generally small enough that flooding status information through the network does not generate an unreasonable amount of network traffic. For these reasons, link-state protocols are efficient for intra-domain routing.

On the other hand, routing at the inter-domain level (between ASes) involves many independent entities across great distances. Flooding network-status information is neither practical, because of the size of the Internet, nor desirable, because AS administrators are often reluctant to share internal status information and, even if they did, are often choosing routes based on different criteria. Therefore, a different type of protocol, a *path-vector protocol*, is primarily used for inter-domain routing. In these protocols, the route to a given destination is determined by a composition of decisions made by routers along potential paths to that destination. Routers iteratively compute routes by first gathering information from local destinations and destinations advertised by neighboring routers, then computing a best route to each destination and sharing these routes with neighboring routers (possibly causing updates to neighbors' best routes) based on individual AS policies. Thus, in contrast to link-state-protocol routing, path-vector-protocol routing is highly dependent on the various independent policies throughout the Internet.

The standard inter-domain routing protocol for the Internet is the Border Gateway Protocol (BGP). The BGP specification [RL95] merely describes the low-level binary formats of update messages sent between neighboring routers, the intended meaning of the fields included in update messages, and the correct behavior of a BGP-speaking router in response to those messages. Configuration of local policies, upon which the computation of routes is so dependent, is left to individual router operators and their ability to program the router hardware. Vendor-developed routing-policy languages have evolved through interactions with network engineers to express the full set of semantically rich policies established by ASes, but this language development has taken place in an environment lacking vendor-independent standards. Router vendors typically provide hundreds of special commands for use in the configuration of BGP. For example, RFC-1997 communities [CTL96] allow policy writers to selectively attach tags to routes and use these to signal policy information to other BGP-speaking routers. These routers can, in turn, use the presence or absence of specific community values to determine the behavior of their policies. This allows network operators to encode complex policies in order to address unforeseen situations and has opened the door for a great deal of creativity and experimentation in routing policies.

This rich expressiveness has come at a cost: The interaction of locally defined routing policies can lead to unexpected global routing anomalies, such as nondeterministic routing and protocol divergence [GSW02, VGE00]. If the policies causing such anomalies are defined in autonomously administered networks, then these problems can be very difficult to debug and correct. For example, the setting of an attribute in one AS to implement "cold-potato routing" can cause protocol divergence in a neighboring AS [CIS01, MGWR02]. Such problems will probably become more common as BGP continues to evolve with richer policy expressiveness. For example, extended communities [STR05] provide an even more flexible means of signaling information within and between ASes than RFC-1997 communities do. At the same time, applications of communities by network operators are evolving to address complex issues of inter-domain traffic engineering [BQ03].

A better understanding of the effect of policy interactions on protocol behavior is essential for network stability. Ideally, we want to characterize a protocol and policies that are *robust*, *i.e.*, those that converge to a predictable set of consistent routes, even after link and node failures. Initial work inspired by specific problems with BGP has motivated the results in this dissertation, which better describe the behavior of path-vector protocols without involving the details of particular network configurations or protocol-specific implementations. As an early body of results in the emerging field of Internet algorithmics, the formal model and Internet algorithms developed in this dissertation are used to prove, *e.g.*, that certain constraints guarantee good behavior, even in worst-case scenarios. This rigorous treatment of protocol behavior is unfortunately not yet a common part of existing protocol-adoption standards; however, the results here, while abstract, suggest design principles that can be applied to improve current routing protocols and to design new ones.

1.2 SUMMARY OF RESULTS

Joint work with Timothy G. Griffin and Aaron D. Jaggard [GJR03] developed the foundation for the results in this dissertation. Presented in Chapter 4, the Path-Vector Policy System (PVPS) framework is an abstract representation of the components of a routing system: the underlying message-passing system for route information (the *protocol*), the languages for and constraints on routing-policy configuration (*policy languages*), and any global conditions on the network assumed to be true for correct protocol behavior (the *global constraint*). Separating and rigorously modeling the components allows us to:

1. identify different parts of the design process during which to include constraints to guarantee good behavior;
2. rigorously define desirable protocol properties; and
3. prove inherent trade-offs in the protocol-design space.

Two of the most important protocol-design results we prove are:

1. A local condition (involving pairs of policy configurations) is enough to guarantee that path-vector protocols converge robustly; and
2. In most cases, a reasonable combination of protocol-design goals, including robust convergence, *requires* a nontrivial global constraint. (Implementing the above local condition, therefore, infringes on some other desirable properties.)

Gao and Rexford [GR01] were the first to present a combination of local and global constraints that guarantee robust BGP convergence. Their solution takes into account basic Internet economics, because they assume that a hierarchical structure based on business relationships underlies the AS-level network graph. Using the PVPS framework, we can show that their constraints represent a point solution in a space of systems designed with similar assumptions about node relationships; we call these systems *class-based systems* and study them in Chapter 6.The results were originally presented in joint work with Aaron D. Jaggard [JR04]. Class-based systems can capture routing configurations as complex as BGP or arbitrary policies based on the next-hop AS of paths. From a simple description of three types of rules regarding policies — (1) consistency in the types of relationships allowed between neighboring ASes, (2) preference assigned to routes learned from different types of neighbors, and (3) restrictions on sharing routes with different types of neighbors — we can derive a global constraint that guarantees robustness. This constraint is the best known, because networks violating the constraint can legally write policies that result in protocol oscillation. We describe two constraint-enforcement algorithms:

1. a centralized algorithm that, given a network and routing-policy configurations involving neighbor relationships, can detect a set of nodes whose relationships permit policies that induce oscillation; and

2. a distributed algorithm that detects if a relationship between two specific nodes permits policies that induce oscillation, given the other nodes' policies in the network. This algorithm maintains the privacy of nodes' policies and the identity of the nodes whose relationship is being tested.

Class-based systems are a good application of the PVPS framework, because the restriction of policy languages to those involving neighbor relationships is both narrow enough to require more tractable global constraints for robustness than in the general case and broad enough to capture a spectrum of protocols and permitted routing-policy configurations. Rather than providing just one solution for robustness, the results in Chapter 6 give a protocol designer the ability to choose a balance between local-policy expressiveness and global-constraint strength.

Before we present the class-based application of PVPSes, we first relate the framework to *path-vector algebras* [SOB03], which is a similar framework for modeling path-vector protocols, in Chapter 5. We provide a translation between PVPSes and algebras and use it to show that the convergence results are essentially equivalent. These results were originally published as joint work with Aaron D. Jaggard [JR05A].

We show, through examples and formal relationships, that protocol specifications and routing configurations can be described at three distinct levels of abstraction:

1. Individual networks and routing-policy configurations can be represented by instances of the Stable-Paths Problem (SPP) [GSW02].

2. A protocol specification can be captured by a path-vector policy system, instances of which are specific networks running the protocol.

3. A set of protocols with common route-selection criteria can be described by an algebra. Protocols consistent with an algebra then correspond to PVPSes, which model more implementation details than the criteria used to describe the algebra.

A benefit of the algebra is that it provides an even more abstract description of protocols than a PVPS; the behavior of select parts of protocols' route-selection procedures can be studied apart from their specific implementations.

Our translation between frameworks adds important concepts to each that were not originally discussed in the respective original papers [GJR03, SOB03]:

1. We add the notion of expressiveness to algebras, which allows a comparison of protocol specifications in the two frameworks based on the types of permitted routing configurations.

2. We add the notion of optimal convergence to PVPSes and derive analogous conditions at the PVPS level of abstraction that guarantee optimality.

We are also able to use the translation to discuss the basic constraints for robust convergence originally presented for both frameworks.

The derivation of all of the above results (and much of the related work on modeling BGP) makes two assumptions about inter-domain routing that do not entirely hold in practice:

1. The procedure for choosing a best route can be described by assigning a rank (in a totally ordered set) to each route and choosing the route with minimal (or maximal) rank; and

2. The network is the AS-level graph; *i.e.*, each node represents one AS, each node has a single routing policy that describes how all inter-domain routes within the AS are established, and there is at most one edge between an AS and each of its neighbors.

Assumption 1 is called *independent route ranking*, because the rank of a path depends only on the route itself. However, use of the Multi-Exit-Discriminator (MED) attribute in BGP, which instructs ASes to perform "cold-potato routing," violates this assumption. Assumption 2 ignores *iBGP* sessions, which are intra-domain BGP sessions maintained to share inter-domain routes between routers within an AS; most work modeling BGP only captures *eBGP*, which are the inter-domain BGP sessions between routers in different ASes.

The complexity of iBGP introduces additional types of routing anomalies that cannot be analyzed with the basic PVPS or algebra frameworks [GW02A, GW02B].

In Chapter 7, we extend the PVPS framework by removing these assumptions, presenting results that have previously appeared in joint work with Aaron D. Jaggard [JR05B]. We allow each node to specify as its route-selection procedure an arbitrary choice function on sets of routes. Because this removes any notion of rank, convergence conditions from earlier chapters no longer apply. However, we take first steps towards designing constraints in this setting. We take as a semantic domain the *Generalized Stable-Paths Problem* [GW02A], which does allow for arbitrary route-selection functions, and incorporate it as a new measure of expressiveness. We then define two new relations on policies; using these, we expand the notion of *evaluation digraphs*, *dispute wheels*, and *dispute digraphs*—structures used to analyze convergence in the original work on SPP [GSW02]—and use them to state conditions for robust convergence in the generalized case. We give examples that show how the extended framework can be used to model the interaction between iBGP and eBGP sessions. We also evaluate several proposals to eliminate oscillations induced by MED-attribute settings.

This dissertation begins with two chapters of background material. In Chapter 2, we review the concepts behind Internet forwarding and route-signaling in order to motivate our careful study of inter-domain routing. In Chapter 3, we discuss related work studying policy interactions and their effects on BGP convergence. In doing so, we present some important definitions and results that are incorporated into the development of the PVPS framework.

CHAPTER 2

BACKGROUND: INTERNET ROUTING

The Internet is made up of many heterogeneous networks. Hosts can be connected using various physical media, each with its own method of carrying messages. Networks can communicate with each other as long as the hosts and physical media connecting them share a common interface to the *Internet Protocol (IP)*, which is a standardized method of identifying hosts and formatting messages. Any host speaking IP that is connected to some physical network medium can send IP traffic to any other IP destination reachable from that network medium. This universal compatibility of IP with different hosts and network types is what gives the Internet its power and scale.

The most widely deployed version of IP is currently version 4 (IPv4) [POS81A]. It assigns hosts unique 32-bit identifiers, known as *IP addresses*, and provides a format for *packets*, each of which contains an *IP header* with information used by IP to deliver messages and a *payload* containing the message being sent. IP guarantees best-effort message delivery, *i.e.*, there are no quality-of-service guarantees, including confirmation of delivery. Traffic is normally broken up into multiple packets, because payload size is limited for compatibility and efficiency reasons; no effort is made to guarantee that packets travel synchronously along similar paths from source to destination. Transport-layer protocols that run on top of IP at the end hosts, *e.g.*, TCP [POS81B] and UDP [POS80], provide *multiplex-*

ing so that traffic from multiple applications on the same hosts are identified as separate flows. TCP also provides some of the additional functionality mentioned above, including ordering packets, controlling congestion, and providing a mechanism to confirm delivery. Internet applications can then be designed to run on top of one of these protocols; the delivery mechanism from end-to-end is assumed to be taken care of by IP.

Because Internet networks are arbitrarily connected, there are often multiple paths between any source-destination pair. Many of these paths traverse several networks (ASes), and thus IP traffic must be directed intelligently. Although an individual network has various means to direct traffic between hosts directly connected to it, devices are needed to connect networks to each other and direct traffic between them. These devices are called *routers*; in order to communicate with different physical-network types, routers speak IP and may have multiple interfaces to different physical networks. Routers perform two fundamental tasks referred to as two distinct *planes*: the *forwarding plane*, in which IP traffic is directed to the next point on a path to its destination, and the *signaling plane*, in which information about Internet destinations is communicated between routers in order to establish paths. Because nodes can join or leave the Internet arbitrarily, and individual networks have control over their topologies, there is no centralized control mechanism for routing information. Routing is thus handled by distributed protocols, and routers are only able to reach Internet destinations as information about potential paths becomes available through routing-protocol messages.

Figure 2.1 shows an example of routers connecting various network types (subnetworks) within a large corporate network and routers connecting networks together (and to the rest of the Internet). Different physical network types are used to connect hosts (shown using circles). Hosts on the same physical network can communicate directly, *e.g.*

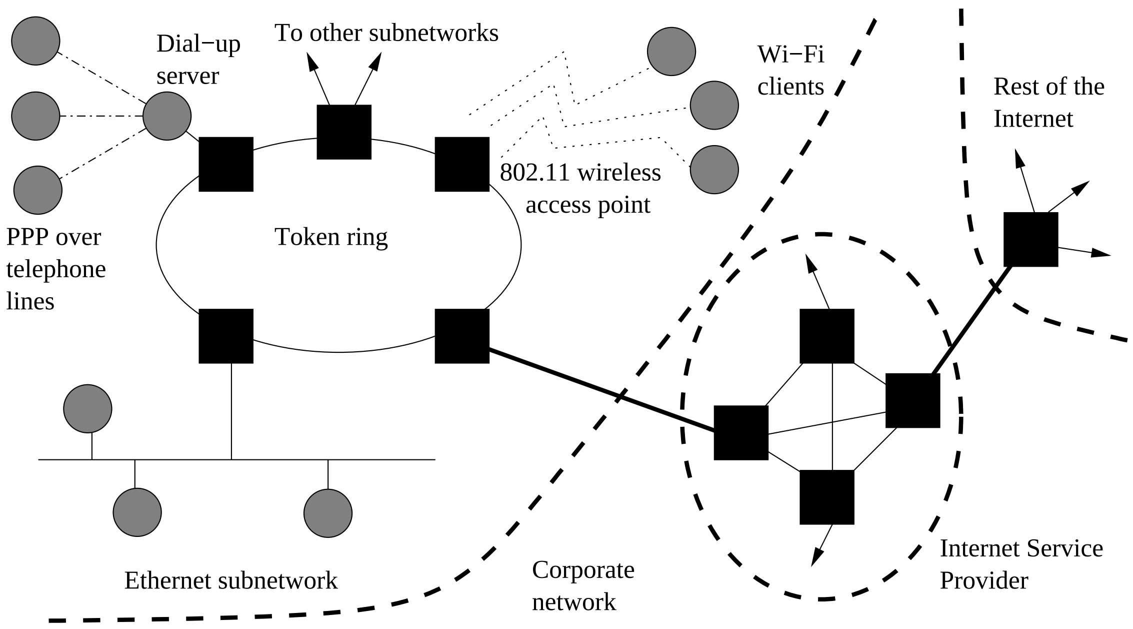


Figure 2.1: Internetworking.

hosts on the *ethernet*, a broadcast network, communicate by broadcasting IP packets on the physical medium so that the intended recipient can recognize the destination address and obtain the messages. For other destinations, routers (shown using square boxes) forward packets through an interface that they believe will eventually reach the destination; packets to destinations outside the network eventually reach the network's Internet Service Provider (ISP) and are routed through the rest of the Internet (thick lines show connections between routers in different networks).

2.1 IP FORWARDING

The basic forwarding operation needs to be very fast in order to accommodate the high volume and desired speed of Internet communication; thus, it consists of a simple table lookup.

IP routers maintain a *forwarding table*; it can be thought of as a simple match between two columns of data, IP-address prefixes and output interfaces. When an IP packet is received, its header is examined for the destination IP address. The *longest-prefix match* for the destination is then located in the forwarding table; this is the entry with the most bits of the IP-address prefix (starting with most significant) in common with the destination IP address. The packet is then forwarded along the physical-network interface in the forwarding table corresponding to that match; this should be the physical link that carries the packet closer to its destination.

Longest-prefix matching is natural, because blocks of consecutive IP addresses, called *Classless Inter-domain Routing (CIDR) blocks*, are often delegated to a network to use for addressing; consecutive addresses are often used for hosts in the same subnetwork, and a hierarchy of networks corresponds to a hierarchy in addressing where deeper levels share more less-significant bits of an IP address. Therefore, destination IP addresses can often be aggregated into CIDR blocks, which can be represented by an IP-address prefix in the forwarding table. For example, the prefix 12.45.68.* matches all IP addresses in the range 12.45.68.0–12.45.68.255; the prefix 12.45.* matches 12.45.0.0–12.45.255.255. If both prefixes are present in a forwarding table, the more specific entry for the destination is used, which corresponds to the longest-prefix match.

The job of routing protocols, then, is to populate the entries of this forwarding table. The task is usually split up among two different types of protocols, depending on the sets of destinations to which paths are being established.[1] The networks that constitute the Internet can be divided into *domains*, each of which contains networks that fall under the same administrative authority; different issues of scale, autonomy, and privacy apply to routing within a domain and routing between domains. Therefore, *Interior Gateway Pro-*

[1]In theory, a router can simultaneously run any number of routing protocols, but the task of coordinating their results is nontrivial. As we discuss in Section 3.4, the interaction of the two most commonly used protocols does not overwrite entries in the forwarding table but does create some routing anomalies.

tocols (IGPs) are used for intra-domain routing, but *Exterior Gateway Protocols (EGPs)* are used for inter-domain routing. In the next two sections, we review the concepts behind these; both are important for our ultimate task of analyzing inter-domain routing because inter-domain routes are often dependent on intra-domain connectivity and configuration.

2.2 IGPS AND INTRA-DOMAIN ROUTING

The goal of an IGP is to establish a set of consistent, best routes at every router to each intra-domain destination. By *consistent*, we mean that every subpath of a chosen best route is also a chosen best route. (Inconsistency can lead to IP-forwarding loops.) Because domains operate under the same administrative authority, the following assumptions usually apply to running an IGP:

1. Domains are usually small enough that status information about network links can be flooded throughout, giving each router the ability to compute the network topology.
2. Domains usually have the same notion of "best," which is often shortest (fewest number of hops) or lowest-cost (assuming that edges can be assigned a transit cost, *e.g.*, corresponding to congestion or distance).

For these reasons, most common IGPs, *e.g.*, OSPF, are *link-state protocols*. In these protocols, routers send out link-state packets containing the status of links to neighboring routers; as these are received, routers can update their view of the network topology, which can be represented as a weighted graph. A simple lowest-cost-paths calculation, *e.g.*, using Dijkstra's algorithm, will establish a set of consistent routes to each intra-domain destination; as these are computed, the *next hop* (or neighboring router) on the path to the destination is used to populate the forwarding-table entry for that destination. Changes to the network topology trigger new link-state packets, which in turn trigger re-computations of the network topology and best routes.

In Figure 2.1, intra-domain links are shown using thin lines connecting routers and hosts. Each subnetwork has a corresponding router that is responsible for that subnetwork's connectivity to the rest of the domain; this router is often called the *default gateway* for the subnetwork. Within the corporate network or within the ISP, an IGP would be used to calculate best paths between routers for traffic with destinations inside the respective domains. It is important to note that the scope of link-state information only covers intra-domain routers and thus intra-domain destinations; even though traffic destined for hosts outside the domain may traverse intra-domain links, the IGP does not carry information about these destinations.

2.3 EGPS AND INTER-DOMAIN ROUTING

Routing at the inter-domain level is more complex because of the scale of the Internet graph and the autonomy desired by domain administrators in configuring routes—in the inter-domain case, we cannot assume that all routers have the same definition of "best" route. Once a packet is forwarded to another domain, that domain has total control over it and its future path. This is unlike intra-domain routing, in which paths are wholly contained in networks under the same administrative authority, and the behavior of neighboring routers is known. In the IP-forwarding model, destination reachability always depends on the paths chosen by neighboring routers; but, in the case of inter-domain routing, *all* the information received about destinations comes from neighboring routers and their choices of routes. The Internet is simply too big, and there are too many routes; so, sharing more than potentially consistent routes through immediate neighbors is inefficient.

Therefore, EGPs are usually *path-vector protocols*, the name of which comes from the mechanism used to prevent loops: In the information shared about potential routes, the entire path (called the *path vector*) is included; nodes can then check whether they already appear in the path vector and, if so, discard that route option. Paths are maintained at the *autonomous-system (AS)*—or domain—level. This is because the EGP assumes that destination and transit domains can appropriately route traffic within the domain using the IGP; therefore, EGP messages do not contain information about paths used within a domain. This allows a domain to preserve its autonomy over internal routes and keep its network topology private from others. Therefore, an abstract model of an EGP network is a graph where each node, an AS, represents one domain, and each edge represents an inter-domain connection.

Each router maintains a *routing table* with AS paths to various IP prefixes; the actual network next-hop associated with a path is used to populate the forwarding-table entry for the IP prefix. (IP forwarding does not occur at the AS level.) When more than one route to a destination is available, the choice of best route depends on the router's *local-policy configuration*; the types of policies depend on the specific protocol and hardware and are often quite expressive, constrained only by the configuration language provided by router vendors. Local policies influence the setting of *attribute values*—information about a route stored in its routing-table entry—when routes are learned and shared with neighbors; the attribute values are directly involved in the computation of best routes.

The EGP route-selection procedure is depicted in Figure 2.2. Knowledge about destinations is learned through *advertisements* from neighboring routers; once a path to another AS is established, an AS will share that *reachability information* with its neighbors so that they gain knowledge of the destination as well. Assuming that initial paths are first *originated* by

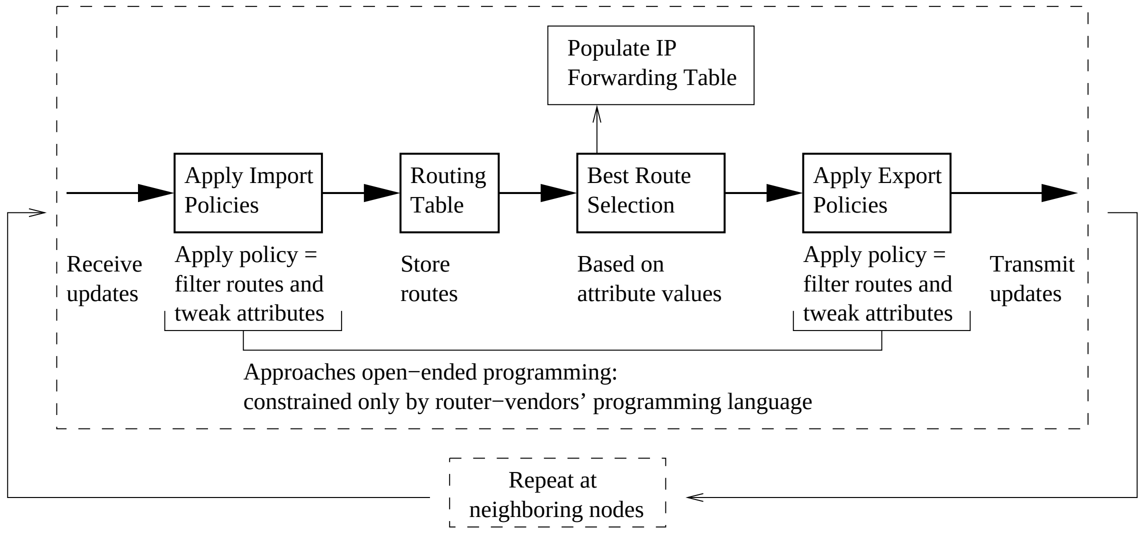

Populate IP Forwarding Table
Apply Import Policies
Routing Table
Best Route Selection
Apply Export Policies
Receive updates
Apply policy = filter routes and tweak attributes
Store routes
Based on attribute values
Apply policy = filter routes and tweak attributes
Transmit updates
Approaches open−ended programming: constrained only by router−vendors' programming language
Repeat at neighboring nodes

Figure 2.2: Dynamics of EGP best-route selection.

the EGP routers responsible for the corresponding destinations, paths are established by repeating the following three-step process:

IMPORT Information about established routes through neighboring routers is collected, called *importing* routes. The route data stored in the local routing table depends on the route information in the update message and the *import policy*; the policy may *filter* routes entirely, *i.e.*, remove them from consideration.

SELECTION For each destination, the protocol's best-route selection procedure is used to choose best routes from the local routing table. Best routes are then used to populate the forwarding table for these destinations.

EXPORT Best routes are advertised to neighboring routers through update messages, a process called *exporting*. Update-message information about these routes is influenced by *export policy*, which may also filter routes.

The routers exchanging this information with inter-AS connections are *border routers*; however, to establish connectivity for all hosts in an AS, non-border routers must learn how to reach external destinations as well. Because its update messages can carry inter-domain route information, the inter-domain protocol is also used to share external destinations with internal routers. As a result, EGPs accomplish two inter-domain routing tasks:

1. establishing connectivity and sharing reachability information across inter-domain links; and

2. distributing knowledge of inter-domain routes to non-border routers.

Therefore, even though the abstract model of an EGP network hides intra-domain paths at the AS level, the intra-domain structure of an AS is a component of establishing inter-domain routes in that AS. (It is interesting to note that inter-domain routing protocols are themselves built on top of IP; therefore, in order for the EGP to function, it is expected that intra-domain IP forwarding can take place based on computations of the IGP. Forwarding across inter-domain links to permit EGP sessions is usually accomplished by hard-coding entries into routers' forwarding tables.)

2.4 THE BORDER GATEWAY PROTOCOL

The standard Internet EGP is the Border Gateway Protocol (BGP) [RL95]. A BGP session is maintained between neighboring routers: *eBGP sessions* are used between different ASes' border routers; *iBGP sessions* are used between routers in a single AS. Originally, BGP required all routers in an AS to maintain iBGP sessions with each other, called a *full mesh*—this ensured that all inter-domain routes would be available to each router. For purposes of scalability, BGP has been modified to allow a smaller full mesh of select routers, called *route reflectors*, each of which maintains iBGP sessions with a set of *clients* that do not learn all inter-domain routes, but only those best routes chosen by the route reflectors [BCC00].

The BGP specification documents the low-level binary formats of messages used to communicate route information and the intended meaning of message fields; the specification does not explicitly constrain routing policies used to set attribute values on route import and export. Several attributes play a role in the best-route selection procedure, which we now describe. Some attributes are not relevant when we consider only eBGP and assume that the network is simply the AS-level graph.

When a route is imported, it is given a *local-preference value* based on import policy; this is the main criterion for determining the preference of a route. The *Multi-Exit Discriminator (MED)* attribute is set by the *exporting* AS to indicate *its* preference among multiple inter-AS connections. Path length and (for iBGP) IGP distance to the border router are also attributes involved in best-route computation, but these are not set by local-policy configuration. The full best-path selection procedure for BGP is as follows:

1. Routes with the largest local preference are chosen as best.

2. In the case of a tie, routes with the shortest AS-path length are chosen.

3. In the case of a tie, if there are multiple paths through the same AS, then *for each next-hop AS*, choose the path with the lowest MED value. MED values are only compared among paths through the same AS; therefore, this step may leave multiple paths for consideration (but at most one path will remain through any given next-hop AS).

4. If there remains a tie because there are paths through multiple ASes, choose routes learned from eBGP sessions over those learned from iBGP sessions.

5. If multiple paths remain, choose the path with the shortest IGP distance to its *egress point*, or first border router.

The importing AS has ultimate authority by setting local-preference values, but these are often set to the same value for all routes through the same next-hop AS, even across different inter-AS links. (This practice is consistent with having AS-level policies.) In practice, this allows a neighboring AS to influence the decision between multiple inter-AS links using the MED attribute.

One typical example of MED usage is *cold-potato routing*. Assuming MEDs are not used (*i.e.*, ignoring step 3), the route-selection procedure above (via step 4 and 5) breaks ties based on closest egress point (minimal IGP distance). This is known as *hot-potato routing*. Depending on the destination prefix, a neighboring AS may specify alternate preferences for ingress points using the MED attribute, *e.g.*, to avoid using expensive intra-domain links. Consider two inter-AS connections: one in San Francisco and one in New York. A small customer network may have high costs sending traffic across its internal links. When advertising destinations to its Internet provider, the customer can attach appropriate MED values to the destinations so that the provider chooses the egress point that closest to the destination, minimizing the use of the customer's internal links. If the provider instead used basic hot-potato routing, the closest egress point in the provider network would be chosen, possibly causing the customer to handle transcontinental traffic.

Most previous work modeling path-vector-protocol behavior (reviewed in Chapter 3) and the results we present in Chapters 4–6 apply to networks at the AS level, modeling only eBGP and ignoring iBGP. In Chapter 7, our model is generalized to capture network configurations describing both eBGP and iBGP sessions and use of the MED attribute.

CHAPTER 3

REVIEW OF LITERATURE: MODELING BGP*

Having seen the complexity of BGP's route-selection procedure, it is clear that understanding its behavior, which is determined in part by the independently configured routing policies in autonomous networks, is difficult. As discussed in Chapter 1, policy-induced global routing anomalies have been documented in BGP. In this chapter, we review the development of existing theoretical models of BGP and path-vector protocols. They were motivated by a desire to rigorously analyze the cause of routing anomalies. In doing so, we provide simple characterizations of the problems with inter-domain routing that we attempt to remedy in later chapters.

The results in this dissertation are strongly motivated by the previous work discussed in this chapter; many of the results are directly incorporated into the larger protocol-design framework developed by this dissertation. Of particular importance are the definitions and theorems in Section 3.2, which were originally presented in [GSW02].

*Portions of the text in this chapter have previously appeared in joint works with Aaron D. Jaggard [JR05B, JR05C].

3.1 PROBLEMS WITH EBGP

As network nodes write their routing policies and share data using BGP, the interaction among these policies can have undesirable effects. These can be generally classified into two types of routing anomalies: *persistent route oscillation*, in which the protocol never settles on a stable set of routes; and *nondeterministic routing*, in which the protocol may eventually settle on a stable set of routes, but convergence is not guaranteed, and the set of routes is not predictable. Below, we trace the characterization of these anomalies in previous work.

3.1.1 PERSISTENT ROUTE OSCILLATION

Varadhan, Govindan, and Estrin [VGE00] were first to outline a basic view of routing dynamics and used this to characterize the timing-independent oscillations that could occur in simple classes of network topologies. An example of this is shown in Figure 3.1. We call this BAD GADGET, adopting the name of a similar example given in [GSW02]. It consists of a small network in which nodes 1, 2, and 3 try to select routes to node 0; every pair of nodes in this network is connected by a link. Next to each node in Figure 3.1 are the *permitted paths* that each node will consider, listed in order of preference: node 1 prefers the path through node 2 to node 0 over the path directly from node 1 to node 0; these paths are denoted 120 and 10, respectively. We assume that node 1 does not learn about other routes because of routing policies (*e.g.*, node 3 might not share route information with node 1, and paths containing loops are filtered out). Similarly, nodes 2 and 3 prefer routes through nodes 3 and 1, respectively, over their direct routes to node 0. Assume that these nodes also do not learn other routes.

If no links fail, the direct paths to node 0 are always known. Suppose that these direct paths are chosen at nodes 1–3. Following BGP dynamics (Section 2.4), these choices are

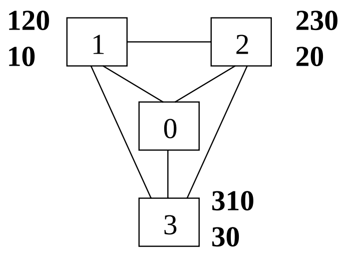


Figure 3.1: The routing configuration (SPP instance) BAD GADGET.

advertised to neighbors, making available the more preferred, indirect paths. Once the indirect paths are chosen, the direct paths are no longer advertised; withdrawal of these routes makes the indirect paths unavailable, and all nodes choose the direct paths again. This process repeats *ad infinitum*, never converging to a choice of routes.

As shown in [VGE00], this oscillation of path selections does not depend on timing in the network. Note that, if any one of the outer nodes' policies were changed to prefer the direct path to node 0, this oscillation would not occur, and the nodes would have a stable set of consistent routes.

3.1.2 NONDETERMINISTIC ROUTING

We call a stable set of consistent route choices a *solution*. In a solution, every node is assigned a path such that (1) all paths are valid extensions of neighbors' paths, and (2) no node is able to choose a more preferred path given its neighbors' choices. (This resembles a Nash-equilibrium condition.) A solution may or may not be unique in a given network. The routing configuration shown in Figure 3.2, originally given in [GSW02] and called DISAGREE, has two solutions: (1) 10 and 210; and (2) 20 and 120. Either set of routes remains stable because no node can learn of a more preferred route. Unfortunately, the protocol does not have to converge to either of these solutions. In an oscillation similar to that of BAD GADGET, if both nodes 1 and 2 begin by choosing the direct routes 10 and

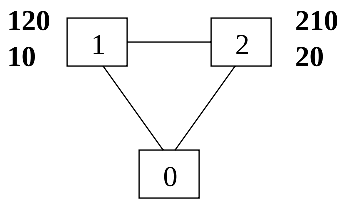


Figure 3.2: The routing configuration (SPP instance) DISAGREE.

20 and advertising those to the other, the protocol can oscillate indefinitely, although this depends upon the timing of various router operations (unlike the persistent oscillation of BAD GADGET).

Routing configurations with multiple solutions are not *predictable*: Delays in sending BGP update messages or different orderings of link failures and recoveries can result in different choices of routes for the same set of routing policies. In these cases, routing might appear nondeterministic to network operators, making problem diagnosis difficult.

3.2 THE STABLE-PATHS PROBLEM

Griffin, Shepherd, and Wilfong [GSW02] gave a more detailed formal model for networks running path-vector protocols and proposed the *Stable Paths Problem (SPP)* as the underlying problem that BGP solves. An *instance* of SPP contains essentially the information given in Figures 3.1–3.2 above: a graph corresponding to the network and a set of permitted paths at each node, each of which is assigned a *positive rank* by the node at which it is permitted corresponding to that node's level of preference for the path. (The rank of a path must be determined independently; when this is not the case, as with BGP's MED attribute, the modeling and analysis of protocols becomes much more difficult; see Section 3.4 below.) Which paths are permitted and path-rank values result from interactions between routing policies across the network being modeled; thus, an SPP essentially captures the static semantics of a network's routing-policy configuration. A solution to an instance corresponds to a stable set of consistent routes, as we described earlier. Formally, we have the following definitions.

DEFINITION 3.2.1. The quadruple

$$S = (G,\ v_0,\ \mathcal{P},\ \Lambda)$$

is an *instance of the Stable Paths Problem (SPP)* if $G = (V,\ E)$ is a finite undirected graph, $v_0 \in V$ (called the *origin*), $\mathcal{P}$ is a set of simple paths in G terminating at v_0, and the mapping Λ takes nodes $v \in V$ to a path ranking function $\lambda^v = \Lambda(v)$. Each λ^v is a function that takes a path in $\mathcal{P}^v = \{P \in \mathcal{P} \mid P \text{ is a path starting at } v\}$ to its *rank* in $\mathbb{N}$. If $W \subseteq \mathcal{P}^v$, then the subset of "best paths" in W, $\min(\lambda^v,\ W) \subseteq W$, is defined as the set

$$\{P \in W \mid \text{for every } P' \in W,\ \lambda^v(P) \leq \lambda^v(P')\}.$$

DEFINITION 3.2.2. A *path assignment* for an SPP-instance S is any mapping π from V to subsets of $\mathcal{P}$ such that $\pi(v) \subseteq \mathcal{P}^v$. The set $\mathrm{candidates}(u, \pi)$ consists of all permitted paths at u that can be formed by extending the paths assigned to neighbors of u. For $u = v_0$, $\mathrm{candidates}(u, \pi) = \{(u)\}$, and for $u \neq v_0$,

$$\mathrm{candidates}(u, \pi) = \{uQ \in \mathcal{P}^u \mid \{v,\ u\} \in E \text{ and } Q = \pi(v)\}.$$

A path assignment π is a *solution* for an SPP if, for every node u, we have

$$\pi(u) = \min(\lambda^u,\ \mathrm{candidates}(u, \pi)).$$

That is, if F is a functional that takes path assignments π to path assignments $F(\pi)$, defined as $F(\pi)(u) = \min(\lambda^u,\ \mathrm{candidates}(u, \pi))$, then the solutions of the SPP are exactly the fixed points of F (for any solution π we have $F(\pi) = \pi$, and $F(\pi) = \pi$ implies π is a solution).

A convenient abbreviation for the best path at u under π is defined to be

$$\text{best}(u, \pi) = \min(\lambda^u,\ \text{candidates}(u, \pi)).$$

Then π is a solution if $\pi(u) = \text{best}(u, \pi)$ at each node u.

REMARK 3.2.3. The definition for SPP given here is a bit more general than that of [GSW02] in that we do not require "strictness," which guarantees that $|\pi(v)| \leq 1$ for every solution π. In addition, we have changed the order of the ranking to prefer paths with smaller (not larger) rank. Finally, we have allowed any node $v_0 \in V$ to be the origin.

This rigorous definition of the routing problem led to several insights in [GSW02]. First, it was shown that solving the routing problem—*i.e.*, determining whether or not a routing configuration has at least one solution—is *NP*-complete. Checking a specific set of routing policies for a stable route assignment is thus believed to be impractical. Varadhan *et al.* [VGE00] suggested that shortest-path routing—in which each node prefers the route with the fewest hops—might be the only provably safe routing in arbitrary network topologies. (Note that the instances BAD GADGET and DISAGREE are inconsistent with shortest-paths routing.) Griffin *et al.* [GSW02] did show that this suggestion works: in general topologies, if routers choose paths with the fewest hops, then a solution is reached. However, using their formalism, they could also prove much broader sufficient conditions on network policies that guarantee good behavior.

3.2.1 EVALUATION DIGRAPHS AND CONVERGENCE PROPERTIES

We are not only interested in whether policies interact such that there is a stable path assignment, *i.e.*, whether or not an SPP has a solution, but also in how EGPs can reach that assignment. Given an SPP, there is a naturally corresponding structure that describes the execution of path-vector protocols on the SPP instance, called the *evaluation digraph*.

DEFINITION 3.2.4. The *evaluation digraph* of an SPP instance S is a directed graph $\mathcal{T}(S) = (V_{\mathcal{T}}, E_{\mathcal{T}})$, in which the nodes represent *protocol selection states*, and the edges represent transitions between states. A selection state is a path assignment $\pi \in \left(\prod_{v \in V} \mathcal{P}^v\right)$; if $\alpha \in V_{\mathcal{T}}$, then we write the path assignment corresponding to this node as π_α. The *start state* is the node corresponding to the empty path assignment, in which $\pi(v_0) = (v_0)$, and, for $u \neq v_0$, $\pi(u) = \epsilon$, the empty path. The directed edge (α, β) is present in $E_{\mathcal{T}}$ iff for all $u \neq v_0 \in V$, $\pi_\beta(u) = \mathrm{best}(u, \pi_\alpha)$, *i.e.*, given that nodes select the paths π_α and then broadcast these selections to their neighbors through asynchronous FIFO links, nodes might next select the paths π_β. Note that there may already be path data in the links that have been delayed in transit, so that $\pi_\alpha(v) = P$ and $\pi_\beta(v) = P'$ but, for a neighbor u, $\pi_\alpha(u) = Q$ and $\pi_\beta(u) = uP$. (Therefore, states may not be consistent; these states are not acceptable as solutions.)

We can follow the execution of a path-vector protocol on an SPP instance by its *trace*, which corresponds to a directed path in the evaluation digraph beginning at the start state. Traces can either end at *sink states*, *i.e.*, nodes whose only outgoing edges are loop edges, or cycle through states indefinitely. Because the evaluation digraph is finite, if all traces are acyclic (ignoring loop edges), then all protocol runs will converge. Conversely, it is clear that if the network can dynamically oscillate during route selection, then there is a cycle in the corresponding evaluation digraph; each of the paths among which a node oscillates will appear in at least one of the states in the corresponding cycle.

PROPOSITION 3.2.5. *A path assignment corresponds to a sink state iff it is a solution.*

Proof. A solution is a stable set of consistent routes. Suppose π_α is a solution; then by Definition 3.2.2, for all $u \neq v_0 \in V$, $\pi_\alpha(u) = \mathrm{best}(u, \pi_\alpha)$. By Definition 3.2.4, this is equivalent to α having no outgoing edges in the evaluation digraph other than loop edges, meaning that α is a sink state. □

Therefore, we can define protocol-convergence properties in terms of the structure of the corresponding evaluation digraph. The following combinations of the existence of solutions and the ability of protocols to reach those solutions are of interest to us.

DEFINITION 3.2.6. The following are convergence properties of SPP instances.

SOLVABILITY An SPP is *solvable* if there exists at least one path assignment that is a solution; *i.e.*, the evaluation digraph of the SPP has at least one sink state.

UNIQUE SOLVABILITY (PREDICTABILITY) A routing configuration is *uniquely solvable* if there exists exactly one SPP path assignment that is a solution; *i.e.*, the evaluation digraph contains exactly one sink state.

SAFETY A routing configuration is *safe* if a path-vector protocol is able to converge to a solution; *i.e.*, all traces in the SPP's evaluation digraph are acyclic. The existence of a solution does not determine safety.

ROBUSTNESS A routing configuration is *robust* if it and all sub-instances (resulting from node or link failures) are uniquely solvable and safe; *i.e.*, all traces in the SPP evaluation digraph are acyclic and end at the same sink state.

REMARK 3.2.7. Note that the definition of robustness, while requiring all sub-instances to be predictable and safe, requires only that all traces in the *original* SPP's evaluation digraph are acyclic and end at the same sink. This is because sub-instances have evaluation digraphs that are subgraphs of the original instance's evaluation digraph (with some paths no longer possible because of failures); the property of acyclicity holds on subgraphs.

We are interested in robust path-vector protocols, because these avoid nondeterminism and divergence, which are problems that are difficult for network operators to understand and debug when they occur at the inter-domain level.

3.2.2 DISPUTE WHEELS AND ROBUST CONVERGENCE

We begin broadening conditions for convergence by generalizing shortest-paths routing. In the first logical step, let each network link be assigned an arbitrary positive cost; the cost of a path is then the sum of the costs of its component edges. Lowest-cost routing (choosing the cheapest path) is thus analogous to shortest-paths routing and, indeed, always converges to a routing solution. Griffin *et al.* [GSW02] directly proved an even more general condition, showing the sufficiency of *coherent cost assignments* for robustness. In a coherent assignment, edges may have negative costs, so long as every cycle in the graph has a positive cost (*i.e.*, the sum of edge costs around any cycle is positive). Intuitively, the potential problem with negative edge costs is that a path can traverse a cycle of negative-cost edges multiple times, artificially reducing total path cost; however, preventing non-positive cycles makes this impossible. Coherence also precludes the divergent examples from above. However, the authors of [GSW02] gave an example of a convergent routing configuration in which policies are not consistent with coherent costs, suggesting that one might write a broader sufficient condition.

Proving the coherence result involved characterizing a necessary condition for divergence. Using SPP's abstract model of a network configuration, Griffin *et al.* showed that a generalization of BAD GADGET, called a *dispute wheel*, captures this condition. In particular, they described a procedure that attempts to construct a routing solution given an SPP; an unsuccessful attempt implied the existence of a dispute wheel in the SPP. They also showed that multiple routing solutions implied the existence of a dispute wheel. Thus, an

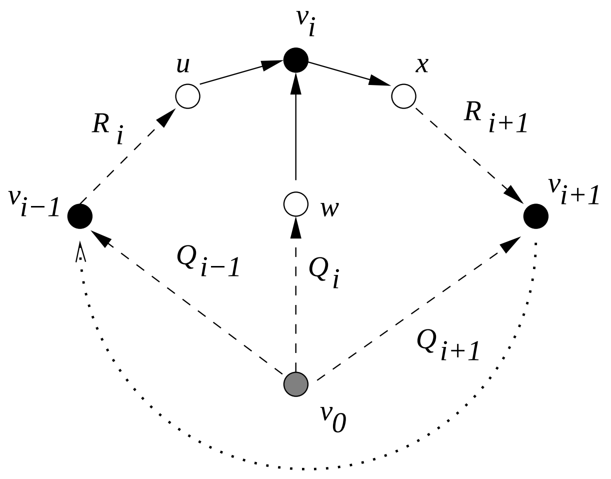


Figure 3.3: Dispute wheel.

SPP without a dispute wheel is robust; this observation was central to the coherence result and motivates the results in Chapters 4–7. We provide a formal statement of these results below.

A generic dispute wheel is shown in Figure 3.3. It comprises a *rim* and *spokes*, which are paths in the network graph such that routing policies at the *active nodes*—where spokes connect to the rim—conflict, allowing bad routing behavior. (Nodes and edges may appear multiple times in a single wheel.) In particular, each of these nodes v_i learns a path Q_i to the destination from its neighbor w down the spoke but would prefer to use the path along R_i that follows the rim clockwise through u to v_{i-1} and then goes down the next spoke Q_{i-1}. (It is easy to see that a three-node version of this network configuration is BAD GADGET.)

DEFINITION 3.2.8. A *dispute wheel* (see Figure 3.3) is a cycle of *active nodes*

$$v_1, v_2, \ldots, v_k, v_{k+1} = v_1$$

in an SPP instance such that there exist paths

$$R_1, R_2, \ldots, R_k, R_{k+1} = R_1 \text{ and } Q_1, Q_2, \ldots, Q_k, Q_{k+1} = Q_1$$

such that $Q_i \in \mathcal{P}^{v_i}$, $R_{i+1}Q_{i+1} \in \mathcal{P}^{v_i}$, and $\lambda^{v_i}(R_{i+1}Q_{i+1}) < \lambda^{v_i}(Q_i)$. The nodes and paths R_i are on the *rim* of the dispute wheel, while the paths Q_i are called the *spokes* of the wheel.

Suppose all active nodes start by only knowing (and thus selecting) spoke paths. As time progresses, extensions of these routes may be further propagated through the network so that at each active node, a path counterclockwise through the rim to the next active node and then down that node's spoke becomes available. Because these routes are preferred, they will all be selected. Once this happens, active nodes can no longer advertise their spoke paths because they are not selected, and the spoke paths will be withdrawn. This eventually makes the extended paths through the rim unavailable, reverting all choices back to the direct spoke paths, at which point this sequence can start again.

While such an oscillation is not guaranteed to occur if a network contains a dispute wheel—there may be other paths preferred over all of the paths in the wheel—a dispute-wheel-free network cannot produce this type of oscillation.

THEOREM 3.2.9 (Section V, [GSW02]). *If an SPP instance contains no dispute wheel, then it is robust.*

Proof. In [GSW02], Theorem V.3 states that a dispute-wheel-free S has a solution, Theorem V.4 states that it has a unique solution, Theorem V.9 guarantees that a path-vector protocol will converge to a solution for S, and Theorem V.10 guarantees that a unique solution can be found in the presence of link and node failures. □

The condition of dispute-wheel-freeness is not a *local* condition but instead a restriction on how the local routing decisions of nodes may interact *globally*. This global condition forms the basis for many of the results in Chapters 4–7.

3.3 HIERARCHICAL BGP

Gao and Rexford [GR01] were the first to discuss the role of local constraints defined in terms other than cost increments assigned to links. They showed that an assumption about the Internet AS-graph structure and a combination of simple rules for nodes' policies are enough to guarantee BGP's convergence. Fortunately, these rules and assumptions are consistent with, and are naturally enforced by, common Internet economics.

Two connected ASes usually view their relationship either as between a customer and a provider of network connectivity or as between two equals; in the second case, these "peers" may use their connection to provide backup connectivity, to connect their customers, or to short-cut expensive or longer routes through provider links. In this "Hierarchical BGP" (HBGP) model, every AS assigns one of the labels "customer," "provider," or "peer" to each of its neighbors such that this view is consistent with that of other ASes (*e.g.*, an AS's customers view it as a provider).

HBGP then requires that nodes' routing policies satisfy certain restrictions, defined in terms of these labels, on the relative ranking of routes and with whom routes are shared; these restrictions are natural given the traffic agreements usually made with the three types of neighbors. (For example, routes learned from customers must be preferred to routes learned from providers, and the latter are shared only with customers, not peers or other providers.) Finally, it is assumed that no "customer/provider" cycles exist, *i.e.*, no AS is an indirect customer of itself. This last restriction is also natural in that it is very unlikely that a local ISP would sell network connectivity to a top-level network.

The initial results for HBGP convergence were proven directly. Gao, Griffin, and Rexford [GGR01] later generalized this work by adding back-up routes to HBGP, and proved their convergence results by using machinery from [GSW02]: They showed that networks satisfying generalizations of the conditions from [GR01] avoid dispute wheels and are thus robust.

Although this work provides one solution that offers a combination of local and global constraints that guarantee robust convergence of a protocol, our results in Chapter 6 generalize these constraints by characterizing sets of protocols and constraints that do so.

The major result of [GGR01, GR01] is that, because the required global condition is naturally enforced, stable Internet routes can be achieved by modifying local policies without real global coordination. This balance between local and global constraints underlies the main approach of this dissertation. In Chapter 4, we show that local constraints can guarantee robust convergence with no global assumption at all, but that other desirable protocol properties are lost when these constraints are implemented.

3.4 MED-INDUCED AND IBGP ANOMALIES

The modeling work described so far only applies to a specific type of path-vector protocol—those in which the best-route selection procedure can be modeled by mapping paths under consideration to a rank, or weight, in some totally ordered set and choosing the path of minimum (or maximum) rank. This property is called *independent route ranking* (IRR) because the rank of a path can be determined from the attributes of that path's data structure, which, in turn, can be used to compare it to any other path for best-route selection (assumption 1, Section 2.2). However, BGP's full route-selection procedure (Section 2.4) cannot be modeled in this way. In particular, use of the multi-exit discriminator (MED) attribute, which is common when two ASes share multiple inter-connections and want to perform cold-potato routing, violates IRR.

MED-induced oscillations are a well-known problem of BGP [CIS01, DS98, MGWR02], and it has been conjectured that the violation of IRR is the major reason. These oscillations are especially difficult to analyze and debug on a real network because they are a product of not only BGP policy settings—involving attributes set in separately configured, independent ASes—but also internal distance settings within an AS (determined by an IGP; see Section 2.2).

In addition to this complexity of iBGP, the use of route reflectors (see Section 2.4) introduces additional anomalies that mirror the oscillation and nondeterminism examples for eBGP. These were rigorously characterized in [GW02B], and numerous examples of iBGP anomalies not involving the use of MEDs were given. These fall into two major categories:

1. *signaling anomalies*, in which configuration of iBGP sessions prevents convergence to a stable, consistent set of inter-domain routes; and

2. *forwarding anomalies*, in which a conflict between IGP-determined forwarding and iBGP-assigned egress points, while undetected in the signaling plane, causes traffic to be forwarded in loops or to be *deflected*—taking an unintended intra-domain path to an egress point.

We note that much of the related work we discuss below on the MED-oscillation problem also refers to iBGP anomalies and misconfiguration in general. In this dissertation, we focus on MED-induced anomalies because modeling MEDs (and, more generally, any route-selection procedure violating IRR) requires an interesting extension to existing work that we present in Chapter 7. In addition, the principles derived in Chapters 4–6, when combined with the model in Chapter 7, can be used to model and study iBGP anomalies in general.

There has been some theoretical work on the consequences of using the MED attribute, but the results have not been as complete as those derived from modeling eBGP alone. Basu *et al.* [BOR$^+$02] and Musunuri and Cobb [MC04] proved that including in advertisements routes not chosen as best prevents MED-induced oscillations, but this change to BGP would increase the size of routing tables and the number or size of update messages. (We note that the solution proposed in [MC04], as opposed to that of [BOR$^+$02], uses changes to BGP update messages between route reflectors and their clients to avert iBGP forwarding anomalies, in addition to iBGP and MED-induced signaling anomalies.) In Section 7.3.1, we suggest an improvement that requires fewer additional routes to be broadcast.

Griffin and Wilfong [GW02A] enumerated canonical examples of MED-induced oscillations and described them using an extension to their SPP model, called the *Generalized Stable-Paths Problem (GSPP)*. Instances of GSPP contain nodes with arbitrary choice functions on sets of routes that describe their route-selection procedures, rather than an IRR-compliant, linear preference ordering on permitted paths as in the original SPP. This work, however, did not propose broad configuration suggestions for using the MED attribute nor a robustness constraint analogous to that given for the original SPP model. It was shown that in the particular case of MEDs, GSPP instances could be translated to an SPP instance containing proxy nodes. These instances could be tested for policy-induced anomalies using the SPP conditions described in [GSW02], but those conditions do not easily translate to conditions on GSPP instances. In Chapter 7, we incorporate GSPP into our framework to accommodate IRR-violating route-selection procedures; however, unlike [GW02A], we are able to provide analogous convergence constraints. We defer the definition of GSPP to Chapter 7 in order to avoid complicating the derivation of results in Chapters 4–6, which only rely on the original version of SPP.

Other suggestions to solve the MED-oscillation problem affect the use of route reflectors and configuration of iBGP sessions within an AS [WCRS02] or require changing the interpretation of attributes [MGWR02]. In Section 7.3, we use our framework to prove that some of these conjectured solutions do indeed prevent MED-induced oscillations.

CHAPTER 4

PATH-VECTOR POLICY SYSTEMS*

In this chapter, we introduce the Path-Vector Policy System (PVPS) framework. The motivation for developing this framework is that a root cause of configuration problems is a lack of design for the policy languages that are used to configure protocols. BGP policy languages have evolved in a rather organic fashion with little or no effort made to avoid policy-interaction problems. We believe that researchers should start to consider how to *design* policy languages and path-vector protocols that together avoid such risks and yet retain other desirable features. We take a few steps in this direction by identifying the important dimensions of this design space and characterizing some of the inherent design trade-offs. We do this in a general way that is not constrained by the details of BGP. As a result, our framework may offer guidance not only in the analysis of proposals to correct or extend BGP but also in the analysis of other BGP-like protocols such as a version of BGP supporting Virtual Private Networks [RR99], Telephony Routing over IP (TRIP) [RSS02], and of various proposals for interdomain routing of optical paths [RLA04, XBX03].

*This chapter has previously appeared in joint work with Timothy G. Griffin and Aaron D. Jaggard [GJR03].

4.1 THE DESIGN SPACE OF PATH-VECTOR PROTOCOLS

We identify six important design goals for any path-vector protocol and policy language:

EXPRESSIVENESS From the perspective of a network operator, we desire policy languages that are as *expressive* as possible. For example, shortest-path routing is not expressive enough for the requirements of current interdomain routing because it is unable to capture the "natural" routing conditions arising from the pervasive economic roles of customer, provider, and peer [HUS99A, HUS99B]. The challenge then is to design policy languages that are as expressive as possible, and yet not so expressive that other design goals are sacrificed.

ROBUSTNESS We require predictability, *i.e.*, that any non-determinism in routing policies is not the result of unwanted policy interactions, and the existence of a routing solution that is always found by the protocol (this prevents protocol divergence). Furthermore, we insist that the same is true of any configuration that results from any combination of link and node failures in the network. The goal of robustness is the primary constraint on the expressive power of a policy language; we are generally uninterested in non-robust policies.

AUTONOMY Network operators often require a high degree of *autonomy* when defining routing policies. We may have a good intuition about what this means—that policy writers are given wide latitude in defining policies that reflect their own interests and not the interests of their neighbors. Here, generalized autonomy will mean the ability to define a partition on routes and then rank the partition classes arbitrarily. Operationally, autonomy is important because it isolates an autonomous system from policy changes occurring in other (neighboring or distant) autonomous systems. Without a high degree of autonomy, network operators would have to continually "tweak" their policies to compensate for unseen changes made to policies elsewhere.

In addition to a generalized definition, we present one notion of autonomy important for BGP—*autonomy of neighbor ranking*—that allows policy writers to classify neighbors and set route preferences in accordance with this classification. This type of autonomy is required for a BGP policy language to support policies compatible with present-day commercial realities of the Internet.

PROTOCOL TRANSPARENCY Many "obvious" approaches to achieving very expressive and robust systems involve a high cost; they add machinery that is invisible to policy writers to the underlying path-vector system. What is lost is *protocol transparency*—the ability of network operators to understand the semantics of policies they write. If the protocol itself is allowed to dynamically modify the input policies (in order to ensure robustness, for example), then it may become very difficult, if not impossible, to maintain and debug routing policies.

GLOBAL CONSISTENCY One way to achieve robustness is to implement a mechanism enforcing a global-consistency constraint that guarantees robustness. This constraint could be enforced in any number of ways, including an additional protocol or set of protocols, by convention, by regulation, by economic incentives, or by some combination of methods. Of course, the easier such a constraint is to check, the better. We note that in the current Internet, there is no global-consistency checking of BGP policies.

POLICY OPAQUENESS This design goal measures the degree to which details of routing policies are to be kept private or hidden from those outside of a routing domain (the term is from Geoff Huston [HUS01]). Full policy opaqueness is, of course, in direct conflict with any sort of global-consistency enforcement. Therefore, the design challenge is to find a happy medium that allows for the exposure of just enough information to ensure robustness while at the same time allowing for a sufficient amount of information hiding to satisfy policy writers.

Our formalization starts with defining three distinct components of any path-vector protocol: the underlying path-vector system, the policy language, and any global consistency assumptions about the network. The path-vector system should be thought of as the low-level means of carrying messages between systems, much like RFC 1771 [RL95]. Section 4.2 presents a definition for path-vector systems that formalizes the information that nodes exchange, various restrictions on nodes' behavior, and the way that protocols mediate interactions between nodes. As we define various components, we illustrate them with a running example that models BGP. Additional examples are given in Section 4.3.

We separate the definition of a path-vector system from the definition of a policy language: a policy language is a high-level declaration of how the attributes describing a route change when the route is exchanged between neighbors. Section 4.2.2 defines the intended role of policy languages in path-vector-system configuration.

The notions of expressiveness and robustness are formalized in Sections 4.4 and 4.5. For both we employ the Stable Paths Problem (SPP) [GSW02] as a semantic model of path-vector systems. We identify one class of robust systems as our target for expressiveness (Definition 4.5.4 and Theorem 4.5.9). Autonomy and transparency are formalized in Sections 4.6.1 and 4.6.2. Policy opaqueness is briefly discussed in Section 4.6.4, while global constraints are considered in Section 4.7.

Besides the more obvious trade-offs already mentioned, we identify several more subtle ones:

1. Any system with a policy language that is maximally expressive but has no global constraint must give up either autonomy of neighbor ranking or transparency (or both) (Theorem 4.6.9).

2. Any autonomous, transparent, and robust system with a policy language at least as expressive as shortest-path routing must have a non-trivial global constraint (Theorem 4.7.4).

These results tell us that, if we seek to design expressive policy languages that are transparent, autonomous, and robust, then we must consider the global constraint as an integral part of the design. Indeed, current path-vector protocols may succeed in part because of assumptions about the global network; our framework highlights the importance of this component of design.

Figure 4.1 illustrates the design space for robust and transparent path-vector policy systems. (This figure is meant to aid in developing intuitions, and should not be taken too literally.) The x-axis represents the expressive power of systems, and the y-axis represents the relative difficulty of checking the global constraint. Combinations of path-vector systems and policy languages which fall close to the bottom right of Figure 4.1 are generally desirable.

Some points in the space deserve attention. On the bottom horizontal line lie systems that require no global constraint to be robust. In this paper, we assume "minimal" expressiveness is "Shortest-Paths" routing; a simple extension to this is "Shortest-Available Paths," which allows routes to be filtered (even if they are the shortest) and chooses the shortest path from the remaining routes. (Both examples are given in Section 4.3.) We take "maximal" expressiveness to be the expressive power of a natural class of robust systems that we define in Section 4.5.3. Two possible systems which possess the property "Globally

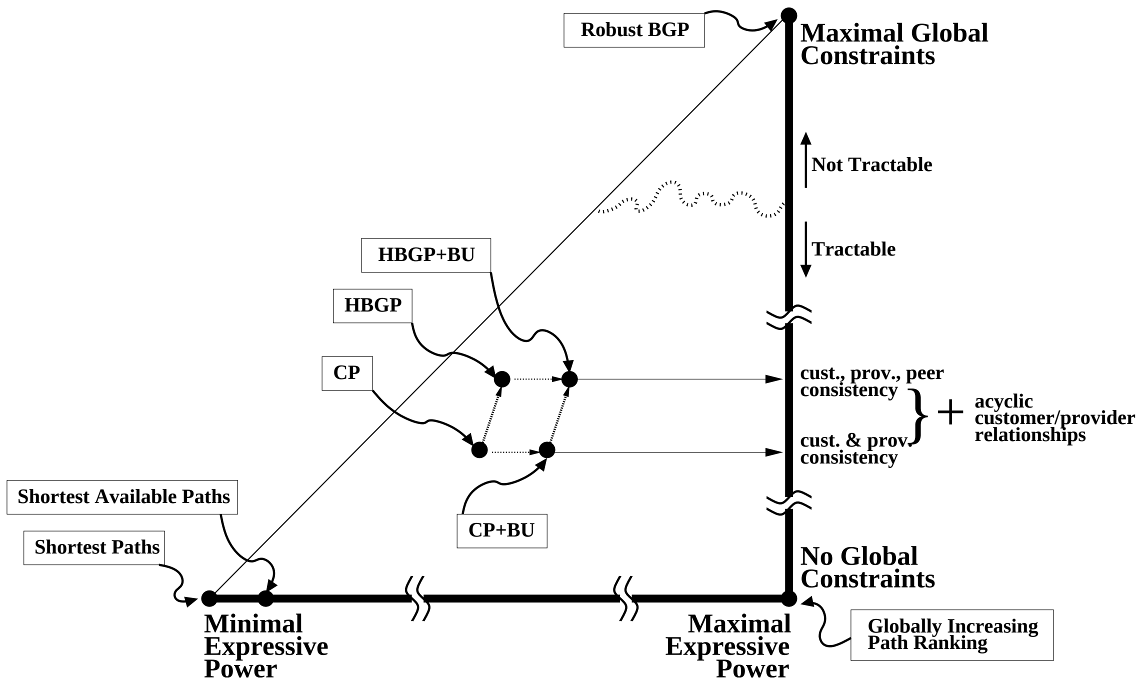


Figure 4.1: Design space for robust path-vector systems.

Increasing Path Ranking" are discussed in Section 4.6.3; while these achieve maximal expressiveness with no global constraint, they sacrifice other design goals in the process. The final extreme point, "Robust BGP," is a system in which all BGP policies are collected and verified not to contain conflicting policies. One might use the Routing Policy Specification Language (RPSL) [ABG$^+$98] in the manner suggested in [GAE$^+$99] to accomplish this. Many practical issues make this scenario unlikely; furthermore, it was shown in [GSW02] that, in the worst case, checking various global-consistency constraints is *NP-hard*.

Hierarchical BGP systems (inspired by [GR01, GGR01]) provide examples from today's commercial Internet. Figure 4.1 includes the system CP, a BGP-like system in which the policy language allows nodes to classify neighbors as customers and providers and to rank routes consistent with those relationships; CP is robust if there are no cycles in the customer/provider graph and if classifications of neighbors are consistent. We might increase the expressiveness of this system in two ways: (1) allow an additional classification of neighbors as peers, in which case we must modify the global constraint to additionally check the consistency of peer classifications (the system HBGP); or (2) modify the policy language to permit marking routes for backup use (the system CP+BU). Combining both approaches achieves the expressiveness of the system HBGP+BU. These types of systems are discussed fully in Chapter 6. Note that in the real world, there are no existing methods to enforce either the local or global constraints, although Internet economics seems to ensure that networks behave in close approximation to the rules described by the above-mentioned robustness conditions.

4.2 DEFINITION OF PATH-VECTOR POLICY SYSTEMS

In this section, we define the "protocol part" of our framework: the underlying exchange system for route information. We sketch the components independent of any particular system or instance of a system. Using the definitions presented here, we can rigorously explore the protocol design space in later sections.

4.2.1 FORMAL DEFINITION OF PATH-VECTOR SYSTEMS

As we develop our framework, we will use a simplified model of BGP as a running example. This example model assumes that each node (router) represents an entire autonomous system and thus treats only External BGP (not Internal BGP). It also ignores most BGP attributes and simplifies others. We will adorn the elements of this example system with the subscript μbgp.

ROUTE INFORMATION

A path descriptor is a data record about a path that contains enough information (*e.g.*, the routing destination, the sequence of AS numbers along the entire path, routers' preference values for the path, transmission cost, *etc.*) for a router to compare it to other paths and to inform its neighbors about the path so that they can do the same. A router learns of paths by receiving descriptors from neighbors and preserves knowledge of potential best routes by storing descriptors for paths to all known destinations.

The path-vector-system specification includes a description of the components in a path descriptor and a map that ranks them using values from a totally ordered set. This ranking permits routers to determine best routes based on just the information contained in the available descriptors to a destination; in particular, the rank of a descriptor depends only on that descriptor. Determining rank normally involves some components of path descriptors that can be transformed by both locally configured policies and the underlying message-exchange protocol itself.

DEFINITION 4.2.1. Let the quadruple

$$\mathcal{I} = (\mathcal{D},\ \mathcal{R},\ \mathcal{U},\ \omega)$$

be the *route-information* portion of the path-vector-system specification. The components are defined as follows:

$\mathcal{D}$ is the set of possible routing destinations;

$\mathcal{R}$ is the set of path descriptors, such that to every $r \in \mathcal{R}$ there must be associated a unique $dest(r) \in \mathcal{D}$;

$\mathcal{U}$ is a set totally ordered by $\leq$; and

ω is a function (the *ranking function*) from $\mathcal{R}$ to $\mathcal{U}$ that determines how path descriptors are ranked (thus, the role of path-descriptor attributes in choosing routes).

REMARK 4.2.2. Although the mechanics of determining "best" routes will be discussed in Section 4.2.5, we observe the convention that the ranking function will map more preferred paths to smaller elements of $\mathcal{U}$.

RUNNING EXAMPLE, PART 1. In our example system, let $\mathcal{D}$ be the set of all IPv4 CIDR blocks. Let the set of path descriptors be

$$\mathcal{R}_{\mu bgp} = \mathcal{D}_{\mu bgp} \times \mathbb{N} \times \mathrm{Seq}(\mathbb{N}) \times \mathbb{N} \times 2^{\mathcal{C}},$$

where $\mathbb{N}$ is the set of natural numbers, $\mathrm{Seq}(\mathbb{N})$ is the set of finite sequences of natural numbers, and $\mathcal{C}$ is the set $\{\mathrm{red},\ \mathrm{blue},\ \mathrm{green}\}$. If $r = (d,\ l,\ P,\ n,\ S) \in \mathcal{R}_{\mu bgp}$, then d is the destination of r, l is the *local preference*, P is the *AS path*, n is the *next hop*, and the elements of S are the *colors* of r. Colors are meant to be a very simple model of BGP communities [CTL96].

Let $\mathcal{U}_{\mu bgp} = \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ and $\omega((d,\ l,\ P,\ n,\ S)) = (l,\ |P|,\ n)$, with the ordering $\leq_{\mu bgp}$ on $\mathcal{U}_{\mu bgp}$ given by $(l,\ m,\ n) \leq_{\mu bgp} (l',\ m',\ n')$ if and only if:

1. $l > l'$; or
2. $l = l'$ and $m < m'$; or
3. $l = l'$, $m = m'$, and $n \leq n'$.

The combination of $\leq_{\mu bgp}$ and $\omega_{\mu bgp}$ prefers higher local preference, with ties broken by preferring smaller AS-path length and then smaller value of the next hop.

IMPORT AND EXPORT POLICIES

Path-vector systems explicitly include operations for importing routes from neighbors and exporting routes to neighbors. Router operators provide separate *import and export configuration policies* to describe router behavior when exchanging route information, *e.g.*, to change path-descriptor attributes for a route affecting its rank or to filter out routes altogether. The set of node policies across the network would therefore be a component of a specific instance of the path-vector system. On a low level, the import and export policies are per-neighbor functions on path descriptors that transform their components to make preference changes in accordance with local policy. We expect that policies will usually be written in a higher-level policy language, which motivates the policy-language component of design.

A path-vector system includes *local-policy constraints* on what import and export policies are allowed. These limits on the expressiveness of local policies can help guarantee robustness and can help ensure that a protocol achieves its goals; *e.g.*, if policies can only add a positive value to a path-cost attribute that alone determines path rank, the path-vector system implements lowest-cost-path routing.

Formally, let elements of the function space $2^{\mathcal{R}} \to 2^{\mathcal{R}}$ be called *policy functions* (these are functions on sets of path descriptors, thus describing transformations on them). We then define local-policy constraints in the following way.

DEFINITION 4.2.3. Let the triple

$$\mathcal{C} = (\textsc{l}^{in},\ \textsc{l}^{out},\ \textsc{o})$$

be the *local-constraints* portion of the path-vector-system specification. $\textsc{l}^{in}$ and $\textsc{l}^{out}$ are predicates on import and export policy functions, respectively. If $\textsc{l}^{in}(f)$ or $\textsc{l}^{out}(f)$ holds, then f is a *legal* local-policy function. Furthermore, we assume that if either $\textsc{l}^{in}(f)$ or $\textsc{l}^{out}(f)$ holds, then f satisfies:

(1) for each $X \subseteq \mathcal{R}$, if $|X| = 1$ then $|f(X)| \leq 1$;

(2) for each $X \subseteq \mathcal{R}$, $f(X) = \bigcup_{r \in X} f(\{r\})$; and

(3) for each $r_1,\ r_2 \in \mathcal{R}$, if $f(\{r_1\}) = \{r_2\}$, then $dest(r_1) = dest(r_2)$.

O is a predicate defined on subsets of $\mathcal{R}$ used to define what sets of path descriptors can be originated at a node. A node can only advertise newly originated destinations described by $X \subseteq \mathcal{R}$ if $\mathrm{O}(X)$ holds.

RUNNING EXAMPLE, PART 2. In our simplified-BGP example, we want policies to affect only the local-preference and colors (communities) attributes of path descriptors. We let $\mathrm{L}^{in}_{\mu bgp}(f)$ and $\mathrm{L}^{out}_{\mu bgp}(f)$ hold if and only if f satisfies conditions (1)–(3) above as well as

(4) $f((d,\ l,\ P,\ n, S)) = \{(d',\ l',\ P',\ n',\ S')\}$ implies $d' = d$, $P' = P$, and $n' = n$.

Additionally, the only path descriptors which may be originated by nodes are those with an AS path containing the AS alone (because the destination should be in the originating AS's domain) and a default local preference of 0, so we let $\mathrm{O}_{\mu bgp}(X)$ be true if and only if $(d,\ l,\ P,\ n,\ S) \in X$ implies $l = 0$ and $P = v$ where v is the originating AS.

APPLICATION OF POLICIES

Although import and export policies allow router operators to configure their routers, we must recognize that it is the router (or the protocol itself) actually applies those policies to path descriptors encountered while running the protocol. Therefore, path-vector-system specifications include a *policy-application function* for both the import and export operations. These functions describe the transformations used by the protocol to apply operator-provided policies to path descriptors. This allows the application of policies to be consistent with the goals of the protocol, *e.g.*, routers may only apply policies when they satisfy a local condition guaranteeing robustness. These functions are often used to make changes to path descriptors uniformly throughout all information exchanges in addition to applying the operator-provided configuration policy (*e.g.*, appending a node name to the described path or hiding certain attributes when they contain private information). Formally, we have the following.

DEFINITION 4.2.4. Let the pair

$$\mathcal{T} = (t^{in},\ t^{out})$$

be the *protocol-transformation* portion of the path-vector-system specification. Both t^{in} and t^{out} are functions of type $(\mathbb{N} \times \mathbb{N} \times (2^{\mathcal{R}} \to 2^{\mathcal{R}}) \times 2^{\mathcal{R}}) \to 2^{\mathcal{R}}$; the first two arguments are node names, the third is the policy function to apply, the fourth is the target set of path descriptors.

RUNNING EXAMPLE, PART 3. We now give the protocol transformations for our model of BGP. If u and v are nodes, f is a policy function (expected to be u's export policy function for v), and X is a set of path descriptors (expected to be known to u), then

$$t^{out}_{\mu bgp}(u,\ v,\ f,\ X) = \{(d,\ 0,\ vP,\ u,\ S) \mid (d,\ m,\ P,\ w,\ S) \in f(X)\}\,.$$

The protocol applies the (export) policy function (which may change local preference and colors) and then updates the AS-path and next-hop values to reflect the edge $\{u,\ v\}$ in the extended path. It also sets the local preference value to 0, hiding this value from the node receiving information about this path. If Y is a set of path descriptors (expected to be $t^{out}_{\mu bgp}(u,\ v,\ f,\ X)$) and g is v's import policy function for u, then we let

$$t^{in}_{\mu bgp}(v,\ u,\ g,\ Y) = \{g(r) \mid r \in Y, r \text{ describes a simple path}\}\,.$$

The protocol thus takes care of filtering any paths which contain loops.

PATH-VECTOR SYSTEM

DEFINITION 4.2.5. A *path-vector system* is a triple of the form

$$PV = (\mathcal{I},\ \mathcal{C},\ \mathcal{T})$$

where the components are as defined in Definitions 4.2.1–4.2.4.

4.2.2 POLICY LANGUAGES

Of course, policy writers do not actually write mathematical functions, but rather write specifications in a *path-vector policy language*. We expect that such languages can be given a rigorous semantics so that policies written in the language can be treated as specifications for functions on path descriptors. A policy language essentially is a local constraint on the policy functions that can be written for a path-vector system. Policy-language designers must ensure that legal policy specifications are guaranteed to have semantics that conform to the constraints of the target path-vector system(s). In practice, this may involve some type of *compilation* to low-level, vendor-specific configuration commands—a transformation that may be rather complex. However, separating the definition of a policy language from the definition of a path-vector system allows us to consider multiple policy languages for the same path-vector system. We can also discuss using different path-vector systems to implement the same policy language.

DEFINITION 4.2.6. A *policy language PL* for a path-vector system is a language and a *semantic function $\mathcal{M}$* that maps each *policy configuration p* written in this language to a triple

$$\mathcal{M}(p) = (m^{in},\ m^{out},\ m^{orig})$$

of partial functions of types

$$\begin{aligned} m^{in},\ m^{out} &\ :\ V \times V \to (2^{\mathcal{R}} \to 2^{\mathcal{R}}) \\ m^{orig} &\ :\ V \to 2^{\mathcal{R}} \end{aligned}$$

If u and v are node identifiers, then $m^{in}(v,\ u)$ and $m^{in}(v,\ u)$ are called the *import* and *export policy functions at v for u*, respectively, and $\text{L}^{in}(m^{in}(v,\ u))$ and $\text{L}^{out}(m^{out}(v,\ u))$ hold whenever these policy functions are defined. These functions transform sets of path descriptors. Finally, the function m^{orig} maps node identifiers v to finite subsets of $\mathcal{R}$ such that $\textsc{o}(m^{orig}(v))$ holds whenever $m^{orig}(v)$ is defined.

We take policy configurations to be the language-specific definitions of policies for one or more nodes; the set of valid policy configurations is part of the language PL.

RUNNING EXAMPLE, PART 4. We define a simple policy language $PL_{\mu bgp}$. A policy configuration in this language is a list of *declarations*, each having one of the forms:

1. export from v to W : RULE
2. import at v from W : RULE
3. originate from v : $(d,\ 0,\ \epsilon,\ v,\ S)$

The first and second type declare export and import policies, respectively, and the third type declares routes to be originated from a node. The sets W represent all of the neighboring nodes to which a given declaration is applied. Each RULE is a transformation of objects in $\mathcal{R}_{\mu bgp}$ defined by a list of *clauses*:

$$\begin{array}{ccc} C_1 & \Longrightarrow & A_1 \\ C_2 & \Longrightarrow & A_2 \\ \vdots & \vdots & \vdots \\ C_n & \Longrightarrow & A_n \end{array}$$

where each C_i is a boolean predicate over path descriptors and each A_i is an *action* to be taken on the input path descriptor. The actions are either of the form REJECT, or they are statements that modify the local preference or colors of a path descriptor. For each path descriptor r input to such a rule, the action associated with the first predicate that evaluates to TRUE is performed on r. If no clause matches, the empty set is returned. $\mathcal{M}_{PL_{\mu bgp}}(p)$ is easy to determine given the form of policy configurations in $PL_{\mu bgp}$; see part 5 of the running example in the following subsection.

4.2.3 INSTANCES OF PATH-VECTOR SYSTEMS

DEFINITION 4.2.7. An *instance* of a path-vector system PV with respect to a policy language PL (or *an instance of* $(PV,\ PL)$) is a pair

$$I = (G,\ P),$$

where $G = (V,\ E)$ is an undirected graph, called the *signaling graph*, and the *configuration function* P maps nodes $v \in V$ to policy configurations $P(v) = p_v$ in the policy language PL so that $\mathcal{M}(p_v) = (F_v^{in},\ F_v^{out},\ F_v^{orig})$. We require that $F_v^{orig}(v)$ is defined and that, for every $\{v,\ u\} \in E$, both $F_v^{in}(v,\ u)$ and $F_v^{out}(v,\ u)$ are defined. We will assume that the vertex set V is a subset of $\mathbb{N}$.

Let $F(I) = (F^{in},\ F^{out},\ F^{orig})$ where

$$\begin{aligned} F^{in}(v,\ u) &= F_v^{in}(v,\ u) \\ F^{out}(v,\ u) &= F_v^{out}(v,\ u) \\ F^{orig}(v) &= \bigcup_{w \in V} F_w^{orig}(v) \end{aligned}$$

F is a summary configuration function for the instance that represents the collection of policy configurations provided by nodes in the instance. However, F technically describes transformations on path descriptors, and thus is a somewhat "compiled" or "lower-level" version of the policies for the instance, independent of the policy language used to specify them.

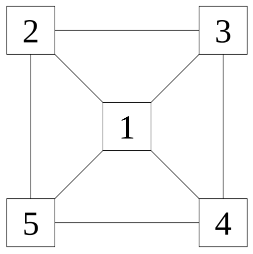


Figure 4.2: An example five-node network.

REMARK 4.2.8. In most cases, nodes will not originate descriptors on behalf of other nodes, *i.e.*, $F_w^{orig}(v) = \emptyset$ for $w \neq v$, and nodes will not have policies for non-incident edges, *i.e.*, $F_w^{in}(v,\ u)$, $F_w^{out}(v,\ u)$ are not defined for $w \neq v$. In addition, we suggest and often assume that the origination constraint includes a clause to check that nodes only originate path descriptors for destinations they represent or contain, *i.e.*,

$$\mathrm{O}(X) \Rightarrow \big[r \in X \Rightarrow \big(dest(r) = v \Rightarrow r \in F_v^{orig}(v)\big)\big]$$

DEFINITION 4.2.9. Given two instances $I = (G,\ P)$ and $I' = (G',\ P')$ of $(PV,\ PL)$, the instance I' is said to be a *sub-instance* of I if G' is a subgraph of G and the configuration function P' is equal to P when restricted to G'. For example, given any instance $I = (G,\ P)$ and G', a subgraph of G, the instance $I' = (G',\ P)$ is a sub-instance of I.

RUNNING EXAMPLE, PART 5. One instance of $(PV_{\mu bgp},\ PL_{\mu bgp})$ consists of the five-vertex graph shown in Figure 4.2 and policy configurations in Figure 4.3.

4.2.4 REALIZABLE PATH DESCRIPTORS

We are particularly interested in the path descriptors that arise as the result of first originating a path descriptor at some node and then forwarding it along some path in the network, applying the appropriate export, import, and protocol transform functions along the way. We call these *realizable path descriptors*. Because we do not usually make use of the path de-

```
originate from 1 : $(d,\ 0,\ (1),\ 1,\ \emptyset)$
export from 1 to $\{2\}$ :
  true $\Longrightarrow$ $r$.colors := $\{$red$\}$
export from 1 to $\{3,\ 4\}$ :
  true $\Longrightarrow$ $r$.colors := $\{$blue$\}$
export from 1 to 5 :
  true $\Longrightarrow$ $r$.colors := $\{$green$\}$

import at 2 from $\{1,\ 3,\ 5\}$ :
  blue $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 100
  red $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 50
  green $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 10
export from 2 to $\{3,\ 5\}$ :
  true $\Longrightarrow$ $r$

import at 3 from $\{1\}$ :
  true $\Longrightarrow$ $r$.local-preference := 100
import at 3 from $\{2,\ 4\}$ :
  green $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 1000
  blue $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 500
export from 3 to $\{2,\ 4\}$ :
  true $\Longrightarrow$ $r$

import at 4 from $\{1\}$ :
  true $\Longrightarrow$ $r$.local-preference := 10
import at 4 from $\{3,\ 5\}$ :
  green $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 50
  blue $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 25
export from 4 to $\{3,\ 5\}$ :
  true $\Longrightarrow$ $r$

import at 5 from $\{1,\ 2,\ 4\}$ :
  green $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 2
  red $\in$ $r$.colors $\Longrightarrow$ $r$.local-preference := 1
export from 5 to $\{2,\ 4\}$ :
  true $\Longrightarrow$ $r$
```

Figure 4.3: Example policy configurations in $PL_{\mu bgp}$.

scriptors that arise after applying an export transform but before applying the corresponding input transform, we combine these functions into *arc policy functions* for convenience.

DEFINITION 4.2.10. Let I be an instance of $(PL,\ PV)$ with signaling graph $G = (V,\ E)$; let $\{v,\ u\} \in E$ be any edge. Then the *arc policy function* $F_{(v,u)}$ is the function which takes the path descriptors at u and produces the path descriptors that v has after import from u. Thus, for $X \subseteq \mathcal{R}$,

$$F_{(v,u)}(X) = t^{in}\left(v,\ u,\ F^{in}(v,u), t^{out}(u,\ v,\ F^{out}(u,v),\ X)\right).$$

Note that it may be the case that $F_{(v,u)}(X) = \emptyset$ for some $X \neq \emptyset$. In this case we say that the path descriptors of X have been *filtered out by* $F_{(v,u)}$.

Conditions (1)–(3) given in Definition 4.2.3 only need to hold for the functions $\{F_{(v,\ u)} \mid \{v,\ u\} \in E\}$; however, because t^{out} and t^{in} are specified separately from the policies F^{out} and F^{in}, it may be easier for those designing the protocol transformations t^{in} and t^{out} to assume that all policies satisfying L^{out} or L^{in} also satisfy these conditions (and for the compilers of policies into functions to know that it suffices to satisfy these conditions).

Suppose that the path P is a simple path in G from a node v to node w; we write this as a sequence $P = vx_1 \ldots x_k w$ of distinct nodes starting with v and ending with w. If $r_w \in F^{orig}(w)$, then we let $r(P,\ r_w) \subseteq \mathcal{R}$ be the result of passing r_w along P and applying the corresponding arc policies. Formally, if $P = w$, set $r(w, r_w) = \{r_w\}$. If $v \neq w$ then write $P = vx_1 \ldots x_k w = vP'$ and let $r(vP', r_w) = F_{(v,x_1)}(r(P', r_w))$.

DEFINITION 4.2.11. The set of path descriptors *realizable at u in I* is the set $\mathcal{R}_I^u$ of descriptors which may be originated at u or which may be obtained by (legally) originating a descriptor elsewhere and passing it along a network path, successively transforming it with the appropriate arc policies:

$$\mathcal{R}_I^u = F^{orig}(u) \cup \left\{ r' \in r(P,\, r_w) \mid w \in V,\; r_w \in F^{orig}(w),\ \text{and } P \text{ is a path from } u \text{ to } w \right\}.$$

4.2.5 PATH-VECTOR SOLUTIONS

A solution for an instance of a path-vector system is an assignment of path descriptors to nodes which is both realizable and which satisfies each node's preferences to as great an extent as possible given the assignments to the surrounding nodes.

DEFINITION 4.2.12. A *path assignment* ρ is a mapping from V to $2^{\mathcal{R}}$. Given a path assignment ρ, define the set of *candidates* at node v to be

$$C(\rho,\, v) = F^{orig}(v) \cup \left\{ r \in \mathcal{R} \mid (\{v,\, u\} \in E) \wedge \left(r \in F_{(v,\, u)}(\rho(u)) \right) \right\},$$

i.e., those path descriptors which are either originated at v or which are the result of importing descriptors assigned by ρ to v's neighbors.

DEFINITION 4.2.13. For $X \subseteq \mathcal{R}$, let the set $\min(X)$ be the set of descriptors in X (for all destinations) which are minimally ranked among the descriptors with the same destination, *i.e.*, define

$$\min(X) = \{ r \in X \mid \forall r' \ dest(r') = dest(r) \Rightarrow \omega(r) \leq \omega(r') \}.$$

The assignment ρ is a *solution for I* if for each $v \in V$ we have (1) $\rho(v) \subseteq \mathcal{R}_I^v$ and (2) $\rho(v) = \min(C(\rho,\, v))$.

For the instance I, let $sol(I)$ be set of solutions for I. Note that it may be the case that $sol(I) = \emptyset$.

RUNNING EXAMPLE, PART 6. The unique solution $\rho_{\mu bgp}$ to the instance from part 5 of our running example is shown in Table 4.1. Note that the sub-instance obtained by deleting the edge $\{1,\ 5\}$ from the graph has two solutions; so, this instance is not robust.

v	$\rho_{\mu bgp}(v)$
1	$\{(d,\ 0,(1),\ 1,\ \emptyset)\}$
2	$\{(d,\ 50,\ (2,1),\ 1,\ \{red\})\}$
3	$\{(d,\ 1000,\ (3,4,5,1),\ 4,\ \{green\})\}$
4	$\{(d,\ 50,\ (4,5,1),\ 5,\ \{green\})\}$
5	$\{(d,\ 2,\ (5,1),\ 1,\ \{green\})\}$

Table 4.1: Unique solution for $PV_{\mu bgp}$ running example.

4.3 EXAMPLES

We first discuss two points in the design space that were mentioned in the overview and then present an additional, more complex example.

4.3.1 SHORTEST-PATHS ROUTING

EXAMPLE 4.3.1 (Shortest Paths). Let $\mathcal{R}_{sp} = \mathcal{D}_{sp} \times \mathbb{N} \times \mathrm{Seq}(\mathbb{N})$. The second component of $r \in \mathcal{R}_{sp}$ is a non-negative length associated with the path in the third component of r; this length is the sole factor in path ranking, with shorter paths preferred. We permit nodes to increment the length of a path on import or export, so that $\mathrm{L}^{in} = \mathrm{L}^{out} = \mathrm{L}_{sp}$ where $\mathrm{L}_{sp}(f)$ holds iff there exists a positive integer n such that for all $d \in \mathcal{D}_{sp},\ m \in N,\ P \in \mathrm{Seq}(\mathbb{N})$, we have $f(\{(d,\ m,\ P)\}) = \{(d,\ m+n,\ P)\}$.

We define the export policy-application function $t^{out}_{sp}(u,\ v,\ f,\ X)$ to produce the set

$$\{(d,\ m, uP) \mid (d,\ m,\ P) \in f(X)\}\,.$$

That is, t^{out}_{sp} merely extends the path P with the node u. We define the import policy-application function $t^{in}_{sp}(u,\ v,\ f,\ X)$ to produce the set

$$f\left(\{r \mid r = (d,\ l, P) \in X \text{ where } P \text{ is a simple path}\}\right).$$

That is t^{in}_{sp} eliminates path descriptors with a loop, and then applies the import policy.

REMARK 4.3.2. Note that by replacing $\mathrm{Seq}(\mathbb{N})$ with $\mathbb{N}$ we could model "distance vector" protocols similar to RIP [HEN88]. However, we will restrict our attention to those systems that do not allow signaling paths of arbitrary length.

EXAMPLE 4.3.3 (Shortest-Available Paths). This system is a slight extension of Shortest Paths in which path descriptors can be filtered out, both on import and export. We simply modify the local constraints L^{in} and L^{out} to allow filtering, leaving all other definitions unchanged. The new constraint $\mathrm{L}_{sap}(f)$ holds iff there exists a positive integer n such that for all $d \in \mathcal{D}_{sp},\ m \in N,\ P \in \mathrm{Seq}(\mathbb{N})$, either $f(\{(d,\ m,\ P)\}) = \emptyset$ or $f(\{(d,\ m,\ P)\}) = \{(d,\ m+n,\ P)\}$.

4.3.2 A CATALAN EXAMPLE

We now give an example that is rather unlike traditional routing problems and suggests the broad applicability of the framework we have presented. The policy-application functions of this path-vector system ensure that the path descriptors which are passed between nodes are those whose paths are subpaths of lattice paths related to the famous Catalan numbers. We thus denote this path vector system by PV_{cat}. The set $\mathcal{U}_{cat}$ includes ∞, and the ranking function ω_{cat} is constructed so that exactly the desired lattice paths are given finite rank; subpaths of the desired paths are not filtered but instead given infinite rank.

The policies written by nodes in an instance of this system do not affect which paths are imported and exported; they only determine the rank of the path descriptors which are constrained by PV_{cat} to have finite rank. Given the myriad of combinatorial interpretations of the Catalan numbers, there are many ways that nodes in an instance of PV_{cat} can interpret and then "naturally" order the path descriptors that they receive from their neighbors. We suggest a few such policies below.

THE PATH VECTOR SYSTEM PV_{cat}

We assume that each node in an instance of PV_{cat} has a neighbor one step to the north and one step to the east (as though points with integer coordinates in $\mathbb{R}^2$) and that the protocol knows the spatial relationship between neighbors.

Let $\mathrm{Seq}(0,1)$ be the set of all finite $0-1$ sequences, and let

$$\mathcal{R}_{cat} = \mathcal{D}_{cat} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathrm{Seq}(0,1).$$

We then make the following definitions.

$$\begin{aligned} \mathcal{U}_{cat} &= \mathbb{N} \cup \{\infty\} \\ dest_{cat}(d,\ x,\ y,\ z,\ P) &= d \\ \omega_{cat}(d,\ x,\ y,\ z,\ P) &= \begin{cases} z, & x = y \\ \infty, & \text{otherwise} \end{cases} \end{aligned}$$

In $r = \{d,\ x,\ y,\ z,\ P\}$, we use P to encode the corresponding path (using 0 for east steps and 1 for north steps) and x and y to store the number of east and north steps in the path.

We let $\mathrm{O}_{cat}(X)$ hold if and only if $r \in X \Rightarrow r = (d,\ 0,\ 0,\ m,\ \epsilon)$, where ϵ is the empty sequence. Let $\mathrm{L}_{cat}(f)$ hold if and only if for every $r = (d,\ x,\ y,\ z,\ P) \in \mathcal{R}_{cat}$, $f(\{r\}) = \{(d,\ x,\ y,\ z',\ P)\}$, so that f may only change the fourth element of the path descriptor. We take L_{cat} to be the constraint on both import and export functions.

$$\begin{aligned} \mathrm{L}^{in}_{cat}(f) &= \mathrm{L}_{cat}(f) \\ \mathrm{L}^{out}_{cat}(f) &= \mathrm{L}_{cat}(f) \end{aligned}$$

REMARK 4.3.4. Note that L_{cat} ensures that policies do not filter paths as in Shortest Paths (Example 4.3.1). This could be changed to allow filtering as in Shortest-Available Paths (Example 4.3.3).

We define the export policy-application function $t_{cat}^{out}(u,\ v,\ f,\ X)$ to be the set

$$\{(d,\ x+1,\ y,\ z,\ 0P) \mid (d,\ x,\ y,\ z,\ P) \in f(X)\}$$

if v is 1 step east of u, the set

$$\{(d,\ x,\ y+1,\ z,\ 1P) \mid (d,\ x,\ y,\ z,\ P) \in f(X)\}$$

if v is 1 step north of u, and $\emptyset$ otherwise. Thus t_{cat}^{out} restricts the export of descriptors to those neighbors which are one step east or north from the exporting node. It also updates the path P, prepending a 0 or 1, depending on whether this export is to the east or north, and the total number of east (x) and north (y) steps in P. Note that we do not make assumptions about the labels of the nodes (although we could express these restrictions using node labels from $\mathbb{N} \times \mathbb{N}$). We define $t_{cat}^{in}(u,\ v,\ f,\ X)$ to be the set

$$f(\{(d,\ x,\ y,\ z,\ P) \in X \mid y \geq x\}).$$

The combination of t_{cat}^{out} and t_{cat}^{in} ensures that the path descriptors which have not been filtered correspond to paths with north and east steps and that, starting at the destination, have never made more east steps than north steps as they are forwarded. It is well known that the number of such paths with exactly n steps north and n steps east is the n^{th} Catalan number $\frac{1}{n+1}\binom{2n}{n}$. The definition of ω_{cat} means that the path descriptors which have finite rank are exactly those which have passed along equally many north and east steps. While PV_{cat} determines the set of descriptors which are assigned finite rank at each node, it has no impact on the ordering of the descriptors in this set. These rankings will be determined by the policies of nodes in an instance of PV_{cat} and may correspond to natural orderings on some of the many families of objects counted by the Catalan numbers (66 examples of which are given in Exercise 6.19 of [STA99]).

POLICIES FOR PV_{cat}

Assume that we have some policy language PL_{cat} for PV_{cat} in which a node can describe: a family of objects counted by the Catalan numbers; a ranking of these objects; and an appropriate bijection between the objects and Catalan sequences. (A *Catalan sequence* of size n is an element of $\mathrm{Seq}(0,1)$ with n 0s and n 1s, such that no initial subsequence has more 0s than 1s.) We now consider different policy functions, compiled from policies written in PL_{cat} and which satisfy L_{cat}, which may arise in PV_{cat}. These functions must be of the form

$$f(\{d,\ x,\ y,\ z,\ P\}) = \{(d,\ x,\ y,\ z',\ P)\},$$

so we will define the functions below by defining z' in each instance.

The first two examples use as objects lattice paths (*i.e.*, composed of the steps $(1,0)$ and $(0,1)$) from $(0,0)$ to (n,n) which never fall below the diagonal $y = x$. They also use the bijection described in the definition of PV_{cat} in which a 1 appearing in an element of $\mathrm{Seq}(0,1)$ corresponds to a step of $(0,1)$ in a lattice path. For the first example, let the ranking of a path be its *area*, *i.e.*, the number of whole squares below the path and above the diagonal $y = x$. The import function then sets z' to be the area of the path corresponding to P. For our second example, we prefer shorter paths to longer ones, and given two paths of the same length, we prefer the one which has the $(1,0)$ step at the first step where they differ. For a sequence P of length $2n$, the import function then sets z' to be $\sum_{i=1}^{n-1}\binom{2i}{i}/(i+1)$ plus the number of paths of length $2n$ that have a $(1,0)$ step in the first place where they differ from P.

Among all paths of length $2n$, the path along the diagonal (alternating north and east steps) will be the most preferred using both of these policies, while the path consisting of n steps north followed by n steps east will be the least preferred. However, the first policy will prefer *any* path along the diagonal to any other path, regardless of the lengths of the two paths, in contrast to the second policy. They will also disagree on the relative rankings of the two paths encoded by $P_1 = 1011111\ldots00000\ldots$ and $P_2 = 110010101010\ldots$.

Policies might also be written which view the object encoded by a sequence P of length $2n$ as an ordering π of $\{1, \ldots, n\}$ which does not have three (possibly non-adjacent) elements in decreasing order (a 321-avoiding permutation). (See [STA99] for a bijection to the lattice paths we have been considering.) The import function could assign to z' any number of values, including various permutation statistics (*e.g.*, descents, inversions) evaluated on π. Once the path P is viewed as a permutation, there are a wide variety of ways to define z.

4.4 EXPRESSIVENESS

To rigorously capture the *expressive power* of path-vector systems, we use the Stable-Paths Problem (SPP) [GSW02] as a sematic domain. In this section, we show how to map path-vector instances to equivalence classes of SPP instances and use this to compare the expressiveness of path-vector policy systems.

4.4.1 MAPPING PATH-VECTOR SYSTEMS TO SPP INSTANCES

Suppose that $I = (G(V, E),\ F)$ is an instance of some $(PV,\ PL)$. We may represent I as a set of instances of the Stable-Paths Problem (SPP). For each $w \in V$ and each $r_w \in F^{orig}(w)$ we construct an SPP instance $S_{(I,w,r_w)}$.

DEFINITION 4.4.1. Define $I(w,\ r_w)$ to be a *restriction* of instance I where the only descriptor originated is r_w at node w. Given $I(w,\ r_w)$, define the corresponding SPP instance $S_{(I,w,r_w)}$

as described below, and let

$$\mathcal{S}(I) = \{I(w,\ r_w) \mid w \in V,\ r_w \in F^{orig}(w)\}$$

be the set of all SPP instances which correspond to a restriction of I.

Let the set of permitted paths in $S_{(I,w,r_w)}$ be $\mathcal{P}_{(I,w,r_w)} = \{P \mid r(P,\ r_w) \neq \emptyset\}$. For each $v \in V$, set the values of the ranking function $\lambda^v_{(I,w,r_w)}$ such that the following holds: $\lambda^v_{(I,w,r_w)}(P_1) \leq \lambda^v_{(I,w,r_w)}(P_2)$ if and only if $\{r_1\} = r(P_1,\ r_w)$, $\{r_2\} = r(P_2,\ r_w)$, and $\omega(r_1) \leq \omega(r_2)$.

It may be that $\lambda^v_{(I,w,r_w)}(P_1) = \lambda^v_{(I,w,r_w)}(P_2)$ for paths $P_1 \neq P_2$. This can happen in one of two ways. First, it may be the case that $r(P_1,\ r_w) = r(P_2,\ r_w)$. That is, two distinct signaling paths may result in the same path descriptor. Or, it may be the case that $r_1 = r(P_1,\ r_w) \neq r(P_2,\ r_w) = r_2$, but $\omega(r_1) = \omega(r_2)$.

There is an exact correspondence between the set of solutions for I and the set of solutions for $\mathcal{S}(I)$ as shown by the following theorems.

THEOREM 4.4.2. *If π is a solution for $S_{(I,w,r_w)}$, then*

$$\rho_\pi(v) = \bigcup_{P \in \pi(v)} r(P,\ r_w)$$

is a solution for $I(w,\ r_w)$.

Proof. It is clear that for each v, all path descriptors in $\rho_\pi(v)$ are realizable. We must show that for each v, $\rho_\pi(v) = \min(C(\rho_\pi,\ v))$. If $v = w$, then $\rho_\pi(w) = \{(w)\} = \min(C(\rho_\pi,\ w))$ by definition. Suppose that $v \neq w$. We first note that for any $Y \subseteq \mathcal{P}^v$,

$$\begin{aligned} A &= \bigcup_{P \in \min(\lambda^v,\ Y)} r(P,\ r_w) \\ &= \min\left(\bigcup_{P \in Y} r(P,\ r_w)\right) \\ &= B, \end{aligned}$$

because

$$\begin{aligned}
r \in A \quad &\iff \quad \exists\, P \in \min(\lambda^v,\ Y) : \{r\} = r(P,\ r_w) \\
&\iff \quad \exists\, P \in Y,\ \{r\} = r(P,\ r_w) : \forall\, P' \in Y,\ \lambda^v_{(I,\ w,\ r_w)}(P) \leq \lambda^v_{(I,\ w,\ r_w)}(P') \\
&\iff \quad \exists\, P \in Y : \forall\, P' \in Y,\ \{r\} = r(P,\ r_w), \{r'\} = r(P,\ r_w),\ \text{and}\, \omega(r) \leq \omega(r') \\
&\iff \quad \forall\, r' \in \left(\bigcup_{P \in Y} r(P,\ r_w) \right),\ \omega(r) \leq \omega(r') \\
&\iff \quad r \in B.
\end{aligned}$$

Let $Y = \{(vQ \in \mathcal{P}^v \mid \{v,\ u\} \in E \text{ and } Q = \pi(u)\}$. Because π is a solution we have $\pi(v) = \min(\lambda^v,\ Y)$ and

$$\begin{aligned}
\rho_\pi(v) \quad &= \quad \bigcup_{P \in \pi(v)} r(P,\ r_w) = \bigcup_{P \in \min(\lambda^v,\ Y)} r(P,\ r_w) = \min \left(\bigcup_{P \in Y} r(P,\ r_w) \right) \\
&= \quad \min \{ r \in \mathcal{R} \mid \exists\, P \in Y : r \in r(P,\ r_w) \} \\
&= \quad \min \{ r \in \mathcal{R} \mid \exists\, P \in \{vQ \in \mathcal{P}^v \mid \{v,\ u\} \in E,\ Q = \pi(u)\} : r \in r(P,\ r_w) \} \\
&= \quad \min \left\{ r \in \mathcal{R} \mid \{v,\ u\} \in E \text{ and } r \in \bigcup_{Q \in \pi(u)} r(vQ,\ r_w) \right\} \\
&= \quad \min \left\{ r \in \mathcal{R} \mid \{v,\ u\} \in E \text{ and } r \in \bigcup_{Q \in \pi(u)} F_{(v,\ u)}(r(Q,\ r_w)) \right\} \\
&= \quad \min \left\{ r \in \mathcal{R} \mid \{v,\ u\} \in E \text{ and } r \in F_{(v,\ u)} \left(\bigcup_{Q \in \pi(u)} r(Q,\ r_w) \right) \right\} \\
&= \quad \min \left\{ r \in \mathcal{R} \mid \{v,\ u\} \in E \text{ and } r \in F_{(v,\ u)}(\rho_\pi(u)) \right\} \\
&= \quad \min C(\rho_\pi,\ v),
\end{aligned}$$

which completes the proof. □

THEOREM 4.4.3. *If ρ is a solution for $I(w,\ r_w)$, then $\pi_\rho(v) = \{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq \rho(v)\}$ is a solution for $S_{(I,w,r_w)}$.*

Proof. We need to show that for each v we have $\pi_\rho(v) = \min(\lambda^v,\ \text{candidates}(v, \pi_\rho))$. Because ρ is a solution for $I(w,\ r_w)$, we know that $\rho(v) = C(\rho,\ v) = \min(F^{orig}(v) \cup Y)$, where

$$Y = \{r \in \mathcal{R} \mid \{v,\ u\} \in E \text{ and } r \in F_{(v,\ u)}(\rho(u))\}.$$

It is easy to show that for any X we have

$$\{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq \min(X)\} = \min(\lambda^v, \{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq X\}).$$

When $v \neq w$, then

$$\begin{aligned}
\pi_\rho(v) &= \{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq \rho(v)\} \\
&= \{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq \min(Y)\} \\
&= \min(\lambda^v, \{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq \{r \in \mathcal{R} \mid \{v, u\} \in E \text{ and } r \in F_{(v,u)}(\rho(u))\}\}) \\
&= \min(\lambda^v,\ \{(vQ \in \mathcal{P}^v \mid \{v, u\} \in E \text{ and } Q = \{P' \in \mathcal{P}^v \mid r(P',\ r_w) \subseteq \rho(u)\}\}) \\
&= \min(\lambda^v,\ \{(vQ \in \mathcal{P}^v \mid \{v, u\} \in E \text{ and } Q = \pi_\rho(u)\}) \\
&= \min(\lambda^v,\ \text{candidates}(v, \pi_\rho))
\end{aligned}$$

When $v = w$, note that $\rho(v) = \{r_w\}$, so we have

$$\pi_\rho(v) = \{P \in \mathcal{P}^v \mid r(P,\ r_w) \subseteq \{r_w\}\} = \{(w)\} = \min(\lambda^v,\ \text{candidates}(v, \pi_\rho)).$$

□

THEOREM 4.4.4. *$\pi_{\rho_\pi} = \pi$ and $\rho_{\pi_\rho} = \rho$.*

Proof.

$$\begin{aligned}
\pi(v) &= \left\{P \in \mathcal{P}^v \mid \emptyset \neq r(P,\ r_w) \subseteq \bigcup_{Q \in \pi(v)} r(Q,\ r_w)\right\} \\
&= \{P \in \mathcal{P}^v \mid \emptyset \neq r(P,\ r_w) \subseteq \rho_\pi(v)\} \\
&= \pi_{\rho_\pi}(v).
\end{aligned}$$

$$\begin{aligned}\rho(v) &= \bigcup_{P \in (\{P \in \mathcal{P}^v \mid \emptyset \neq r(P,\, r_w) \subseteq \rho(v)\})} r(P,\, r_w) \\ &= \bigcup_{P \in \pi_\rho(v)} r(P,\, r_w) \\ &= \rho_{\pi_\rho}(v).\end{aligned}$$

□

RUNNING EXAMPLE, PART 7. An SPP corresponding to our running example is presented in Figure 4.4. Node 1 is the origin. Next to each node are the permitted paths of that node listed in order of preference, starting with the most preferred at the top. Note that the actual values of the ranking function are not important, only the relative preference of each permitted path at each node; this figure can be taken to represent an entire equivalence class of SPPs with different values for each λ^v but the same orderings on each set $\mathcal{P}^v$.

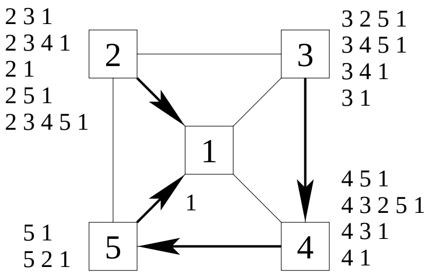


Figure 4.4: SPP for $PV_{\mu bgp}$ running example.

4.4.2 DEFINITION OF EXPRESSIVE POWER

Two distinct SPPs can represent the same set of solutions because the specific values in $\mathbb{N}$ that a ranking function λ^v takes on are not really important—what is important is the *relationship* between the rankings of permitted paths at a given node v.

For any SPP instance S, define two relations, $\ominus_S$ and $\oslash_S$, on permitted paths $\mathcal{P}$. First, $P_1 \ominus_S P_2$ if and only if $P_1,\ P_2 \in \mathcal{P}$ and P_1 is a subpath of P_2, *i.e.*, there exists a path Q (possibly ϵ, the empty path) such that $QP_1 = P_2$. Note that $\ominus_S$ is a partial order on permitted paths. Second, $P_1 \oslash_S P_2$ if and only if there is a $v \in V$ such that $P_1,\ P_2 \in \mathcal{P}^v$ and $\lambda^v(P_1) \leq \lambda^v(P_2)$. Define relation $\odot_S$ to be the transitive closure of the relation $\ominus_S \cup \oslash_S$.

DEFINITION 4.4.5. We say that two SPPs S_1 and S_2 are *equivalent* if they are defined on the same graph, have the same set of permitted paths, and $\odot_{S_1} = \odot_{S_2}$. Define the set $\mathcal{E}(S)$ to be the set of all SPPs equivalent to S.

DEFINITION 4.4.6. We define the *expressive power* of a path-vector policy system $(PV,\ PL)$ as the set

$$\mathcal{M}(PV,\ PL) = \{\mathcal{E}(S) \mid S \in \mathcal{S}(I) \text{ for some } (PV,\ PL)\text{-instance } I\}.$$

$\mathcal{M}(PV)$ means the maximal expressive power of PV when it is not constrained by a policy language, *i.e.*, the maximal expressive power of PV with respect to a policy language allowing all legal policy functions to be expressed.

REMARK 4.4.7. We note that

$$\mathcal{M}(PV_{sp}) \subsetneq \mathcal{M}(PV_{sap}) \subsetneq \mathcal{M}(PV_{\mu bgp}).$$

Shortest-Available Paths (PV_{sap}) allows nodes to filter routes while Shortest Paths (PV_{sp}) does not. Any routing configuration in PV_{sp} is captured by PV_{sap}. But, given any configuration permitted in PV_{sp}, we can filter one of the routes and obtain a new configuration where the policies are permitted by PV_{sap} but not PV_{sp}; thus, $\mathcal{M}(PV_{sp}) \subsetneq \mathcal{M}(PV_{sap})$. Likewise, because $PV_{\mu bgp}$ essentially allows nodes to rank routes in any order, it permits a routing configuration where a node prefers a longer path to a shorter one. Therefore its expressive power is more than that of PV_{sap}.

4.5 ROBUSTNESS

We first define robustness using SPP semantics and then present a natural class of expressive, robust SPPs, characterizing this class in the path-vector framework.

4.5.1 DEFINITION OF ROBUSTNESS

DEFINITION 4.5.1. An instance I over $(PV,\ PL)$ is said to be *robust* if it has a unique solution and every sub-instance of I has a unique solution. If every instance of a path-vector policy system $(PV,\ PL)$ is robust, then $(PV,\ PL)$ is said to be *robust*.

DEFINITION 4.5.2. In a similar manner, we can define robustness of SPP instances. Define the set

$$\mathcal{RSPP} = \{\mathcal{E}(S) \mid S \text{ is a robust SPP instance}\}.$$

Given the results of the previous section, we then see that a path-vector policy system $(PV,\ PL)$ is robust if and only if

$$\mathcal{M}(PV,\ PL) \subseteq \mathcal{RSPP}.$$

We are interested in the design space of robust path-vector policy systems.

CONJECTURE 4.5.3. *For every $(PV,\ PL)$, if $\mathcal{M}(PV,\ PL) \subseteq \mathcal{RSPP}$, then there exists an $\mathcal{E}(S) \in \mathcal{RSPP}$ such that $\mathcal{E}(S) \notin \mathcal{M}(PV,\ PL)$. In other words, no path-vector policy system can capture exactly all robust systems.*

4.5.2 A NATURAL SET OF ROBUST SYSTEMS

DEFINITION 4.5.4. The SPP S is *almost-partially ordered* if $\odot_S$ is reflexive, transitive, and obeys the following rule:

RULE 4.5.5. *$P_1 \odot_S P_2$ and $P_2 \odot_S P_1$ implies that $P_1 = P_2$ or $\exists\, v$ such that $P_1,\ P_2 \in \mathcal{P}^v$.*

(Traditional notions of antisymmetry and partial ordering for $\odot_S$ do not allow permitted paths of equal rank at any node; thus, we use the slightly modified notion given above.) Then let

$$\mathcal{APOSPP} = \{\mathcal{E}(S) \mid S \text{ is almost-partially ordered}\}$$

be the set of all almost-partially ordered equivalence classes of SPPs.

If the SPP S is almost-partially ordered, then we will write $P_1 \leq P_2$ for $P_1 \odot_S P_2$, and we will write $P_1 < P_2$ if $P_1 \odot_S P_2$ but $P_2 \not\odot_S P_1$.

THEOREM 4.5.6. *If an SPP instance S is almost-partially ordered, then it is robust.*

The following lemma connects dispute wheels and Definition 4.5.4, and will be useful in proving Theorem 4.5.6.

LEMMA 4.5.7. *The SPP S is almost-partially ordered if and only if it has no dispute wheel.*

Proof. First, suppose that S is almost-partially ordered. Furthermore, suppose that S has a dispute wheel with R_i, Q_i as in Definition 3.2.1 Because $\lambda^{u_i}(Q_{i-1}) \leq \lambda^{u_i}(R_iQ_i)$, we know that $R_iQ_i \leq Q_{i-1}$ because $\leq$ subsumes relation $\oslash_S$. And because Q_i is a subpath of R_iQ_i, we know that $Q_i < R_iQ_i$. Therefore, $Q_i < Q_{i-1}$. Following this chain of inequalities around the dispute wheel yields the contradiction $Q_i < Q_i$. Therefore, S has no dispute wheel.

For the other direction, suppose that S has no dispute wheel and also assume that S is not almost-partially ordered. If S is not almost-partially ordered, then there must exist paths P_1 and P_2 that violate Rule 4.5.5 because the relation $\odot_S$ is inherently reflexive and transitive; *i.e.*,

$$\begin{aligned} &\exists\, P_1 \neq P_2 \text{ such that} \\ &\quad \text{(i)} \quad P_1 \odot_S P_2, \\ &\quad \text{(ii)} \quad P_2 \odot_S P_1, \text{ and} \\ &\quad \text{(iii)} \quad \forall_{v \in V} : \{P_1, P_2\} \not\subset \mathcal{P}^v \end{aligned}$$

Conditions (i) and (ii) imply that there exist sets of paths $\{Y_i\}$ and $\{Z_j\}$, not necessarily distinct, such that

$$P_1 = Y_1 \ominus_S Y_2 \oslash_S \cdots \ominus_S Y_{n-1} \oslash_S Y_n = P_2$$

and

$$P_2 = Z_1 \ominus_S Z_2 \oslash_S \cdots \ominus_S Z_{n-1} \oslash_S Z_n = P_1,$$

respectively. From (iii) we know that it is not the case that $P_1 \oslash_S P_2$ or that $P_2 \oslash_S P_1$; if $P_1 \ominus_S P_2$ and $P_2 \ominus_S P_1$ then $P_1 = P_2$, which is not possible if P_1 and P_2 violate Rule 4.5.5. Therefore, there must be intervening distinct paths in the cycle of relationships above, *i.e.*, $(\{Y_i\} \cup \{Z_j\}) \setminus \{P_1, P_2\} \neq \emptyset$. Using the "cycle of paths" in $\{Y_i\} \cup \{Z_j\}$, we can build a dispute wheel: if $X_1 \ominus_S X_2$ for $X_1, X_2 \in \{Y_i\} \cup \{Z_j\}$, then X_1 is a subpath of X_2 and X_1 can be a spoke path while X_2 can be the spoke path X_1 exported to a rim neighbor; then $X_2 \oslash_S X_3$ and X_2 is the rim path preferred to the spoke path X_3, *etc.*

The existence of a dispute wheel in S is a contradiction; thus S is almost-partially ordered. □

With Lemma 4.5.7 in hand and a result from [GSW02], we can proceed with the proof of Theorem 4.5.6.

Proof. If S is almost-partially ordered, then by Lemma 4.5.7 it has no dispute wheel. Then by Theorem 3.2.9, S is robust. □

REMARK 4.5.8. An alternative proof may be possible using fixed point theory. As remarked in Definition 3.2.2, the solutions of the SPP are exactly the fixed points of F, because $F(\pi) = \pi$ implies π is a solution, and for any solution π we have $F(\pi) = \pi$. Perhaps there is some relation that we can impose on the function space of path assignments so that if S is almost partially ordered, then: (1) this relation is partially ordered; (2) F is monotonically increasing; and (3) F is continuous with respect to this order. Then the above proof could dispense with dispute wheels and instead use standard fixed point theorems.

THEOREM 4.5.9. *If $\mathcal{M}(PV, PL) \subseteq \mathcal{APOSPP}$, then the path-vector policy system (PV, PL) is robust.*

Proof. This follows from Theorem 4.5.6. □

4.5.3 INCREASING PATH-VECTOR SYSTEMS

DEFINITION 4.5.10. The SPP instance S is *increasing* if

$$\lambda^u(Q) < \lambda^v(vQ)$$

for all edges $\{u,\ v\}$ with path Q permitted at u and path vQ permitted at v. (We are comparing the rankings assigned by *different* nodes; these values have no *a priori* relationship.) Let

$$\mathcal{ISPP} = \{\mathcal{E}(S) \mid S \text{ is increasing}\}$$

be the set of all increasing equivalence classes of SPPs.

LEMMA 4.5.11. *If $S \in \mathcal{APOSPP}$, then there exists an SPP instance $S' \in \mathcal{ISPP}$ such that $S' \in \mathcal{E}(S)$.*

Proof. We give an iterative process that will converge to a path-ranking function Λ that is increasing.

Define the *path-rank function for node v at step k* to be λ^v_k. For all $v \in V$ and $P \in \mathcal{P}^v$, let $\lambda^v_k(P) = \infty$ for all $k \leq 0$. For $k > 0$, define λ^v_k as follows: At every node $v \neq v_0$, consider exactly the paths permitted at v, $\mathcal{P}^v$, which have the form vuP', where either $u = v_0$ and $P' = \epsilon$ or $u \neq v_0$ and $uP' \in \mathcal{P}^u$. List these in decreasing order of preference as $P_1 = vu_1P'_1,\ P_2 = vu_2P'_2,\ \ldots,\ P_i = vu_iP'_i$. (Ties can be broken arbitrarily.) If $u_1 = v_0$, then let

$$\lambda^v_{k+1}(P_1) = 1,$$

and if $u_1 \neq v_0$ let

$$\lambda^v_{k+1}(P_1) = \lambda^{u_1}_k(P'_1) + 1.$$

For the less preferred paths P_j, $2 \leq j \leq i$, if $u_j = v_0$, let

$$\lambda^v_{k+1}(P_j) = \lambda^v_k(P_{j-1}) + 1,$$

and for $u_j \neq v_0$ let

$$\lambda^v_{k+1}(P_j) = \max\left\{\lambda^{u_j}_k(P'_j),\ \lambda^v_{k+1}(P_{j-1})\right\} + 1.$$

Assume that all undefined values of λ are ∞ in the above.

Assuming that the set of permitted paths is closed under the taking of subpaths, if the longest permitted path in the SPP has k edges, then for all $v \in V$ and for all $P \in \mathcal{P}^v$, $\lambda^v_{k'}(P) \neq \infty$ for every $k' \geq k$. The path-rank functions will stabilize over iterations if the SPP S is almost-partially ordered, so in S', let

$$\Lambda(v) = \lim_{k\to\infty} \lambda^v_k.$$

Note that in using the above iterative process, ranks are always set higher than neighboring ranks because of the increment used in defining λ^v_k. Indeed, $\lambda^v(vuP) > \lambda^u(uP)$ after convergence, thus Λ and S' are increasing.

Finally, it is clear that $S' \in \mathcal{E}(S)$, because the ranking given by the converging import functions is consistent with the SPP preference list at every node. $\square$

REMARK 4.5.12. Any almost-partially ordered SPP can be convered to an increasing SPP using the method described above. It can also be shown that an SPP which cannot converge with respect to the above process (*i.e.*, for some $P \in \mathcal{P}^v$, there does not exist any integer k' such that $\lambda_k^v(P) \neq \infty$ for $k \geq k'$) must have a dispute wheel and thus is not almost-partially ordered.

THEOREM 4.5.13. $\mathcal{APOSPP} = \mathcal{ISPP}$.

Proof. Clearly $\mathcal{ISPP} \subseteq \mathcal{APOSPP}$ because if the SPP S is increasing, its preferences are already consistent with the subpath relation so that $\odot_S$ is an almost-partial order; so, we only need to show that $\mathcal{APOSPP} \subseteq \mathcal{ISPP}$. If S is an SPP such that $\mathcal{E}(S) \in \mathcal{APOSPP}$, then we can *topologically sort* the permitted paths of S, as in Lemma 4.5.11. We can then create a new SPP S' by creating a new ranking function λ' which both respects this topological order (so that the system is increasing) and which has the same relative preferences as λ. Clearly $\mathcal{E}(S) = \mathcal{E}(S')$; as S' is increasing, $\mathcal{E}(S) \in \mathcal{ISPP}$. $\square$

Ideally, we would like to construct a $(PV,\ PL)$ pair such that $\mathcal{M}(PV,\ PL) = \mathcal{ISPP}$, thus obtaining expressiveness and robustness. We now examine two ways to modify the running-example system $PV_{\mu bgp}$ so that the result is an increasing path-vector system. As we see in the next section, each of these systems lacks some desirable property, a conflict which is in fact unavoidable (Theorem 4.6.9).

EXAMPLE 4.5.14. System PV_{up} shares local preferences between nodes (therefore, it is not policy-opaque) and has local policy constraints that enforce increasing rank between neighbors. Modify the definition of t^{out} so that the local-preference value is passed between neighbors:

$$t^{out}_{up}(u,\ v,\ f,\ X) = \{(d,\ m,\ uP,\ u) \mid (d,\ m,\ P,\ x) \in f(X)\}.$$

Let the export constraint be

$$\mathrm{L}^{out}_{up}(f) \Leftrightarrow \forall\, r,\ \omega(\{r\}) \leq \omega(f(\{r\}))$$

and let the import constraint be

$$\mathrm{L}^{in}_{up}(f) \Leftrightarrow \forall\, r,\ \omega(\{r\}) < \omega(f(\{r\})).$$

That is, we constrain the legal policies to be those that increase path rank; in theory, such policies can be written because nodes have access to neighbors' local-preference values.

EXAMPLE 4.5.15. System PV_{force} modifies both protocol transformations so that they filter out descriptors whose rank does not increase under the application of the policy function in question. If $r = (d,\ l,\ P,\ n) \in X$, define $h(r) = (d,\ 0,\ P,\ n)$. Then let $t^{in}_{force}(u,\ v,\ f,\ X)$ be the set

$$\{f(\{h(r)\}) \mid r \in X \text{ describes a simple path and } \omega(\{r\}) < \omega(f(\{h(r)\}))\}$$

and let $t^{out}_{force}(u,\ v,\ f,\ X)$ be the set

$$\{(d,\ l,\ uP,\ u) \mid r = (d,\ l,\ P,\ x) \in f(X) \text{ and } \omega(\{r\}) \leq \omega(f(\{r\}))\}.$$

REMARK 4.5.16. Note that $\mathcal{M}(PV_{up}) = \mathcal{M}(PV_{force}) = \mathcal{ISPP}$.

4.6 AUTONOMY, TRANSPARENCY, AND OPAQUENESS

The systems in Examples 4.5.14 and 4.5.15, while robust and expressive, each lack one of the desirable properties defined in this section. These drawbacks are just examples of a more general design trade-off presented below.

4.6.1 AUTONOMY

Network operators often require a high degree of *autonomy* when defining routing policies, *i.e.*, they want wide latitude to write policies that reflect their own interests.

We first define a general notion of autonomy. A collection of predicates on path descriptors, such that exactly one predicate holds for each descriptor in $\mathcal{R}$, induces a partition Π of $\mathcal{R}$. A partial order on these predicates induces a partial order on $\mathcal{R}$. A path-vector policy system is autonomous with respect to $(\Pi, \leq_\Pi)$ if there exists a legal policy that ranks routes consistent with the partial order on Π induced by $\leq_\Pi$.

For example, a policy writer may wish to rank routes solely as a function of the value of one particular attribute of descriptors in the system. If he or she is to do so with full freedom, the system must be autonomous with respect to every partial ordering of the collection of predicates which test the value of that attribute. A system without this autonomy may have local-policy constraints preventing the desired policy configuration.) We can say that the space of ordered partitions given which a path-vector policy system is autonomous represents the *autonomy* of the system, and that *full autonomy* is reached when policy writers can write policies consistent with all possible partitions.

Formally, we have the following.

DEFINITION 4.6.1. A path-vector system PV is *autonomous with respect to partition* Π *of* $X \subset \mathcal{R}$ iff for any partial order $\leq_\Pi$ on the partition, there exists a legal import policy f (*i.e.*, $\mathrm{L}^{in}(f)$ holds) such that for all $r_i \in \Pi_i, r_j \in \Pi_j$ with $\Pi_i <_\Pi \Pi_j$, there exist $\hat{r}_i, \hat{r}_j \in \mathcal{R}$ such that

$$(f(r_i) = \hat{r}_i \text{ and } f(r_j) = \hat{r}_j) \Rightarrow \omega(\hat{r}_i) < \omega(\hat{r}_j).$$

Useful partition types, as described above, include partitions based on attributes, *e.g.*, "Let $r \in \Pi_i \subset \mathcal{R}$ iff $A(r) = i$" (the index set of the partition is the set of possible attribute values $A(r)$). If PV is autonomous with respect to such a partition, we will say that PV is autonomous with respect to A.

REMARK 4.6.2. If PV is *autonomous with respect to A and B together* (*i.e.*, "Let $r \in \Pi_{\{ij\}} \subset \mathcal{R}$ iff $A(r) = i$ and $B(r) = j$"), then PV is both autonomous with respect to A and autonomous with respect to B. The converse of this is not true.

DEFINITION 4.6.3. The *autonomy* of a path-vector system PV is

$$\mathcal{A}(PV) = \{\Pi \mid PV \text{ is autonomous with respect to } \Pi\}$$

One intuitive definition for the concept of full autonomy might be that PV is autonomous with respect to all possible predicates Π. However, this is not reachable. To give a more useful definition, we first introduce the following concept.

DEFINITION 4.6.4. $Q(r,\ v)$ is an *importability predicate* iff $Q(r,\ v)$ holds if t^{in} applies some $F^{in}(v, u)$ to $r \in X \subset \mathcal{R}$.

DEFINITION 4.6.5. PV has *full autonomy* iff there exists a PL such that for all instances I over $(PV,\ PL)$ and all vertices v in the instance graph there exists an importability predicate $\mathrm{Imp}(r, v)$ such that for all partitions Π of $\{r \in \mathcal{R} \mid \mathrm{Imp}(r, v)\}$, PV is autonomous with respect to Π.

This definition of full autonomy is more reasonable because it includes node independence and limits the scope of path descriptors considered to those that are actually imported at a given node. Informally then, a path-vector system has full autonomy when imported path descriptors can be ranked freely at every node.

We now define a more specific notion of autonomy suitable for BGP-like systems. It describes the ability to classify neighbors, *e.g.*, so that an ISP can prefer routes from customers over routes from peers.

DEFINITION 4.6.6. The path-vector policy system $(PV,\ PL)$ supports *autonomy of neighbor ranking* if, for every instance I, node v, and a partition $C_1,\ C_2, \ldots, C_k$ of the set of neighbors of v, there exists a legal import policy at v that does not filter routes such that, for $1 \leq j \leq k - 1$, v always prefers routes sent from partition C_j over those sent from partition C_{j+1}.

Note that autonomy of neighbor ranking is simply autonomy with respect to a partition on the value of the next hop (or path vector) attribute of "importable" path descriptors.

The system PV_{up} in Example 4.5.14 does not support autonomy of neighbor ranking. However the system PV_{force} in Example 4.5.15 does, but in what might be called a *draconian* manner, *i.e.*, the policy-application functions enforce increasing rank even if the policy writer's policies do not – routes that are not increasing in rank are simply filtered out by the protocol (not the policies).

4.6.2 PROTOCOL TRANSPARENCY

This brings us to another important property for policy writers: they should be able to easily understand the semantics of policies that they write. For example, the import-policy application $Y = t^{in}(v,\ u,\ f,\ X)$ is defined with the user-supplied policy f as input, but there is no guarantee that the policy writer can easily understand why the output Y is obtained.

DEFINITION 4.6.7. Suppose there exists a function $\hat{t}^{in}$ whose definition does not depend on f, such that $t^{in}(v,\ u,\ f,\ X) = f(\hat{t}^{in}(v,\ u,\ X))$. Then PV is said to apply import policies *transparently*. Similarly, if there exists a function $\hat{t}^{out}$ such that $t^{out}(v,\ u,\ f,\ X) = \hat{t}^{out}(v,\ u,\ f(X))$, then PV is said to apply export policies *transparently*. If both of these conditions hold, then PV is *transparent*. In this case, we can define the function

$$t(v,\, u,\, X) = \hat{t}^{in}(v,\, u,\, \hat{t}^{out}(u,\, v,\, X))$$

and note that

$$F_{(v,\, u)}(X) = F^{in}(v,\, u)(t(v,\, u,\, F^{out}(u,\, v)(X))).$$

That is, the transformation between two neighboring nodes participating in PV can be easily understood as the composition of three functions: the export policy at one node; a fixed, uniform transformation t given by PV; and the import policy at another node.

REMARK 4.6.8. The system PV_{force} is not transparent, but PV_{up} and $PV_{\mu bgp}$ are.

4.6.3 A DESIGN TRADE-OFF

We saw that the systems PV_{up} and PV_{force} are both robust, yet one supports autonomy of neighbor ranking but is not transparent while the other is transparent but does not support autonomy of neighbor ranking. This is just one example of a more general design trade-off:

THEOREM 4.6.9. *If $(PV,\ PL)$ is any path-vector policy system such that $\mathcal{M}(PV,\ PL) = \mathcal{APOSPP}$, then either $(PV,\ PL)$ does not support autonomy of neighbor ranking, or PV is not transparent, or both.*

Proof. The SPP instance GOOD GADGET in Figure 4.5(a) is in $\mathcal{APOSPP}$, so it must be expressible by some $(PV,\ PL)$ instance. If $(PV,\ PL)$ supports autonomy of neighbor ranking, then node 2 can change its policies to prefer paths through node 3, producing the SPP instance BAD GADGET in Figure 4.5(b) which has no solution. Therefore, because $\mathcal{M}(PV, PL) = \mathcal{APOSPP}$, the policy-application functions of PV must not allow this policy to take effect, *i.e.*, the system is not transparent. □

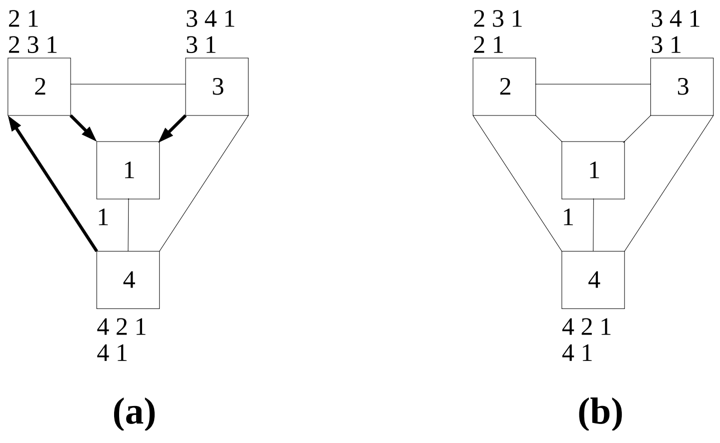


Figure 4.5: (a) The SPP GOOD GADGET and its unique solution shown in bold. (b) The SPP BAD GADGET.

4.6.4 POLICY OPAQUENESS

Policy writers might often think of autonomy and transparency in terms of path-descriptor attributes. In particular, a policy writer might be concerned with what freedom he or she has to change a path-descriptor attribute and what effect such a change might have. A related concern, the property of *policy opaqueness* that we discuss in this section, is whether attribute settings are shared with neighbors or kept private. On one hand, the exchange of information might be important to allow policy writers to make important conditional assignments that affect ranking or the overall robustness of the system; on the other hand, policy writers may not want to disclose their changes to path-descriptor settings (especially when these changes should not influence others).

Informally, an opaque system is one where policy-related attributes are kept hidden when path descriptors are exchanged between nodes. It is expected that this "information hiding" occurs in the protocol transform functions (specifically t^{out}, because we expect t^{in} to be executed by a router that is different than the one that last set attribute values) as a built-in transformation to the path descriptor.

So that we may conveniently discuss the opaqueness of a system in terms of which attributes are shared and which are kept private, we make the following definition. Let r^{-A} be the path descriptor r with attribute A removed.

DEFINITION 4.6.10. Attribute A is *opaque* iff, for any two path descriptors $r_1, r_2 \in \mathcal{R}$, $r_1^{-A} = r_2^{-A}$ implies that

$$t^{out}\left(v, u, F^{out}(v,u), \{r_1\}\right) = t^{out}\left(v, u, F^{out}(v,u), \{r_2\}\right)$$

for all v, u (*i.e.*, either r_1 and r_2 are both filtered or they produce the same descriptor).

An opaque attribute, then, is one that is essentially cleared on export (after application of t^{out}).

REMARK 4.6.11. The local-preference attribute is opaque in the system PV_{force} and in the system $PV_{\mu bgp}$, but not in the system PV_{up}. In this case, the opaqueness of local-preference and autonomy of neighbor ranking are closely intertwined because adjusting rank for next-hop involves adjusting the local-preference value accordingly; this is not arbitrarily permitted in PV_{up}. It is the implementation of ranking restrictions in PV_{up} that removes the opaqueness of local preference. It is not generally true that loss of autonomy of neighbor ranking goes hand-in-hand with a loss of opaqueness.

4.7 GLOBAL CONSTRAINTS

Theorem 4.6.9 shows that the expressive power of $\mathcal{APOSPP}$ can be reached only if a path-vector policy system gives up either transparency or some autonomy. However, both of these may be very important in many applications. In this section, we discuss an approach that will allow us to move beyond this dilemma: relying on global assumptions in the network.

The expressive power of a path-vector policy system is largely dictated by the *local constraints* included in the specification and those enforced by the policy language. We introduce the complementary notion of a *global constraint* as any function κ that maps any $(PV,\ PL)$ instance I to $\{\text{TRUE},\ \text{FALSE}\}$.

DEFINITION 4.7.1. A *globally constrained* path-vector policy system is a triple $(PV,\ PL,\ \kappa)$, where κ is a global constraint for $(PV,\ PL)$. I is a *legal instance* of $(PV,\ PL,\ \kappa)$ if I is an instance of $(PV,\ PL)$ and $\kappa(I) = \text{TRUE}$.

DEFINITION 4.7.2. Let $\mathcal{M}(PV,\ PL,\ \kappa)$ be the set

$$\{\mathcal{E}(S) \mid S \in \mathcal{S}(I) \text{ for a legal } (PV,\ PL) \text{ instance } I\}.$$

DEFINITION 4.7.3. Define the constraint κ_{apo} as

$$\kappa_{apo}(I) \Leftrightarrow \forall\, S \in \mathcal{S}(I),\ \mathcal{E}(S) \in \mathcal{APOSPP}.$$

We say that the global constraint κ is *robust* for $(PV,\ PL)$ if, for every instance I, $\kappa(I)$ implies $\kappa_{apo}(I)$.

The following theorem implies that global constraints are indeed an integral part of path-vector-system design.

THEOREM 4.7.4. *Suppose the global constraint* κ *is robust for a transparent* $(PV,\ PL)$ *allowing autonomy of neighbor ranking such that* $\mathcal{M}(PV_{sp}) \subsetneq \mathcal{M}(PV, PL, \kappa)$ *(i.e., at least as expressive as shortest paths). Then* κ *must be non-trivial.*

Proof. If we are not restricted to shortest-paths routing, then autonomy of neighbor ranking and transparency allow us to express BAD GADGET (Figure 4.5(b)). Only a non-trivial global constraint could prevent this. □

CHAPTER 5

PATH-VECTOR ALGEBRAS*

João Sobrinho presented an independently developed formal model for path-vector routing, called a *path-vector algebra* [SOB03]. It, like the PVPS framework, is a framework for protocol design, rigorously defining desirable protocol properties and identifying the conditions needed to achieve them. It establishes an abstract formalism with which to discuss protocol semantics separate from specific networks or implementation details. Furthermore, it too provides protocol-design guidelines that provably guarantee protocol convergence on any network.

This chapter establishes the relationship between path-vector algebras and path-vector policy systems. In doing so, it provides a context for understanding the process of protocol design from abstract specification to implementation. We not only prove that many results in the two frameworks are indeed equivalent, but we also show how to translate a protocol-design specification and framework-specific design properties from one model into the other. This is beneficial because, as we show, each model has its particular strengths in discussing different parts of the protocol-design puzzle. The models focus on different levels of abstraction of protocol design; being able to use the complementary machinery of these two frameworks allows a more complete analysis of a protocol.

*This chapter has previously appeared in joint work with Aaron D. Jaggard [JR05A].

5.1 DEFINITION OF PATH-VECTOR ALGEBRAS

5.1.1 COMMON DESIGN-SPACE PROPERTIES

Chapter 4 identifies several dimensions of the protocol design space and the trade-offs inherent in this space. The framework in [SOB03] is more abstract, thus it does not model all of these dimensions. In this chapter we focus on three properties regarding protocol convergence: (1) robustness, (2) expressivness, and (3) optimality. The algebra framework assumes an equivalent definition of robustness and, as we will see, an equivalent sufficient condition for it. Our translation between frameworks will add a notion of expressiveness to the algebra framework, which was not originally considered; it will also permit us to compare specifications in the two frameworks. Finally, our translation will add the notion of optimality to the PVPS framework:

DEFINITION 5.1.1. A protocol *converges optimally* if every router is assigned its most preferred path out of all possible paths (not just those it learns while running the protocol).

5.1.2 FRAMEWORK COMPONENTS

DEFINITION 5.1.2. A *path-vector algebra* [SOB03] describes some basic semantics of a path-vector protocol. It is a seven-tuple $(\mathcal{W},\ \preceq,\ \mathcal{L},\ \Sigma,\ \phi,\ \oplus,\ f)$ comprising:

$\mathcal{W}$ a set of *weights* totally ordered by $\preceq$;

$\mathcal{L}$ a set of *labels*;

Σ a set of *signatures* containing the special signature ϕ;

$\oplus$ a binary operation $\oplus : \mathcal{L} \times \Sigma \to \Sigma$; and

f a "weighing function" $f : \Sigma \to \mathcal{W}$.

Signatures model the path data structure; they contain enough information to determine a path's weight using the function f. Weights influence (but do not completely determine) choice of best route; heavier paths are less preferred. Each directed signaling edge in a network is associated with a label, which models the transformation made to a path's data structure when advertised along the edge, *i.e.*, labels correspond to import and export policies along the edge. The operation $\oplus$ computes the signature for a path advertised to a neighbor given the label on the signaling edge and the path's original signature; this amounts to *applying* the edge policy to the path data structure on extension.

Note that instantiating an algebra with a specific best-path selection procedure that obeys the weight ordering on signatures induced by f produces a protocol; this protocol may then be instantiated with a specific network and assignments of labels to edges, *etc.*, producing a routing configuration.

5.1.3 DYNAMICS

Given an algebra, a path-vector protocol consistent with it would run in accordance with the following dynamics: a node v knows of path P to d when it has a signature for P, either $s(d)$ for the empty path to d (when d is in v's own AS), or $s(P)$ for a path extending a neighbor's path to d; the best path P' to d is a path with lowest weight; to advertise a path P to u, $s(P)$ is sent along the signaling edge (v, u) with some associated label l, and $s(uP) = l \oplus s(P)$ is the signature of the imported, extended path at u.

5.1.4 ALGEBRA PROPERTIES

It is assumed that the following two properties hold for any path-vector algebra.

DEFINITION 5.1.3. An algebra obeys *maximality* iff $\forall\, \alpha \in \Sigma - \{\phi\},\ f(\alpha) \prec f(\phi)$.

DEFINITION 5.1.4. An algebra obeys *absorption* iff $\forall\, l \in \mathcal{L},\ l \oplus \phi = \phi$.

The special signature ϕ represents an unusable path, and so these properties mean that an unusable path is always least preferred and is never extended to a usable path.

The following three properties are relevant to studying protocol convergence and optimality.

DEFINITION 5.1.5. An algebra is *isotonic* iff

$$\forall\, l \in \mathcal{L},\ \forall\, \alpha, \beta \in \Sigma\ (f(\alpha) \preceq f(\beta)) \Rightarrow (f(l \oplus \alpha) \preceq f(l \oplus \beta))\,.$$

DEFINITION 5.1.6. An algebra is *monotonic* iff

$$\forall\, l \in \mathcal{L}\, \forall\, \alpha \in \Sigma,\ f(\alpha) \preceq f(l \oplus \alpha);$$

it is *strictly monotonic* iff

$$\forall\, l \in \mathcal{L}\, \forall\, \alpha \in \Sigma - \{\phi\},\ f(\alpha) \prec f(l \oplus \alpha).$$

Isotonicity implies that path extension does not reverse a strict preference relationship between paths. Monotonicity means that the actual weights of paths do not decrease as they are extended. Strict monotonicity is enough to guarantee robust protocol convergence on any network; monotonicity alone only guarantees protocol convergence on networks with the following property (*free* networks).

DEFINITION 5.1.7. A network is *free* iff for all cycles $u_n \cdots u_1 u_0$,

$$\forall\, w \in \mathcal{W} - \{f(\phi)\},\ \exists\, 0 \leq i \leq n : \forall\, \alpha \in \Sigma,\ (f(\alpha) = w) \Rightarrow (f(l(u_i, u_{i-1}) \oplus \alpha) \neq w).$$

This will be discussed in more detail in Section 5.4.

5.2 THREE LEVELS OF ABSTRACTION

In this section we examine how the two frameworks are applied to a set of simple protocols. Although these protocols have a simple route-selection procedure so that the example is easy to diagram (see Tables 5.1–5.3), the frameworks can just as easily isolate the important factors for convergence of protocols having the complexity of BGP itself.

From this example, it becomes clear that modeling occurs at three levels of abstraction: moving from (1) properties that describe a set of protocols; to (2) a specification for one particular protocol with some added implementation details; finally to (3) the properties of a given protocol on particular networks. These three levels of abstraction naturally correspond to: (1) the algebra framework; (2) the PVPS framework; and (3) instances of the Stable Paths Problem (SPP). We then ask the natural question of how the two frameworks, and these levels of abstraction, fit together. In this section, we give the intuition behind using both frameworks to analyze protocols at all three levels. Then, in Section 5.3, we give a rigorous translation between the two frameworks; this translation relates the language and notation originally suited to each framework's context. Finally, in Section 5.4, we examine the relationship between the frameworks' protocol-design guidelines using our translation.

5.2.1 EXAMPLE PROTOCOLS

Number	Primary Rank Criterion	Secondary Rank Criterion	Cost Constraint	Rank Property
(1)	cost, prefer lower	path length, prefer shorter	nondecreasing	strictly monotone
(2)	cost, prefer lower	path length, prefer shorter	none	none
(3)	path length, prefer shorter	cost, prefer lower	none	strictly monotone
(4)	cost, prefer lower	none	nondecreasing	monotone
(5)	cost, prefer lower	none	none	none
(6)	path length, prefer shorter	none	none	strictly monotone

Table 5.1: Example protocols using the path data structure $\{\mathrm{cost}, \mathrm{path\ length}\}$.

Table 5.1 gives a summary of six example path-vector protocols. In each of these protocols, the path data structure includes a path cost in $\mathbb{Z}$ and a path length in $\mathbb{N}$; we assume that nodes may modify path cost on import and export, and that the protocol automatically updates the path-length value when paths are extended. The protocols differ in which of these two components they use as the primary determinant of path rank, whether they use the other component as a secondary factor, and whether they place additional restrictions on how the path cost is modified as paths are extended. Because we are primarily interested in relating these protocols to the algebra and PVPS frameworks, we do not assume any protocol details at the level of, *e.g.*, [RL95].

Protocols (1)–(3) use both cost and path length to determine the rank of a route while Protocols (4)–(6) only use one of these components. In Protocols (1)–(2), the cost of a path is the primary criterion for determining rank, and path length is the secondary criterion: paths with lower cost are always preferred, and paths with shorter length are preferred among paths with equal cost. In Protocol (3), the reverse is true, *i.e.*, shorter paths are preferred most and ties are broken by choosing the path with lower cost. For Protocols (1) and (4), we require that path cost does not decrease when a path is extended (this is indicated in the COST CONSTRAINT column), while the other protocols allow negative costs to be associated with edges.

The column RANK PROPERTY abuses notation slightly; it uses some of the algebra properties discussed above to describe what happens to path rank in a protocol (rather than path weight in an algebra) when a path is extended. In Protocols (1), (3), and (6), the rank of a path must increase because path length is included in the calculation of rank and paths only increase in length as they are extended—thus we say that rank is strictly monotone. Note that in Protocol (2), even though path length is a component of rank calculation, rank is not strictly monotone (or even monotone). This is because, although the primary criterion for rank is path cost like Protocol (1), path cost could possibly decrease in Protocol (2), *i.e.*, this protocol does not enforce a constraint on cost values like Protocol (1) does. The lack of a constraint also tells us nothing about rank in Protocol (5). In Protocol (4), path cost is constrained to be nondecreasing and is the sole criterion for determining rank, so we can say the rank is monotone (but not strictly monotone).

5.2.2 ALGEBRAS FOR PROTOCOLS

ALG.	WEIGHT	LABEL	SIGNATURE	CONVERGENCE PROPERTY	PROTOCOLS
(A)	(cost, length), lex. ordered	edge cost $\in \mathbb{N}$, $+1$	cost and length	strictly monotone	(1)
(B)	(cost, length), lex. ordered	edge cost $\in \mathbb{Z}$, $+1$	cost and length	cost constraint	(1), (2)
(C)	(length, cost), lex. ordered	$+1$, edge cost $\in \mathbb{Z}$	length and cost	strictly monotone	(3)
(D)	cost only, prefer lower	edge cost $\in \mathbb{N}$	total cost	monotone	(1), (4)
(E)	cost only, prefer lower	edge cost $\in \mathbb{Z}$	total cost	cost constraint	(1), (2), (4), (5)
(F)	length only, prefer shorter	$+1$	length	strictly monotone	(3), (6)

Table 5.2: Algebras for protocols in Table 5.1.

Now consider the algebras in Table 5.2. Recall that weight essentially describes some part of rank calculation. Lighter paths are always preferred to heavier ones; so, to be consistent with an algebra, a protocol must evaluate a path's weight first in determining rank. By their definitions of weight, Algebras (D) and (E) can, in general, describe protocols whose primary rank criterion is path cost, Algebra (F) can describe protocols whose primary rank criterion is path length, and Algebras (A), (B), and (C) can describe protocols where rank is computed using some combination of the two in the correct order.

Of the protocols from Table 5.1, Algebra (F) can describe Protocols (3) and (6), while Algebra (C) can describe Protocol (3) only (Protocol (6) does not consider cost at all; thus, it may be the case that Protocol (6) breaks ties in path length inconsistent with the smaller-cost preference of Algebra (C)). Both of these algebras are strictly monotone because path length must increase when a path is extended, and both these protocols have strictly monotone rank (as discussed above). Thus, they are indeed consistent with the algebras' prescribed behavior. Note that both protocols implement the semantics of Algebra (F), but Protocol (3) in its specification breaks ties by cost while Protocol (6) does not. However, any protocol implementing Algebra (C) or (F) is strictly monotone and thus converges robustly for any network (see the CONVERGENCE PROPERTY column of Table 5.2);

essentially, the detail of how the protocol then looks at cost is irrelevant to convergence. So, the algebra can be used to isolate the semantics that are most useful for understanding protocol convergence. (In particular, if Protocol (6) preferred higher path cost as its secondary rank criterion, it would still converge robustly on any network.)

Similarly, Algebra (E) can describe Protocols (1), (2), (4), and (5); Algebra (B) can describe Protocols (1) and (2); Algebra (D) can describe Protocols (1) and (4); and Algebra (A) can describe Protocol (1) only. Because path cost is most important in calculating weight for these algebras, the convergence of protocols consistent with these algebras depends on whether the permitted edge costs give monotonicity or not; this is seen in the Convergence Property column in Table 5.2. Protocols consistent with Algebra (B) or (E) could permit negative edge costs, *e.g.*, Protocols (2) and (5), which do not necessarily converge robustly on any network. Indeed, this is specifically why we cannot make a general robustness claim about Algebras (B) and (E), and why any consistent protocol's convergence claim depends on the protocol's cost constraint.

Protocols (1) and (4) do have the additional constraint that cost is nondecreasing; they are not only consistent with Algebra (E) but are also consistent with Algebra (D). Because edge costs in $\mathbb{N}$ give monotonicity, we can say that Algebra (D) is monotone, and any protocol consistent with it will at least have monotone rank.

Note that Protocol (1), while consistent with a monotone algebra, has the additional property of breaking ties in path cost by using a strictly monotone data component (path length); thus, we can say that Protocol (1) has strict monotone rank—this may not be the case for any protocol described by Algebra (D), even if the cost/label set is nonnegative, *e.g.*, Protocol (4). However, the combination of nondecreasing cost and path length in Algebra (A) ensures strictly monotonic weights. Of the example protocols, only Protocol (1) is consistent with Algebra (A); Protocol (2) is inconsistent because of its lack of cost constraint, even though both cost and path length are used in computation of rank, and Protocol (4) could break ties in cost arbitrarily, as it ignores path length.

5.2.3 PATH-VECTOR SYSTEMS FOR PROTOCOLS

PV	RANK MAP	POLICY CONSTRAINT
(1)	$\omega : (c, n) \mapsto (c, n)$, lexically ordered	$\textsc{l}^{in}(f), \textsc{l}^{out}(f) : ((c', n') = f(c, n)) \Rightarrow ((n' = n) \wedge (c \leq c'))$
(2)	$\omega : (c, n) \mapsto (c, n)$, lexically ordered	$\textsc{l}^{in}(f), \textsc{l}^{out}(f) : ((c', n') = f(c, n)) \Rightarrow (n' = n)$
(3)	$\omega : (c, n) \mapsto (n, c)$, lexically ordered	$\textsc{l}^{in}(f), \textsc{l}^{out}(f) : ((c', n') = f(c, n)) \Rightarrow (n' = n)$
(4)	$\omega : (c, n) \mapsto c \in \mathbb{N}$	$\textsc{l}^{in}(f), \textsc{l}^{out}(f) : ((c', n') = f(c, n)) \Rightarrow ((n' = n) \wedge (c \leq c'))$
(5)	$\omega : (c, n) \mapsto c \in \mathbb{Z}$	$\textsc{l}^{in}(f), \textsc{l}^{out}(f) : ((c', n') = f(c, n)) \Rightarrow (n' = n)$
(6)	$\omega : (c, n) \mapsto n \in \mathbb{N}$	$\textsc{l}^{in}(f), \textsc{l}^{out}(f) : ((c', n') = f(c, n)) \Rightarrow (n' = n)$

Table 5.3: Example path-vector systems using the path descriptor $\{\mathrm{cost}, \mathrm{path\ length}\}$.

In Table 5.3, we show six path-vector systems that correspond by number to the protocols in Table 5.1. In these systems, the path descriptor is the same data structure $\{\mathrm{cost}, \mathrm{path\ length}\} \subset \mathbb{Z} \times \mathbb{N}$ used by the protocols; the predicate $\textsc{o}$ requires that originated path descriptors have the form $(0, 0)$;

$$t^{in}(u, v, f, X) = \{f(r) \mid r \in X : r \text{ describes an acyclic path}\},$$

i.e., the import transform applies import policies but filters routing loops; and

$$t^{out}(u,\ v,\ f,\ X) = \{(c, n + 1) \mid (c, n) \in f(X)\},$$

i.e., the export transform applies export policies but increments the path length automatically. The other components—the rank map ω and the local policy constraints $\textsc{l}^{in}$ and $\textsc{l}^{out}$—are shown in Table 5.3.

The correspondence with the protocols is clear: Each path-vector system describes the corresponding Table-5.1 protocol because its rank criteria are captured by the definition of ω and its cost constraint is captured by the definition of $\textsc{l}^{in}$ and $\textsc{l}^{out}$. Therefore, the consistency relationship between the Table-5.2 algebras and these PVPSes is exactly the same as that between the algebras and the Table-5.1 protocols. It can be seen that the path-vector system directly models one protocol, and any protocol that can be described by a path-vector system $PV \in \mathcal{ISPP}$ will converge on any network.

5.2.4 DISCUSSION: LEVELS OF ABSTRACTION

Using the two frameworks, we can discuss protocol design in three distinct levels of abstraction: the algebra level, the protocol level, and the network level. A move from one level of abstraction to another is not uniquely determined, as we explore below.

Because algebras do not specify all the implementation details for a protocol, an algebra can describe a set of path-vector protocols. The detail most relevant to protocol convergence is how to decide between equal-weight routes. Because weight influences path rank but does not totally determine it, some additional method must be used to rank paths of the same weight. Any strictly monotonic criterion, such as path length, guarantees robustness for that protocol. The different methods for tie-breaking correspond to different protocol instantiations of the algebra.

Just as an algebra describes a set of protocols, a given protocol might have multiple algebras that describe it. The definition of weight, labels, and signatures in an algebra correspond to only some part of the path data structure; different subsets correspond to different algebras. Thus, in translating protocols to algebras, a select part of the protocol behavior can be isolated and studied; this is a useful tool for analyzing convergence constraints for a complex protocol and an important role for the algebra framework.

Although a path-vector system does not include the bit-level details of a full protocol specification (*e.g.*, [RL95]), it does include all of the essential details of a protocol; we therefore identify a PVPS with a single protocol. At this level of abstraction, protocol designers can study the balance between enforcing constraints, convergence, and other properties using results from Chapter 4.

Any protocol, whether a PVPS or a protocol instance of an algebra, may be instantiated further to a particular network with specific node policies or edge weights and signatures. Each instance is a routing configuration permitted by that protocol; the set of permitted routing configurations is used later in this paper as a metric of expressiveness to rigorously match algebras and PVPSes. Some protocols conditionally converge and require specific constraints on the network; one of these constraints is the freeness property (Definition 5.1.7, which is an example of a *network-level* design property.

5.3 MAPPING BETWEEN FORMALISMS

5.3.1 INTUITION

It is easy to see that both frameworks model similar protocol components although different notation is used. So, before showing the rigorous translation between formalisms, we outline the intuitive relationship between framework components. Table 5.4 summarizes this correspondence (in addition to the relationship with class-based PVPSes, which will be introduced in Chapter 6) and the properties defined for each.

The rows of the table list components as follows. Paths are preferred based upon their weight in the set $\mathcal{W}$ ordered by $\preceq$ (algebra) or rank in the set $\mathcal{U}$ ordered by $\leq_{\mathcal{U}}$ (PVPS). Each framework has a function—f and ω, respectively—that assigns values from the ranking set to other objects (the set Σ of signatures in an algebra or the set $\mathcal{R}$ of path descriptors in a PVPS).[1] Each path in the network is assigned one of these objects, so we want to view

[1]Path descriptors in class-based systems, introduced in Chapter 6, have a *level* attribute g which is the primary factor in computing the rank of a path. It is natural to associate this to the primary determinant of path preference, *i.e.*, weight, in an algebra; this accounts for the differences between the second and third columns of Table 5.4.

ALGEBRA	CLASS-BASED PVPS	GENERAL PVPS
$\mathcal{W}$	$\{g\} \subset \mathbb{N}$	$\mathcal{U}$
$\preceq$	$\leq$	$\leq_{\mathcal{U}}$
Σ	$\mathcal{R}$	$\mathcal{R}$
$\mathcal{L}$	$\{f_{(u,v)}\}$	$\{f_{(u,v)}\}$
$l \oplus \sigma$	$f_{(u,v)}(r)$	$f_{(u,v)}(r)$
$\phi \in \Sigma$	$\emptyset$	$\emptyset$
$f : \Sigma \to \mathcal{W}$	$\pi_g \omega : \mathcal{R} \to \{g\}$	$\omega : \mathcal{R} \to \mathcal{U}$
monotonicity	$\pi_g \omega(r) \leq \pi_g \omega(f(r))$	$\omega(r) \leq_{\mathcal{U}} \omega(f(r))$
isotonicity	essentially, policies act by edge, not by path	
iso. & mono.	essentially, policies are consistent with a non-negative per-edge cost function	
maximality	prefer any route to $\emptyset$	
absorption	$f_{(u,v)}(\emptyset) = \emptyset$	
optimal in-tree	a solution which gives all nodes most-preferred path	
local-optimal in-tree	a path-vector solution	

Table 5.4: Informal translation between path-vector systems and algebras.

the sets Σ and $\mathcal{R}$ as containing the information about paths that is exchanged between nodes. With this in mind, it is natural to identify a function $l \oplus \_ : \Sigma \to \Sigma$, which modifies signatures passed over an edge labeled with l, with an arc policy function $f_{(u,v)}$, which modifies path descriptors exchanged over the edge (u, v); in particular, we should view the labels of edges as corresponding to the arc policies on edges. The final component of an algebra, the special signature $\phi \in \Sigma$ for unusable paths, is most naturally identified with the empty set, the result of filtering paths.

The informal translation of properties between the two frameworks shows some of their differences. Convergence properties studied in [SOB03] may be translated to the language of PVPSes as shown in Table 5.4. Many of the other design dimensions from Chapter 4 involved policies and implementation details not explicitly included in algebras; we thus do not translate these to the language of algebras, although they may of course be investigated for particular protocols derived from algebras.

Note that it is the *arc policies* (combinations of nodes' import and export policies and the policy application functions) of a PVPS that correspond to the edge labels in an algebra; the abstraction of the algebra framework does not explicitly include the separate import and export policies of individual nodes.

5.3.2 ALGEBRA-PROTOCOL CONSISTENCY

Here we make rigorous the relationship between the three levels of protocol-design abstraction that naturally follow from our example; we do this by extending the notion of expressiveness to algebras. The semantic domain used for expressiveness is the Stable Paths Problem (SPP); an SPP instance corresponds to one routing configuration (at the network level of abstraction).

The constraints of a protocol specification determine the set of network configurations possible when running a protocol. This idea motivated the definition of PVPS expressiveness; we make the analogous definition below for algebras, in which expressiveness is defined as the set of permitted routing configurations.

DEFINITION 5.3.1. Given an algebra A, let G^A be the set of all networks whose edges are assigned labels from A. For each $G \in G^A$, let $T(G, w)$ be the restriction of G where only one destination w is originated with the signature $s(w)$. Each $T(G, w)$ maps to an SPP instance $\mathcal{S}^A(T(G, w))$ in which the set of permitted paths is $\{P \mid s(P) \neq \phi\}$ and the ranking functions are consistent with weights in the following manner:

(I) if $\lambda(P) < \lambda(P')$ then $f(s(P)) \preceq f(s(P'))$,

(II) if $\lambda(P) = \lambda(P')$ then $f(s(P)) = f(s(P'))$, and

(III) if $f(s(P)) \prec f(s(P'))$ then $\lambda(P) < \lambda(P')$.

Define the *expressiveness of an algebra* A to be the set

$$\mathcal{X}(A) = \left\{\mathcal{E}(S) \mid S = \mathcal{S}^A(T(G,w)) \text{ for some } G \in G^A,\ w \in V\right\}.$$

DEFINITION 5.3.2. A path-vector system PV and an algebra A are *consistent* iff $\mathcal{M}(PV) \subseteq \mathcal{X}(A)$.

If A and PV are consistent, we may be able to relate the properties of each. Because consistency is defined in terms of equivalence classes of SPPs, we will use properties which hold for *some* SPP in each of the equivalence classes that defines the expressiveness of an algebra or PVPS. We start with the following definition.

DEFINITION 5.3.3. Let P be an SPP property. If every network instance of an algebra or PVPS is equivalent to an SPP with property P, then we say that the algebra or PVPS itself has *SPP-property* P. Formally, algebra A has property P iff for every $E \in \mathcal{X}(A)$, there exists an SPP $S \in E$ such that S has property P; analogously, PVPS PV has property P iff for every $E \in \mathcal{M}(PV)$, there exists an SPP $S \in E$ such that S has property P.

REMARK 5.3.4. We may use different terminology to describe an SPP-property P at the algebra or PVPS level; using this definition, a property $\hat{P}$ stated in terms of algebras or PVPSes is equivalent to an SPP-property P iff an algebra or PVPS satisfies $\hat{P}$ exactly when it has the SPP-property P.

EXAMPLE 5.3.5. Monotonicity is a property that can be defined in terms of SPP instances because every network instance of a monotone algebra is equivalent to an SPP with rank functions λ nondecreasing on path extension.

PROPOSITION 5.3.6. *If algebra A has SPP-property P, then every PVPS PV consistent with A also has SPP-property P.*

Proof. $\forall\, E \in \mathcal{X}(A),\ \exists\, S \in E$ such that S has property P. If PV is consistent with A, then $E \in \mathcal{M}(PV)$ implies $E \in \mathcal{X}(A)$; thus $\forall\, E \in \mathcal{M}(PV),\ \exists\, S \in E$ such that S has property P. $\square$

5.3.3 MAPPING AN ALGEBRA TO A PATH-VECTOR SYSTEM

Here we show how to use the path-vector-system framework to describe a protocol whose design specification is consistent with a given algebra. At its higher level of abstraction, an algebra describes the behavior of protocols on just a select part of the protocol's path data structure. This part of the data structure is modeled by the algebra's signature, which is used to determine weight; the actual rank of paths with the same weight is determined individually by nodes and is not modeled by the algebra.

To instantiate an algebra, we create a protocol whose data structure (the path descriptor) includes the components of the signature and an opaque local-preference attribute that can be set at a node to differentiate paths of equal weight. The ranking function ω is defined to first weigh the signature components using the algebra function $f : \Sigma \rightarrow \mathcal{W}$ and then consider local preference.

Transformations to signatures are modeled by applying labels from $\mathcal{L}$ using the operator $\oplus$. As noted above, these transformations correspond to arc policy functions. Here we use one possible combination of node policies that yield arc policies that are consistent with the label transformations of the given algebra—we constrain export policies to be the identity function on path descriptors and constrain import policies to be those policy functions that correspond to labels.

This choice does affect the implementation of the protocol and the individual expressiveness of import and export policies; in specific cases, it may be wise to change these constraints so that arc-policy functions still correspond to labels in $\mathcal{L}$ without artificially limiting the expressiveness of export of import policies. Our construction yields the following theorem.

THEOREM 5.3.7. *If $A = (\mathcal{W}, \preceq, \mathcal{L}, \Sigma, \phi, \oplus, f)$ is an algebra, the path-vector system PV with the following components is consistent with it (in particular, $\mathcal{X}(A) = \mathcal{M}(PV)$):*

$$\begin{aligned}
\mathcal{R} &= \Sigma \times \mathbb{N} \\
\mathcal{U} &= \mathcal{W} \times \mathbb{Z} \\
\leq_{\mathcal{U}} &= \preceq \times \leq \\
\omega &= (\sigma \in \Sigma,\ n \in \mathbb{N}) \mapsto (f(\sigma), -n) \\
\textsc{l}^{in}(F) &= [((\sigma', n') = F((\sigma, n))) \Rightarrow (\exists\, l \in \mathcal{L} : \sigma' = l \oplus \sigma)] \\
\textsc{l}^{out}(F) &= \textsc{true} \\
t^{in}(v, u, F, X) &= \{(\sigma', n') \mid ((\sigma', n') \in F(X)) \wedge (\sigma' \neq \phi)\} \\
t^{out}(u, v, F, X) &= \{(\sigma, 0) \mid (\sigma, n) \in X \textit{ is a loopless path when extended to } v\} \\
\textsc{o}(X) &= [((\sigma, n) \in X) \Rightarrow (\sigma \in \Sigma)]
\end{aligned}$$

Proof. Note that by construction, PV allows exactly the same paths as A.

Suppose $E \in \mathcal{M}(PV)$. Then for all SPPs $S \in E$:

(I) $\lambda(P) = \lambda(P')$ iff $\omega(r(P)) = \omega(r(P'))$; and

(II) $\lambda(P) < \lambda(P')$ iff $\omega(r(P)) < \omega(r(P'))$.

We use these properties and the definition of ω to show that $\mathcal{E}(S) \in \mathcal{X}(A)$. By definition, weight primarily determines rank, so path ranks can only be equal if path weights are equal. Thus,

$$\omega(r(P)) = \omega(r(P')) \Rightarrow f(s(P)) = f(s(P')).$$

Substituting using (i) gives

$$\lambda(P) = \lambda(P') \Rightarrow f(s(P)) = f(s(P')). \tag{5.1}$$

For the same reason, a path that has strictly lower weight must be more preferred. Therefore,

$$f(s(P)) \prec f(s(P')) \Rightarrow \omega(r(P)) < \omega(r(P')),$$

and by using (ii) we have

$$f(s(P)) \prec f(s(P')) \Rightarrow \lambda(P) < \lambda(P'). \tag{5.2}$$

Finally, a path is less preferred, by definition, if it has higher weight or it has equal weight but lower local preference. This means that

$$\omega(r(P)) < \omega(r(P')) \Rightarrow f(s(P)) \preceq f(s(P')).$$

Substituting using (ii) gives

$$\lambda(P) < \lambda(P') \Rightarrow f(s(P)) \preceq f(s(P')). \tag{5.3}$$

Statements (5.1)–(5.3) satisfy conditions (I)–(III) of Definition 5.3.1; thus $\mathcal{E}(S) \in \mathcal{X}(A)$.

□

5.3.4 DESCRIBING A PATH-VECTOR SYSTEM WITH AN ALGEBRA

The path-vector-system specification contains a definition of the path data structure, the elements involved in ranking, and constraints on policies. We can easily construct an algebra that has the same specification components.

To do this, let the set Σ of signatures and the set $\mathcal{W}$ of weights be the set $\mathcal{R}$ of path descriptors and the ranking set $\mathcal{U}$, respectively. A natural way to construct labels that correspond to arc-policy functions $f_{(u,v)}(r)$ is simply to list in a label the arguments to the policy application functions—used in the definition of $f_{(u,v)}(r)$—other than r, *i.e.*, the node names u and v and the policy functions $F^{in}_{(v,u)}$ and $F^{out}_{(u,v)}$. Labels then contain sufficient information so that $\oplus$ may used to recover $f_{(u,v)}(r)$ as $l \oplus r$, with ϕ used in place of filtered routes ($f_{(u,v)}(r) = \emptyset$).

THEOREM 5.3.8. *Given the path-vector system $PV = (\mathcal{D},\ \mathcal{R},\ \mathcal{U},\ \leq_{\mathcal{U}},\ \textsc{l}^{in},\ \textsc{l}^{out},\ \textsc{o},\ t^{in},\ t^{out})$, let*

$$L(u, v, f_1, f_2) = X \mapsto t^{in}(v, u, f_1, t^{out}(u, v, f_2, X)),$$

i.e., $L(l)$ is the arc-policy function determined by the tuple $l = (u, v, f_1, f_2)$, where $u, v \in \mathbb{N}$ represent node identifiers defining a signaling edge (u, v), and $f_1, f_2 \in 2^{\mathcal{R}}$ are the import and export policies along that edge. (These tuples are members of the constructed label set $\mathcal{L}$ below.) Then the algebra with the following components is consistent with PV:

$$\begin{aligned}
\mathcal{W} &= \mathcal{U} \\
\preceq &= \leq_{\mathcal{U}} \\
\Sigma &= \mathcal{R} \\
\phi &= \textit{signature for filtered routes} \\
\mathcal{L} &= \mathbb{N} \times \mathbb{N} \times \{F \in 2^{\mathcal{R}} \mid \textsc{l}^{in}(F)\} \\
&\quad \times \{F \in 2^{\mathcal{R}} \mid \textsc{l}^{out}(F)\} \\
\oplus &= l \oplus (r \in \mathcal{R}) \mapsto \begin{cases} L(l)(r), & L(l)(r) \neq \emptyset \\ \phi, & L(l)(r) = \emptyset \end{cases} \\
f &= \omega
\end{aligned}$$

Proof. By definition, this algebra allows exactly the same paths as PV. Suppose $E \in \mathcal{M}(PV)$. To show that $E \in \mathcal{X}(A)$, we show that for some SPP $S \in E$, conditions (I)–(III) of Definition 5.3.1 are satisfied; thus $\mathcal{E}(S) \in \mathcal{X}(A)$. By Definition 4.4.6,

$$\lambda(P) < \lambda(P') \Rightarrow \omega(r(P)) < \omega(r(P')). \tag{5.4}$$

By construction, $\omega = f$; $r(P) = s(P)$ because the labels and operations used to compute $s(P)$ can be replaced with composing arc-policy functions as discussed above. Substituting these in (5.4) satisfies condition (I), namely,

$$\lambda(P) < \lambda(P') \Rightarrow f(s(P)) \prec f(s(P')).$$

Replacing inequality with equality in the above argument gives condition (II). Finally, because $\omega = f$, if $f(s(P)) \prec f(s(P'))$, then $\omega(r(P)) < \omega(r(P'))$, and by Definition 4.4.6, this implies $\lambda(P) < \lambda(P')$, thus giving condition (III). $\square$

Although we do not give an explicit proof here, it is clear that an algebra constructed in this way for an increasing path-vector system will be strictly monotone, because the weighing function is just the increasing rank function ω.

The algebra constructed in Theorem 5.3.8 includes all the components of the path descriptor in the signature, and so is consistent only with protocols that are very similar to the original protocol. The resulting algebra may not yield the most generally applicable results; depending on the protocol, it may be more useful to construct an algebra restricted to some subset of the rank criteria (rather than the entire path descriptor), because the properties of that algebra would apply to a greater number of protocol implementations than those of the construction above. The intuition used behind mapping the components, *i.e.*, the informal correspondence the first and third columns of Table 5.4, would still apply.

5.4 EQUIVALENCE OF DESIGN GUIDELINES

Here we focus on the important convergence properties of robustness and optimality. Most of the results below were stated at the algebra level of abstraction in [SOB03]; here we prove these results at the PVPS level of abstraction, illustrating the utility of our translation discussed above.

5.4.1 MONOTONICITY AND ROBUSTNESS

The main result from both frameworks is that if paths increase in absolute rank when they are extended from router to router, the protocol is guaranteed to converge on any network. Here we show that these results are compatible. We start with the following composition of Theorems 4.5.13 and 4.5.6.

PROPOSITION 5.4.1. *An increasing PVPS is robust.*

This result is meaningful at the protocol level of abstraction, and is consistent with the following extension of Proposition 4 in [SOB03]:

PROPOSITION 5.4.2. *Any protocol (or PVPS) consistent with either a strictly monotone algebra or a monotone algebra with shortest-paths tie-breaking is robust.*

Proof. If an algebra is strictly monotone, then for every $E \in \mathcal{X}(A)$, there exists some SPP $S \in E$ all of whose ranking functions λ increase on path extension. The same is true for monotone algebras with shortest-paths tie-breaking because path length imposes strictness whenever it is not enforced by the algebra's (weakly) monotonic weight. Therefore, both types of algebras are increasing in the sense of Definition 5.3.3. By Proposition 5.3.6, any PVPS or protocol consistent with such an algebra is increasing, and thus, by Proposition 5.4.1, is robust. □

If either one of these propositions is combined with the translation between models given in Section 5.3, *e.g.*, as in the proof above, the other proposition directly follows. This shows that these results are equivalent; this is intuitively clear because any instance of a strictly monotone algebra can be described by an increasing path-vector system.

From this equivalence, we see the different strengths of the frameworks regarding protocol-design analysis. The strength of the path-vector-system model is that a design specification for an increasing system naturally yields a robust protocol whose implementation-specific properties can be studied and a framework for designing compatible policy languages. The strength of the algebra framework is that core properties can be analyzed at a higher level of abstraction; in particular, robustness guarantees can be made from examining just a subset of a protocol's rank criteria.

The following corresponds to Proposition 3 in [SOB03].

PROPOSITION 5.4.3. *Any protocol (or PVPS) consistent with a monotone algebra will converge robustly on free networks.*

Proof. Suppose that there is some such PV is not robust. Then from Definitions 4.5.1 and 4.5.2, there is some SPP S such that $\mathcal{E}(S) \in \mathcal{M}(PV)$ and S is not robust. From the contrapositive of Theorem 4.5.6, S is not almost-partially ordered; thus from Lemma 4.5.7, S contains a dispute wheel. However, the rim of the dispute wheel is a cycle of nodes that violates the property of freeness in Definition 5.1.7. Therefore, any instance on which PV does not converge robustly is not free. □

Because freeness can be checked in time proportional to a polynomial in the number of labels, signatures, and edges of a graph [SOB03], freeness is a feasible network-level global constraint guaranteeing the robustness of any protocol consistent with a monotone algebra.

Proposition 5.4.5 below is a partial converse of Proposition 5.4.2 and captures one direction of Proposition 5 in [SOB03]; we make the following definition to exclude trivial counterexamples. The first part of the proof mirrors [SOB03], and we then translate this to the language of PVPSes.

DEFINITION 5.4.4. An algebra is *range monotone* iff

$$[f(l \oplus \sigma) \prec f(\sigma)] \Rightarrow (\nexists\, l' \in \mathcal{L}, \sigma' \in \Sigma : \ \sigma = l' \oplus \sigma'),$$

i.e., any violations of monotonicity involve signatures that are not the output of the operation $\oplus$.

PROPOSITION 5.4.5. *If an algebra A is not range monotone, there exists a network instance on which some protocol consistent with A does not converge; i.e., every non-range-monotone algebra is consistent with some non-robust PVPS.*

Proof. Given such an algebra A, there exists some $\sigma \in \Sigma$ and $l \in \mathcal{L}$ such that $f(l \oplus \sigma) \prec f(\sigma)$, and there exists some $\sigma' \in \Sigma$ and $l' \in \mathcal{L}$ such that $\sigma = l' \oplus \sigma'$. Construct a network K_3 with node labels 0, 1, 2. Let node 0 originate the signature $s(0) = \sigma'$. Assign the label l' to signaling edges $(0,1)$ and $(0,2)$; assign the label l to signaling edges $(1,2)$ and $(2,1)$. Then at nodes 1 and 2, the signatures σ and $l \oplus \sigma$ are available with $f(l \oplus \sigma) \prec f(\sigma)$. This network instance corresponds to the SPP instance DISAGREE from [GSW02], which has two stable solutions and is thus not robust.

By Theorem 5.3.7, there is a path-vector system PV such that $\mathcal{M}(PV) = \mathcal{X}(A)$ and thus contains the network instance just constructed. Therefore PV is consistent with A and not robust. □

In summary, both the algebra and PVPS models can isolate properties to analyze convergence. The sufficient conditions are equivalent; the properties translate by Proposition 5.3.6.

5.4.2 ISOTONICITY AND OPTIMALITY

Sobrinho [SOB03] effectively shows that while convergence depends on monotonicity, optimal convergence depends on isotonicity. Formally, we have the following result that combines Propositions 1, 2, and 5 from [SOB03].

PROPOSITION 5.4.6. *A protocol that converges robustly and optimally is consistent with some monotone and isotone algebra. Also, there is some protocol consistent with any monotone and isotone algebra that converges robustly and optimally.*

To introduce optimality to the PVPS framework, we must define some additional properties required for the translation of results from [SOB03].

DEFINITION 5.4.7. A path-vector system PV is *nondecreasing* iff for all instances I of PV,

$$\forall\,(u,v) \in E,\ \forall\, r \in \mathcal{R},\ \omega(r) \leq \omega(f_{(u,v)}(r)).$$

DEFINITION 5.4.8. A path-vector system is *PV-isotonic* iff, in all instances, $\forall\,(u,v) \in E$ and $\forall\, r_1, r_2 \in \mathcal{R}$ such that $\omega(r_1) \leq \omega(r_2)$, one of the following hold:

$$\omega(f_{(u,v)}(r_1)) \leq \omega(f_{(u,v)}(r_2)) \text{ if } f_{(u,v)}(r_1) \neq \emptyset;$$

$$f_{(u,v)}(r_2) = \emptyset \text{ if } f_{(u,v)}(r_1) = \emptyset.$$

Nondecreasing is the analog of monotonicity just as increasing is the analog of strict monotonicity. PV-isotonicity, the analog of isotonicity, is a property both of a network and of a PVPS (in the latter case, the condition holds for all possible arc-policy functions rather than all edges in the network). A PVPS that is PV-isotone has only isotone instances and is consistent with an isotone algebra; a PVPS that is not PV-isotone has at least one instance where arc-policy functions violate the PV-isotonicity condition (this instantiation follows Proposition 5.3.6). This algebra-consistency relationship is easy to prove: Given a PVPS PV, construct a consistent algebra A using Theorem 5.3.8. From PV-isotonicity, we have

$$\omega(r(P_1)) < \omega(r(P_2)) \Rightarrow \omega(r(vP_1)) < \omega(r(vP_2)) \tag{5.5}$$

for all paths P_1, P_2 from some node u to destination d and all nodes v adjacent to u. As in the proof of Thm 5.3.8, $f = \omega$, $r(P) = s(P)$, and $r(vP) = f(l \oplus s(P))$ where l is the label corresponding to $f_{(u,v)}$; thus from (5.5) we have that

$$f(s(P_1)) \prec f(s(P_2)) \Rightarrow f(l \oplus s(P_1)) \prec f(l \oplus s(P_2)),$$

which means that A is isotone.

The following propositions about PVPSes are then analogous to Propositions 1 and 2 in [SOB03]. As noted above, their statements and proofs follow from applying our translation between algebras and PVPSes to the corresponding results for algebras [SOB03].

PROPOSITION 5.4.9. *If a path-vector system is PV-isotone and nondecreasing, then in all network instances, if destination d is reachable from u, there exists an optimal (loopless) path from u to d such that all subpaths to d are also optimal.*

Proof. First note that for any path P and some loop L extending P, $\omega(P) \leq \omega(LP)$, *i.e.*, adding loops does not decrease rank because the PVPS is nondecreasing. Any permitted path with loops can be pruned to a permitted path without loops because of PV-isotonicity: suppose node c has descriptors available for paths P and LP where L is a loop starting and ending at c; because $\omega(r(P)) \leq \omega(r(LP))$, if LP is advertised, P must also be advertised because of PV-isotonicity, so any extension of LP can be pruned of the loop L without increasing rank. Thus, if d is reachable from u, the lowest-ranked, loopless paths from u to d are optimal. Now suppose that for none of these optimal paths, every subpath to d is optimal. Choose the optimal path $P = uu_1 \cdots u_k \cdots u_n d$ with k maximal such that subpaths to d from u_i, $1 \leq i < k$, are optimal, but the subpath from u_k to d is not optimal; then there is some other path $u_k Q d$ is optimal.

Suppose this path Q intersects P such that $u_i \in Q$ for some $1 \leq i < k$. Let Q_1 be the subpath of Q from u_k to u_i, and let Q_2 be the subpath of Q from u_i to d. Because the PVPS is nondecreasing, $\omega(r(u_k \cdots u_n d)) \leq \omega(r(u_i \cdots u_n d))$ and $\omega(r(Q_2)) \leq \omega(r(Q_1 Q_2))$. Because of the optimality of $u_i \cdots u_n d$, $\omega(r(u_i \cdots u_n d)) \leq \omega(r(Q_2))$. Putting these inequalities together gives $\omega(r(u_k \cdots u_n d)) \leq \omega(r(Q_1 Q_2))$, which contradicts the assumption that $u_k \cdots u_n d$ is not optimal. Therefore, Q does not intersect P at any u_i, $1 \leq i < k$.

Form the loopless path $P' = uu_1 \cdots u_k Qd$. By PV-isotonicity,

$$\forall\, 1 \leq j < k,\ \omega(r(u_j \cdots u_k Qd)) \leq \omega(r(u_j \cdots u_n d)),$$

and $\omega(r(P')) \leq \omega(r(P))$, *i.e.*, P' is optimal. The subpath of P' from u_k to d is optimal, contradicting the maximality of k in the choice of P. Therefore, our assumption that there exists no optimal path, all of whose subpaths are all optimal, is wrong, proving the theorem.

□

PROPOSITION 5.4.10. *If a path-vector system is PV-isotone, then any nondecreasing instance has an optimal solution.*

Proof. Let π be the solution of such an instance (one exists by Proposition 5.4.9; if π is not optimal, then there is some destination d for which a node u is not assigned an optimal path. However, by Proposition 5.4.9, there exists an optimal path $Q = u_1 u_2 \cdots d$ (where $u_1 = u$) such that all subpaths $Q_k = u_k \cdots d$ are optimal. Let P_k be the assigned path in d from every $u_k \in Q$ to d (some may be empty). Choose the largest i such that P_i is (i) empty or (ii) not optimal. By maximality of i, Q_{i+1} and P_{i+1} are optimal; thus $\omega(r(Q_{i+1})) = \omega(r(P_{i+1}))$. Following the proof of Proposition 5.4.9, because the instance is nondecreasing, $u_i \notin P_{i+1}$. By PV-isotonicity, we can extend to u_i so that $\omega(r(Q_i)) = \omega(r(u_i P_{i+1}))$ and $r(u_i P_{i+1})$ is not filtered. This contradicts possibility (i) because $u_i P_{i+1}$ is available at u_i; thus P_i cannot be empty. Because P_i is the assigned path in π, $\omega(r(P_i)) \leq \omega(r(u_i P_{i+1}))$, but this contradicts (ii) because $\omega(r(P_i)) > \omega(r(Q_i)) = \omega(u_i P_{i+1})$ if P_i is not optimal. □

PROPOSITION 5.4.11. *If a path-vector system is not PV-isotone, then for any non-isotone instance I, there exists another instance I' generated by filtering a subset of permitted routes in I such that I' has no optimal solution.*

Proof. In this instance without isotonicity, there exists some edge $(u,\ v)$ with a policy for two path descriptors r_1, r_2 such that $\omega(r_1) \leq \omega(r_2)$ but either (i) $\omega(f_{(u,v)}(r_2)) < \omega(f_{(u,v)}(r_1))$ or (ii) $f_{(u,v)}(r_1)$ is filtered while $f_{(u,v)}(r_2)$ is not. Construct instance I' from I such that all paths other than those described by r_1, r_2 are filtered at v and that all paths not through v are filtered at u. In case (i), the solution to I' assigns r_1 to v because $\omega(r_1) < \omega(r_2)$ and thus assigns $f_{(u,v)}(r_1)$ to u, but this solution is not optimal because $\omega(f_{(u,v)}(r_1)) > \omega(f_{(u,v)}(r_2))$. In case (ii), the solution to I' assigns r_1 to v, but r_1 is filtered along $(u,\ v)$; thus, u cannot be assigned a path, which is not optimal. □

REMARK 5.4.12. The above theorem implies that there exists some instance of every non-PV-isotone PVPS that has no optimal solution.

CHAPTER 6

CLASS-BASED SYSTEMS*

Class-based systems focus on a generalization of next-hop-preference routing, where routing policy for an AS can be defined by the relationships (commercial or otherwise) between it and its neighboring ASes. They are transparent systems in which some type of autonomy of neighbor ranking is relevant, because route transformations depend on the partition of neighbors into these relationship *classes*. The canonical examples of such systems are the Hierarchical-BGP points in the design space mentioned in Section 4.1. Inspired by [GGR01, GR01], Hierarchical BGP (HBGP) is a simplified version of BGP that takes into account the economic realities of today's commercial Internet—that ASes partition their neighbors into customers, providers, and peers and that there are preference guidelines used to decide between routes learned from neighbors of different classes. The scope of class-based systems, however, goes beyond HBGP; the framework can also be used to build and analyze systems with complete autonomy and those that allow arbitrary next-hop preference routing. Furthermore, we will show that any protocol specification that can be described by a countable-weight, monotone path-vector algebra can also be described by a class-based path-vector system.

*Most of this chapter has previously appeared in joint work with Aaron D. Jaggard [JR04]. Portions of the chapter's introduction and Section 6.1 have appeared in joint work with Timothy G. Griffin and Aaron D. Jaggard [GJR03], and Section 6.1.3 has appeared in other joint work with Aaron D. Jaggard [JR05A].

Theorem 4.7.4 tells us that such systems require a nontrivial global constraint. In this chapter, we provide the best known robustness constraint for class-based systems. The constraint ensures the robustness of networks that satisfy it; it is in fact the best possible robustness guarantee because, in networks that do not satisfy it, some set of nodes may write policies that cause route oscillations. (Our proof of this constructs such policies.) We give an algorithm to generate the constraint given only the design specification of the system. We then provide centralized and distributed algorithms to check networks for violation of the constraint and discuss their applications, including how to use our results to check a network with arbitrary next-hop preferences for potential bad interactions. The distributed algorithm reveals almost no private policy information, provides several options for correcting a constraint violation, and has constant message complexity per link and limited storage at each node. We compare and contrast our algorithms with those in previous work.

Although it may be sufficient to provide a supplementary protocol enforcing some global conditions (and, indeed, the distributed algorithm in this paper, modified for BGP, can be run alongside BGP to detect potentially bad policy interactions), there are several benefits to this approach of analyzing robust protocol convergence from a design-framework perspective. First, the algorithms in our paper preclude all policy-based oscillations in advance; as long as the constraint is enforced, the protocol can safely run on any network. Second, the approach is an integral part of designing policy-configuration languages. The design framework identifies provably sufficient local and global conditions needed for a protocol to achieve its design goals. Our paper precisely gives the trade-off between the strength of local policy guidelines built into the policy-configuration language and the strength of the global assumption needed in the broad class-based context. The designer can use these results to consider what balance between local and global enforcement is desired and can incorporate the guidelines generated by the results in this paper into the design of multiple high-level policy languages—all before running the protocol on an actual network.

6.1 THE CLASS-BASED FRAMEWORK

6.1.1 THE CLASS-BASED PATH-VECTOR SYSTEM

We fix a BGP-like path-vector system that can implement *scoping* and *relative preference* rules dictated by class relationships (such as those in [GR01, GGR01]). By *scope*, we mean the conditions under which routes are shared with neighbors, and by *relative preference*, we mean the difference in rank assigned to routes learned from neighbors in different classes.

In our running-example system $PV_{\mu bgp}$ from Chapter 4, path descriptors r contain a *local preference* attribute $l(r)$ that can be set to assign rank based on the class of the exporting neighbor. This attribute is not shared between nodes, intuitively allowing some autonomy and opaqueness. Limited scoping can be implemented by filtering routes. However, this notion of scope is restrictive, *e.g.*, it does not allow easy flagging of a backup route, especially when the next hop might be through a neighbor of an ordinarily preferred class. Therefore, we extend the path descriptor r, following [GGR01], to include a *level* attribute $g(r)$. This attribute is nondecreasing and shared and will have precedence in ranking; thus, it can be used to communicate notions of scope that override relative-preference rules encoded in the local-preference attribute.

REMARK 6.1.1. If all nodes agreed on an encoding within local preference for indicating backup routes or some information were shared between nodes, backup-route scoping would be possible in BGP ($PV_{\mu bgp}$) without additional attributes. However, the additional attribute can separate the awkward encoding and information sharing from attributes meant for local use. The original description of HBGP+BU in [GGR01] discussed these same issues.

The components of the path-vector system PV_{cb} that we use for class-based applications are as follows.

$$\begin{aligned}
\mathcal{R}_{cb} &= \mathcal{D}_{cb} \times \mathbb{N} \times \mathbb{N} \times \mathrm{Seq}(\mathbb{N}) \\
\mathcal{U}_{cb} &= \mathbb{N} \times \mathbb{Z} \text{ (lexically ordered)} \\
dest_{cb}(d, g, l, P) &= d \\
\omega_{cb}(d, g, l, P) &= (g, -l) \\
\mathrm{O}_{cb}(X) &= (r \in X) \Rightarrow (\exists d \in \mathcal{D}_{cb}, m \in \mathbb{N} \text{ such that } r = (d, 0, 0, m)) \\
\mathrm{L}^{in}_{cb}(f) &= (((d', g', l', P') = f(d, g, l, P)) \Rightarrow (g \leq g' \wedge P = P')) \\
\mathrm{L}^{out}_{cb}(f) &= (((d', g', l', P') = f(d, g, l, P)) \Rightarrow (g \leq g' \wedge P = P')) \\
t^{in}_{cb}(u, v, f, X) &= \{(d, g, l, P) \in f(X) \mid P \text{ is a simple path}\} \\
t^{out}_{cb}(u, v, f, X) &= \{(d, g, 0, uP) \mid (d, g, l, P) \in f(X)\}
\end{aligned}$$

Note that L^{in}_{cb} and L^{out}_{cb} guarantee that the level attribute is nondecreasing and that t^{out}_{cb} guarantees that local preference is not shared. When ranking, a smaller level attribute is first preferred, then higher local preference. Also, note that PV_{cb} is transparent: let

$$t(v,\ u,\ X) = \{(d\ , g\ ,\ 0,\ uP) \mid (d,\ g,\ l,\ P) \in X \text{ where } uP \text{ is a simple path}\}$$

in Definition 4.6.7.

6.1.2 CLASS-BASED POLICY LANGUAGES

The second component of design is a policy language capable of expressing scope and relative-preference rules for class-based systems. We first make formal the notion of class

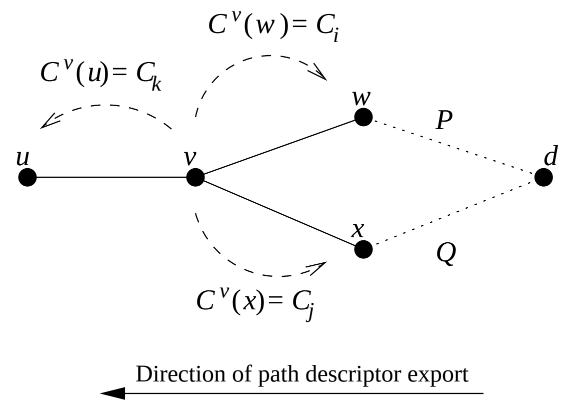


Figure 6.1: Class assignments to neighbors of node v and paths to a destination node d.

relationships. Let $C = \{C_1, C_2, \ldots, C_c\}$ be a set of *classes*. Every node $v \in V$ will have a *class-assignment function* $C^v : V \rightarrow C$ that assigns each neighbor of v a class in C. As an example, consider node v in Figure 6.1. Here, a node v with neighbors u, w, x has assigned classes C_k, C_i, C_j to these neighbors, respectively.

Policies in class-based systems may filter routes, weakly increase the level attribute, and change the local preference as desired, but nothing else. Nodes are also permitted to assign classes to neighbors. Given that assignment, policies are constrained by the three types of rules that essentially define a class-based system: relationship consistency, relative preference, and scope. To encode each of these types of rules, we will use a square matrix of dimension equal to the number of classes.

RELATIONSHIP CONSISTENCY

Class assignments might require some consistency, *e.g.*, that "customer" and "provider" assignments occur in consistent pairs. The *cross-class matrix* X describes the allowable ways that u may view v's role in their relationship given that v views u's role as $C^v(u)$. A necessary condition for an instance to be legal is that for every two adjacent nodes u and v in the network, if $C^v(u) = C_i$ and $C^u(v) = C_j$, then $X_{ij} = 1$; otherwise, $X_{ij} = 0$.

REMARK 6.1.2. Without loss of generality we may view X as a symmetric matrix (X may be replaced with the matrix $(\min(X_{ij}, X_{ji}))_{ij}$ without changing the legality of any class-based instance). Note that if $C^v(u)$ always uniquely determines the value of $C^u(v)$ allowed by X, then X has at most one 1 in each row and column; assuming no classes are superfluous, X is then a permutation matrix.

RELATIVE PREFERENCE

Relative preference rules are described by the matrix W, which has entries from the set $(\bullet) = \{<, \leq, =, >, \geq, \top\}$ of binary comparison operators, where $x \top y$ for every x, y. (We will refer to a generic operator in $(\bullet)$ with the symbol $\bullet$.) If a node v has imported path descriptors r_i and r_j from neighbors whom v views as being in classes C_i and C_j, respectively, if $\bullet = W_{ij}$, and if $g(r_i) = g(r_j)$, then the local preference values $l(r_i)$ and $l(r_j)$ must be set (via v's import policy functions) so that $\omega(r_i) \bullet \omega(r_j)$. The policy-language compiler can enforce this as a constraint on local-preference-attribute values set by import policies. We assume that the preference relations between classes specified by W are consistent with a partial order on C; *e.g.*, W should not be such that $W_{12} = W_{23} = W_{31} =$ '$<$'.

Therefore, W can be used to give a partial ordering on the classes describing which class of neighbor's routes should be preferred. For example, given the HBGP practice of preferring customer routes to peer routes, we would set $W_{ij} =$ '$<$' where C_i is the class "customer" and C_j is the class "peer." Note that W is an antisymmetric matrix of sorts (because the comparisons are antisymmetric relations, if $W_{ij} =$ '$<$' then $W_{ji} =$ '$>$') and that the only viable setting on the diagonal is $W_{ii} = \top$.[1]

[1]The setting $W_{ii} =$ '$=$' is also possible but not viable. In this case, all descriptors with the same level imported from a neighbor of class C_i would have to have equal rank, and then ties could not be broken by changing the local preference based on, *e.g.*, next-hop address or shortest path length.

SCOPE

The scoping rules in a class-based system are described by M, a $c \times c$ matrix with entries from $(\bullet) \cup \{\perp\}$. If $M_{ij} = \perp$, then v may not export path descriptors learned from a neighbor in class C_i to a neighbor in class C_j; this setting is used to prevent the exchange of routes between classes altogether (filtering). If $M_{ij} = \bullet \neq \perp$, then v may export a path descriptor learned from a neighbor in class C_i to one in class C_j; however, if r_i is the descriptor that v receives (after applying the import policy function) from a neighbor in class C_i, v's export policy function $F^{out}(v, u)$ for a neighbor u in class C_j must satisfy $g(r_i) \bullet g\left(F^{out}(v, u)(r_i)\right)$; *e.g.*, if $M_{ij} =$ '$<$', $C^v(u) = i$, and $C^v(w) = j$, then v may export routes learned from u to w, but the export policy function $F^{out}(v, w)$ must strictly increase the level attribute of these descriptors.

Various scoping conditions can be described by allowing or enforcing a change in the level attribute. One example is backup routing: Because lower levels take precedence, a backup route can be assigned a higher level value to avoid being chosen even if it passes through a preferred class.

CLASS DESCRIPTION

DEFINITION 6.1.3. A *class description* is the quadruple

$$CD = (C, X, W, M).$$

CD contains all the information necessary to generate a policy language for PV_{cb} whose "compiler," the semantic function $\mathcal{M}$, can generate tuples $(F^{in}, F^{out}, F^{orig})$ from node policies that (1) list class assignments (*i.e.*, C^v) for neighbors and (2) give local preferences and level settings for routes. Formally, the tuples will honor the scope and relative preference rules described by CD if the compiler does the following at each node v given its policy configuration p in PL:

1. For all neighbors u, let $F^{in}(v,u)$ set the local preference (and possibly level) attributes of imported path descriptors as specified in the policy configuration p. Check that for all pairs of neighbors u, w, if $C^v(u) = C_i$, $C^v(w) = C_j$, and $\bullet = W_{ij}$, then for all $r_u \in F_{(v,u)}(\mathcal{R}_I^u)$ and $r_w \in F_{(v,w)}(\mathcal{R}_I^w)$, we have that $\omega(r_u) \bullet \omega(r_w)$ if $g(r_u) = g(r_w)$.

2. For all neighbors u, let $F^{out}(v,u)$ set the level of outgoing path descriptors as specified in the policy configuration p. Then check that for all pairs of neighbors u, w, if $C^v(w) = C_i$, $C^v(u) = C_j$, and $\bullet = M_{ij}$, then for all $r \in F_{(v,w)}(\mathcal{R}_I^w)$, $g(r) \bullet g(F^{out}(v,u)(r))$, unless $\bullet = \perp$, in which case $F^{out}(v,u)(r) = \emptyset$.

The policy language can enforce the local constraints described by X, W, and M. Class consistency, along with any further conditions necessary for robustness, must be built into the accompanying global constraint.

REMARK 6.1.4. Class-based systems are autonomous with respect to next-hop class if the descriptors have the same level-attribute value; because a neighbor can be assigned any class, as long as the assignments are consistent, this essentially means that class-based systems have a restricted form of autonomy of neighbor ranking.

EXAMPLE 6.1.5. We now describe the system "Hierarchical BGP with Back-up Routes," motivated by the design guidelines in [GGR01, GR01]. It uses three classes, corresponding to standard Internet business relationships: C_1 or "customer" – someone to whom connectivity is sold; C_2 or "peer" – someone with whom an agreement is established to transit traffic, usually to shortcut more expensive or long routes; and C_3 or "provider" – someone from whom connectivity is purchased. The class consistency rules require that $C^v(u) = C_2$

implies $C^u(v) = C_2$, and $C^v(u) = C_1$ iff $C^u(v) = C_3$. The level attribute is used to mark routes that are for backup use; this is done by increasing the level on export. Because this is a shared, nondecreasing attribute that has precedence in ranking, such a back-up route will not be selected if there are routes with lower level available. Following economically motivated practice, customer routes (*i.e.*, routes learned from customers) are preferred to peer routes when their level attributes are equal; both are preferred to provider routes. The scoping rules allow customer routes to be shared with all neighbors (without requiring a level increase). Peer and provider routes may be shared with customers without a level increase; peer routes may be shared with peers and providers if the level is increased, and provider routes may be shared with peers if the level is increased. Provider routes may not be shared with other providers (as no node should carry transit traffic between two of its providers). The class-description components for this system are thus:

$$C = \{C_1,\, C_2,\, C_3\},\;\; X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix},$$

$$W = \begin{bmatrix} \top & < & < \\ > & \top & < \\ > & > & \top \end{bmatrix},\;\; M = \begin{bmatrix} \leq & \leq & \leq \\ \leq & < & < \\ \leq & < & \perp \end{bmatrix}.$$

Omitting the ability to mark routes for backup use (and generally ignoring the level attribute) yields the simpler system, "Hierarchical BGP." Its difference from HBGP+BU is in its M matrix:

$$M = \begin{bmatrix} = & = & = \\ = & \perp & \perp \\ = & \perp & \perp \end{bmatrix}.$$

6.1.3 ALGEBRAS AND CLASS-BASED SYSTEMS

By using the translation methods discussed in Section 5.3, we can discuss how to describe class-based systems using the algebra framework of [SOB03]. The level attribute naturally corresponds to the shared component of a path data structure used as the primary rank criterion, modeled by signature and weight in an algebra, while the local preference naturally corresponds to the local settings used to break ties among paths with the same weight. This informal translation was shown in Table 5.4.

The translation from algebra to class-based systems is not straightforward, however—it is intuitively clear that all algebras are not compatible with the restricted notion of a class-based path-vector system. Because the level attribute is nondecreasing, all protocols described by class-based systems must be monotonic (thus, we cannot translate nonmonotonic algebras). Furthermore, the encoding from signature components to the level-attribute integer is only clear if the weight set is countable and the weighing function f is invertible. In this case, there is a natural bijection ψ from weights to the natural numbers, and then $\sigma = f^{-1}(\psi^{-1}(g))$ is the signature σ corresponding to the level value g; the level-attribute g can be used to encode the signature σ in a path descriptor. Labels then correspond to increments in level-attribute values: If $l \oplus \sigma = \sigma'$, then l corresponds to the increment $\psi(f(\sigma')) - \psi(f(\sigma))$. However, the class structure may provide a better way of describing the constraints on the system than this encoding does, and there is no known general way to generate a set of classes for an algebra.

In the other direction, we note that a consistent algebra can be constructed to describe a class-based system where there is a total relative-preference order among the classes: The weights are (level attribute, class) pairs; the weighing function ranks paths based on level attribute first, then on the class assignment for the next-hop edge; labels are increments to level-attribute values and class assignments for the edge; and the operator $\oplus$ changes the signature of paths using the labels in accordance with the relative-preference and scoping rules of the class-based system. This mapping will not work, however, if the classes are not totally ordered in relative-preference. HBGP is an example of a total order among classes because routes are chosen such that customer routes are most preferred, then peer routes, then provider routes.

If the class-structure design principles are ignored, the following algebra is consistent with any class-based path-vector system because it models only the level attribute. The integers below correspond to level-attribute values (label integers correspond to level-attribute increments). The algebra is monotonic because the level attribute is the primary rank criterion and nondecreasing.

THEOREM 6.1.6. *Any class-based path-vector system is consistent with the following monotonic algebra:*

$$
\begin{aligned}
\mathcal{W} &= \mathbb{N} \\
\preceq &= \leq \\
\Sigma &= \mathbb{N} \\
\phi &= \textit{signature for filtered routes} \\
\mathcal{L} &= \mathbb{N} \cup \{\phi\} \\
\oplus &= n_1 \oplus n_2 \mapsto \begin{cases} n_1 + n_2, & n_1, n_2 \neq \phi \\ \phi, & n_1 \textit{ or } n_2 = \phi \end{cases} \\
f &= \textit{identity}
\end{aligned}
$$

Proof. As in the proof of Theorem 5.3.8, we show that conditions (I)–(III) from Definition 5.3.1 are satisfied for some SPP S in every $E \in \mathcal{M}(PV)$, so that $\mathcal{E}(S) \in \mathcal{X}(A)$. By the algebra construction, $f(s(P)) = g(P)$, where $g(P)$ is the level attribute of P, so we make this substitution in the conditions.

If $\lambda(P) < \lambda(P')$, then it cannot be that $g(P) > g(P')$, otherwise a path with higher level value could be more preferred; this implies $g(P) \leq g(P')$, satisfying condition (I).

If $\lambda(P) = \lambda(P')$, then $\omega(r(P)) = \omega(r(P'))$, which can only happen if $g(P) = g(P')$, satisfying condition (II). Finally, if $g(P) < g(P')$, then $\omega(r(P)) < \omega(r(P'))$; thus $\lambda(P) < \lambda(P')$, satisfying condition (III). $\square$

6.2 CLASS-BASED GLOBAL CONSTRAINTS

DEFINITION 6.2.1. Let the class-consistency constraint C be defined as

$$\forall\, u, v \in V,\ (C^v(u) = C_i) \Rightarrow (C^u(v) \in \{C_j \in C \mid X_{ij} = 1\})\,.$$

DEFINITION 6.2.2. Let $\text{K}_{cb} = \text{C} \wedge \text{J}$, where J is some constraint such that K_{cb} is robust for PV_{cb} with respect to some PL of the form described above.

The goal of this chapter is to suitably define the robustness check J, given a class description.

6.2.1 ROBUSTNESS AND DISPUTE WHEELS

A sufficient global constraint for the Hierarchical-BGP systems in Example 6.1.5 is that the AS-level signaling graph does not contain any customer-provider cycles (*i.e.*, cycles in which each node views one of its two neighbors in the cycle as a customer and the other as a provider) [GR01]. Our goal here is to find good global constraints for class-based systems in general. Given the results from Section 4.5.2, we know that a good starting point for guaranteeing robustness is precluding dispute wheels. Because of the preference and scoping rules associated with class-based systems, we can more easily find potential dispute wheels given the class assignments made by nodes.

PROPOSITION 6.2.3. *The path descriptors corresponding to all paths R_iQ_{i+1} and Q_i on a dispute-wheel rim in an SPP mapped from a class-based instance have equal level-attribute values.*

Proof. In this proof, SPPs are those mapped from instances of a path-vector system; so, if $P \in \mathcal{P}^v$ in the SPP $S \in \mathcal{S}(I)$, define $d(P) \in \mathcal{R}_{cb}$ as the realizable path descriptor for path P at node v in the path-vector instance I. Recall that $g(r)$ is the level attribute of r.

Assume we have a dispute wheel in some SPP $S \in \mathcal{S}(I)$; then for all i, $\lambda^{v_i}(R_i Q_i) < \lambda^{v_i}(Q_{i-1})$, so $\omega(d(R_i Q_i)) < \omega(d(Q_{i-1}))$; this means that

$$g(d(R_i Q_i)) \leq g(d(Q_{i-1})). \tag{6.1}$$

Level is nondecreasing, so

$$g(d(Q_i)) \leq g(d(R_i Q_i)). \tag{6.2}$$

Equations (6.1) and (6.2) together imply that $g(d(Q_i)) \leq g(d(Q_{i-1}))$ for all i; iterating around the wheel yields $g(d(Q_{i-1})) \leq g(d(Q_i))$, thus

$$g(d(Q_{i-1})) = g(d(Q_i)) = g(d(R_i Q_i))$$

for all i, which completes the proof. □

We will use the above result to aid us in generating an appropriate global constraint. Therefore, in our analysis, we are primarily interested in whether or not the entries of W and M allow equality. We thus define two $\{0,1\}$-matrices $\hat{W}$ and $\hat{M}$ as follows.

DEFINITION 6.2.4.

1. Let $\hat{W}_{ij}$ be 0 if W_{ij} permits equality (including $\top$). Let $\hat{W}_{ij} = -1$ if $W_{ij} =$ '$<$' and let $\hat{W}_{ij} = 1$ if $W_{ij} =$ '$>$'.

2. Let $\hat{M}_{ij}$ be 1 if M_{ij} permits equality, *i.e.*, $M_{ij} =$ '=', '$\leq$', *etc.*, and let $\hat{M}_{ij}$ be 0 otherwise.

The following notation and conventions make it easier to discuss class assignments on a dispute wheel.

DEFINITION 6.2.5.

1. Given an undirected network edge $\{v,\ u\}$, we will often refer to it as a directed edge contextually in the direction of signaling, *i.e.*, along $e = (u,\ v)$, u exports path descriptors to v. We write $e' = (v,\ u)$ for the edge in the opposite direction (forwarding or import).
2. The *class of an edge* $e = (v,\ u)$ is $\mathbf{c}(e) = C^v(u)$.
3. Let $\mathbf{x}(C_i) = \{C_j \mid X_{ij} = 1\}$. Then for any edge e, $\mathbf{c}(e') \in \mathbf{x}(\mathbf{c}(e))$.
4. Let $\mathbf{m}(C_i) = \{C_j \mid \hat{M}_{ij} = 1\}$ and let $\mathbf{m}^{-1}(C_i) = \{C_j \mid \hat{M}_{ji} = 1\}$.

So, $\mathbf{m}(C_i)$ is the set of classes to which a path descriptor learned from a neighbor of class C_i can be exported without an increase in level, and $\mathbf{m}^{-1}(C_i)$ is the set of classes from which a path descriptor exported to a neighbor of class C_i without an increase in level could have been imported.

DEFINITION 6.2.6. A dispute wheel is *homogeneous of type* C_k iff for all edges e along the rim, $\mathbf{c}(e) = C_k$. A dispute wheel is *heterogeneous of types* $C_{k_1}, C_{k_2}, \ldots$ iff for all edges e along the rim, $\mathbf{c}(e) \in \{C_{k_1}, C_{k_2}, \ldots\}$.

6.2.2 GENERATING A GLOBAL CONSTRAINT

PVPS solvability, and thus checking robustness, is *NP*-hard in general [GSW02]; we will thus try to find an easy-to-check global constraint in the class-based context which rejects as few robust instances as possible. Proposition 6.2.3 and the matrices in a class description together restrict the edge types (and pairs of edge types) which may appear on the rim of a dispute wheel. Because Theorem 4.5.9 tells us that precluding dispute wheels guarantees

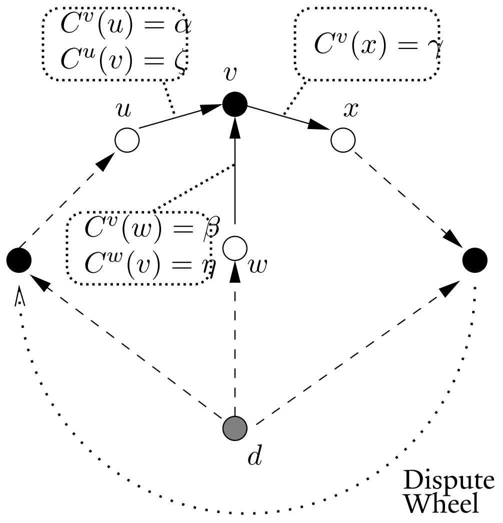


Figure 6.2: Active node v of a dispute wheel.

robustness for all instances, we can use those restrictions to produce a sufficient global constraint.

Call the rim nodes at the ends of paths R_i in a dispute wheel *active nodes*; these are the nodes at which route preferences cause the dispute wheel. In Figure 6.2, node v is an active node with neighboring rim nodes u and x—these may or may not be active nodes themselves. The neighbor on the spoke is w; the dispute occurs because the route through u is preferred over the direct route through w, but the route from w is (or has been) exported to x. Note that edges in Figure 6.2 have arrows in the direction of signaling or export.

Proposition 6.2.3 tells us that the level-attribute values of dispute-wheel paths are the same at the rim, so the matrix M must permit level equality for the class assignments on a dispute-wheel rim. The matrix W must also permit the preferences required by Definition 3.2.1 (a necessary condition for a dispute wheel). In particular, any of the following conditions at one active node would preclude the dispute wheel in Figure 6.2; they are written from the perspective of active node v without loss of generality.

(C1) v could not import a descriptor from w and export it to x without increasing level, *i.e.*, $\hat{M}_{\beta\gamma} = 0$;

(C2) v must prefer routes from w over those from u, *i.e.*, $\hat{W}_{\alpha\beta} = 1$; or

(C3) u could not export a descriptor from a neighbor and export it to v without increasing level, *i.e.*, $\forall\, \psi \in C,\ \hat{M}_{\psi\zeta} = 0$; *etc.*

The above conditions could be achieved by forcing the class assignments α, β, γ, ψ, and ζ to specific values. Note that in (C1) and (C3), a particular pair of class assignments (β and γ, or ψ and ζ) is troublesome because of the associated M-matrix entry. This idea of pairs of assignments will be used later to describe the general global constraint.

We now prove a result based on the discussion above. The following gives conditions that are necessary for an edge to participate in a dispute-wheel rim (*i.e.*, it generalizes (C1) and (C3) above).

PROPOSITION 6.2.7. *Let edge e be on the rim of a dispute wheel. If $\mathbf{c}(e) = C_\alpha$, then*

1. $\exists\, C_\beta \in C : \hat{M}_{\beta\alpha} = 1$*; i.e.,* $\mathbf{m}^{-1}(C_\alpha) \neq \emptyset$*; and*

2. $\exists\, C_\gamma \in C,\ C_\psi \in \mathbf{x}(\mathbf{m}^{-1}(C_\alpha)) : \hat{M}_{\gamma\psi} = 1.$

Proof. Condition (1) follows directly from Proposition 6.2.3.

To prove condition (2), let $e = \{v,\ u\} \in E$ and let $C^v(u) = C_i$. If e is on a dispute wheel rim, then by Proposition 6.2.3, there must be a class assignment of another node w by v such that v can export to u a path descriptor from w without increasing the level attribute. But when an edge lies on a dispute wheel rim, it imports a descriptor from two nodes, one along a spoke edge and one also on the rim; so, this condition is true for both the node adjacent to the spoke edge leading to v and for the node adjacent to the rim edge leading to v. This condition, in turn, applies to the rim edge $\{w,\ v\}$ as well (a dispute wheel must contain at least two distinct directed edges). Note that we cannot iterate further around the wheel, because w could import from rim edge e. $\square$

If, for each class C_α, either one of these conditions does not hold or, if both do hold, no edges of class C_α are allowed, then all instances will be dispute-wheel free.

THEOREM 6.2.8. *Given an instance signaling graph G and class assignments, consider the subgraph H containing only edges e where $\mathbf{c}(e) = C_\alpha$ as in Proposition 6.2.7. If H is acyclic then there is no dispute wheel.*

Proof. Dispute wheel rims must contain edges satisfying the condition in Proposition 6.2.7. Thus if the signaling subgraph containing only these edges is acyclic, no cycle of these edges, including a dispute wheel in the general signaling graph, is possible. $\square$

EXAMPLE 6.2.9. In the case of HBGP or HBGP+BU, this cycle check reduces to checking against customer-provider cycles. A simple case-by-case analysis of possible class assignments, given the constraints in matrices C and M, shows that the only dispute wheels possible are cycles in the customer-provider relationship graph. Consider the other possibilities of edges on the dispute wheel:

1. Suppose we have a rim edge $e = \{v,\ u\}$ where $C^v(u) = C_3$. Then node v must import from a node w without increasing the level attribute; however, only M_{31} permits equality so $C^v(w) = C_1$. Because only $X_{13} = 1$, we have that $C^w(v) = 3$. If w is on a spoke, then because W prefers routes from C_1 neighbors such as w, the adjacent rim node must also be of class C_1. Thus the only situation is one where the adjacent rim edge to v must have the same assignment as this one; this gives the homogeneous customer-provider cycle.

2. Suppose we have a rim edge $e = \{v,\ u\}$ where $C^v(u) = C_2$. Just as with the case above, only M_{21} permits equality, and by a similar argument, the adjacent rim edge to v must be a customer-provider edge. But this results in case (1) above where the dispute wheel must have these edges all the way around the rim, which contradicts the assumption that $C^v(u) = C_2$. Thus this edge e cannot be on a dispute wheel.

3. Suppose we have a rim edge $e = \{v,\ u\}$ where $C^v(u) = C_1$. All values M_{1i} permit equality, so v can import the dispute path descriptors from neighbors of any class. Consider the assigment along the rim edge $e' = \{w,\ v\}$ adjacent to v. By case (2) above, $C^w(v) \neq C_2$. If $C^w(v) = C_3$ then all dispute wheel edges must have this directed assignment, as in case (1) above, so this contradicts the assumed class assignment along edge e. The only other possibility is that $C^w(v) = C_1$, which would give a customer-provider cycle.

Checking for these customer-provider cycles is tractable; even without an explicit check, the basic economics of the current commercial Internet naturally prevent nodes from being customers or providers of themselves.

This constraint is unreasonably strong because it precludes many robust instances; in particular, the role of condition (C2) is ignored, and that could break dispute wheels in some instances. Relying on condition (C2)—tweaking preferences locally—is not enough, however; *e.g.*, if the class assignment to both incoming edges is the same, no system-wide rule can prevent the dispute. We now give four negative results in this regard: each shows the existence of a simple instance containing a dispute wheel, given a certain combination of entries for just one or two classes in the matrices X and M. These are canonical instances that can be generalized. All but one of the cases do not require specific values in matrix W for the construction; this suggests that while Theorem 6.2.8 is too strong, some constraint preventing pairs of class assignments will be necessary.

First we introduce some notation: because matrix M involves the scoping rule between an import and export, the class assignments used to look up values in M are not all in the same direction (that of signaling). We define a matrix S incorporating the matrices M and X: entry $S_{ij} = 1$ if a descriptor can be exported along two signaling edges, first of class C_i, then of class C_j, without increasing the level attribute. Equivalently, we may define S in terms of X and $\hat{M}$ as follows.

DEFINITION 6.2.10. $S = X\hat{M}$ (boolean), *i.e.*, $S_{ij} = \min((X\hat{M})_{ij}, 1) \in \{0, 1\}$.

We now proceed to the negative results. The first negative result involves the existence of dispute wheels with only one type of class assignment on the rim. One of the cases requires certain values in W for the construction; the other case does not.

PROPOSITION 6.2.11. *Suppose $C_\alpha \in C$. If either*

1. $S_{\alpha\alpha} = 1$*, or*

2. *there exists some $C_\beta \in \mathbf{m}^{-1}(C_\alpha)$ such that $\hat{W}_{\beta\gamma} \neq -1$ for some $C_\gamma \in \mathbf{x}(C_\alpha)$,*

then there exists an instance that contains a homogeneous dispute wheel of type C_α.

Proof. For case 1, we can construct a simple dispute wheel where all spokes and rim segments have length one, and for any of these edges e, $\mathbf{c}(e) = C_\alpha$ and $\mathbf{c}(e') = C_\beta$ for the same class $C_\beta \in \mathbf{x}(C_\alpha)$. Then, regardless of $\hat{W}$, it is possible for every rim node to export a spoke descriptor to its neighbor because $S_{\alpha\alpha} = 1$ and for every rim node to prefer the neighbor's path because the spoke and rim neighbors are assigned the same class and we must have $\hat{W}_{\beta\beta} \neq -1$.

For case 2, we can construct a similar dispute wheel, but while $\mathbf{c}(e_R) = C_\alpha$ for rim edges e_R, we have that $\mathbf{c}(e'_Q) = C_\beta$ for the reverse spoke edges e'_Q. Let $\mathbf{c}(e'_R) = C_\gamma$ for γ

as defined in the proposition statement. Then, regardless of $\hat{W}$, it is possible for every rim node to export a spoke descriptor to its neighbor because $C_\beta \in \mathbf{m}^{-1}(C_\alpha)$ and for every rim node to prefer the neighbor's path because $\hat{W}_{\beta\gamma} \neq -1$. □

EXAMPLE 6.2.12. For the system in Example 6.1.5, we have

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

so homogeneous dispute wheels of type customer or provider are possible. Again, this shows that instances without customer-provider cycles are robust.

The next three results show how to construct dispute wheels with two types of class assignments on the rim, regardless of W. None of these results require particular values in W for the existence of the dispute wheels.

PROPOSITION 6.2.13. *If there exist $C_\alpha, C_\beta \in C$ such that $S_{\alpha\beta} = S_{\beta\alpha} = 1$, then there exists an instance containing a heterogeneous dispute wheel of types C_α, C_β.*

Proof. We can create a dispute wheel with origin v_0 and two rim nodes v_1 and v_2. Let $C^{v_0}(v_1) = C_\alpha$, $C^{v_0}(v_2) = C_\beta$, $C^{v_1}(v_2) = C_\beta$, and $C^{v_2}(v_1) = C_\alpha$. Then it is possible for v_1 to assign the same class in $\mathbf{x}(C_\alpha)$ to v_0 and v_2 and prefer the rim path over the spoke path; similarly for v_2 with $\mathbf{x}(C_\beta)$. The export from spoke to rim is possible precisely because $S_{\alpha\beta} = S_{\beta\alpha} = 1$, and no setting in $\hat{W}$ can break this dispute wheel. □

PROPOSITION 6.2.14. *Suppose X is not a permutation matrix. If there exist $C_\alpha, C_\beta \in C$ such that $S_{\alpha\alpha} = 1$, $S_{\beta\beta} = 1$, and $\mathbf{x}(C_\alpha) \cap \mathbf{x}(C_\beta) \neq \emptyset$, then there exists an instance containing a heterogeneous dispute wheel of types C_α, C_β.*

Proof. We can create a dispute wheel with origin v_0 and two rim nodes v_1 and v_2. Let $C^{v_0}(v_1) = C_\alpha$, $C^{v_0}(v_2) = C_\beta$, $C^{v_1}(v_2) = C_\alpha$, and $C^{v_2}(v_1) = C_\beta$. Then because $\mathbf{x}(C_\alpha) \cap$

$\mathbf{x}(C_\beta) \neq \emptyset$, it is possible for v_1 and v_2 to assign the same class $C_\gamma \in \mathbf{x}(C_\alpha) \cap \mathbf{x}(C_\beta)$ to v_0 and the rim neighbor. Because $\hat{W}_{\gamma\gamma} \neq -1$, both can prefer the rim path over the spoke path. The export from spoke to rim is possible precisely because $S_{\alpha\alpha} = S_{\beta\beta} = 1$, and no setting in $\hat{W}$ can break this dispute wheel. □

PROPOSITION 6.2.15. *If there exist $C_\alpha, C_\beta \in C$ such that $\exists\, C_\gamma \in \mathbf{x}(C_\alpha)$ with $\hat{M}_{\beta\gamma} = 1$ and $\exists\, C_\psi \in \mathbf{x}(C_\beta)$ with $\hat{M}_{\alpha\psi} = 1$, then there exists an instance containing a heterogeneous dispute wheel of types C_γ, C_ψ.*

Proof. We can create a dispute wheel with origin v_0 and two rim nodes v_1 and v_2. Let $C^{v_1}(v_0) = C_\alpha, C^{v_2}(v_0) = C_\beta, C^{v_1}(v_2) = C_\alpha$, and $C^{v_2}(v_1) = C_\beta$. These assignments are in the *reverse direction* of export along the wheel. But, given the statement of the proposition, we can set $C^{v_0}(v_1) = C_\gamma$, $C^{v_0}(v_2) = C_\psi$, $C^{v_2}(v_1) = C_\gamma$, and $C^{v_1}(v_2) = C_\psi$ and obtain a heterogeneous dispute wheel of types C_γ, C_ψ similar to the dispute wheel constructed in the proof of Proposition 6.2.13. □

The constructions in the above proofs can be extended to larger dispute wheels, possibly involving more class types, as long as the X-, M-, and, in some cases, W-matrix entries permit. We now construct a predicate $\mathbb{P}$ on classes which generalizes conditions (C1)–(C3). $\mathbb{P}(C_i, C_j)$ will be true exactly when nodes u, v, and w may be part of a dispute wheel in which v is a rim node, v imports from u and exports to w, and $C^v(u) = C_i$ and $C^v(w) = C_j$. Note that from the proofs above, we see that just two permitted rim nodes are necessary for some instance to contain a dispute wheel.

DEFINITION 6.2.16. Let $\mathbb{P}$ be a predicate on two classes

$$\mathbb{P}(C_\alpha, C_\beta) \iff \Big((M_{\alpha\beta} = 1) \vee \Big(\exists C_\gamma \in \mathbf{m}^{-1}(C_\beta) : \hat{W}_{\gamma\alpha} \neq -1 \Big) \Big) .$$

The claim that $\mathbb{P}$ is the condition we want is supported by the following theorem. It states that a cycle where $\mathbb{P}$ holds pairwise along the edges could be a dispute wheel; and, if all cycles where $\mathbb{P}$ holds pairwise along the edges are prevented, no dispute wheel can exist.

THEOREM 6.2.17. *For $1 \leq i \leq n$, let $k_i, k'_i \in \{1, \ldots, c\}$ so that $C_{k_i}, C_{k_i'} \in C$; let $k_{n+1} = k_1$. There exists an instance containing a dispute wheel with rim cycle $e_1, e_2, \ldots, e_n$, where e_n is adjacent to e_1, $\mathbf{c}(e_i) = C_{k_i}$, and $\mathbf{c}(e'_i) = C_{k'_i}$, iff $\forall_{1\leq i\leq n}$, $\mathbb{P}(C_{k'_i}, C_{k_{i+1}})$ is true.*

Proof. We first prove the forward "only-if" direction. Assume there exists an instance containing such a dispute wheel. Let v_i be the node incident to edges e_i and e_{i+1}; its class assignments for the neighbor incident to e_i and e_{i+1} are then $\mathbf{c}(e'_i) = C_{k'_i}$ and $\mathbf{c}(e_{i+1}) = C_{k_{i+1}}$, respectively. Every rim node v_i is either active (one with a direct spoke path) or inactive (within a rim segment). If it is inactive, then some path descriptor imported along e_i must be exported along e_{i+1} without an increase in level (by Proposition 6.2.3); thus, $M_{k'_i k_{i+1}} = 1$, which implies $\mathbb{P}(C_{k'_i}, C_{k_{i+1}})$. If it is active, then the imported spoke path is exported along e_{i+1} without increasing level (by Proposition 6.2.3); thus, it must assign the neighboring spoke node w a class $C^{v_i}(w) = C_\gamma$ such that

$$M_{\gamma k_{i+1}} = 1. \tag{6.3}$$

Furthermore, the descriptor imported along e_i must be preferred more than the spoke path; thus, it must be that

$$\hat{W}_{\gamma k_{i'}} \neq -1 \tag{6.4}$$

so that the rim preference is allowed. (6.3) and (6.4) together imply $\mathbb{P}(C_{k'_i}, C_{k_{i+1}})$. By considering every node v_i in this way, we see that $\mathbb{P}(C_{k'_i}, C_{k_{i+1}})$ must hold for all i.

In the other direction, we construct the specified dispute wheel if $\mathbb{P}(C_{k'_i}, C_{k_{i+1}})$ holds for all i. Build a cycle of edges $e_1, e_2, \ldots, e_n$, with e_n adjacent to e_1, and assign $\mathbf{c}(e_i) = C_{k_i}$,

$\mathbf{c}(e'_i) = C_{k'_i}$. Assume there is a destination node d. As $\mathbb{P}(C_{k'_i}, C_{k_{i+1}})$ holds, either $M_{k'_i k_{i+1}} = 1$ or

$$\exists C_\gamma \in \mathbf{m}^{-1}(C_{k_{i+1}}) : \hat{W}_{\gamma k'_i} \neq -1. \tag{6.5}$$

First assume that $M_{k'_i k_{i+1}} = 1$; in this case, the node between e_i and e_{i+1} can be left an inactive node. If (6.5) is true, then the node v_i between e_i and e_{i+1} can be made an active node; in this case, connect the destination node d to v_i and let $C^{v_i}(d) = C_\gamma$ such that C_γ satisfies (6.5). Then let node v_i prefer the route imported from the rim along e_i over the route from the spoke along (d, v_i). This is permitted because $\hat{W}_{\gamma k'_i} \neq -1$ by (6.5). We note that we can choose at least two nodes v_i to be active nodes—the minimum required for a dispute wheel—because even if $M_{k'_i k_{i+1}} = 1$, then we can set v_i to be an active node connected to d with $\mathbf{c}(v_i, d) = \mathbf{c}(e'_i)$. If both descriptors imported at v_i are from neighbors of the same class, then the rim path can be preferred over the spoke path, which is enough to cause the dispute. Therefore, this cycle of edges $e_1, e_2, \ldots, e_n$ with destination d and class assignments set as indicated forms a dispute wheel that is permitted by the class description. □

Theorem 6.2.17 identifies our robustness constraint exactly: to prevent dispute wheels in any network, we must check against all cycles where $\mathbb{P}$ holds on all pairs of edges in the cycle; if these cycles are permitted, they are *potential dispute wheels*. Formally, we have the following.

CONSTRAINT 6.2.18. *For all cycles of signaling edges $e_1, e_2, \ldots, e_n$, there exists some i, $1 \leq i \leq n$ such that $\mathbb{P}\left(\mathbf{c}(e'_i), \mathbf{c}(e_{i+1})\right)$ does not hold (assume that $e_{n+1} = e_1$).*

By Theorem 3.2.9, instances obeying this constraint are robust. Because the presence of a dispute wheel does not preclude solvability, we cannot say that this constraint is tight.

ALGORITHM 6.2.19. Given a class description, we can find the "troublesome" pairs of assignments satisfying $\mathbb{P}$ in $O(c^3)$ time, where c is the number of classes, using the following naïve procedure:

1. For all $1 \leq i, j \leq c$, examine $\hat{W}_{ij}$. If $\hat{W}_{ij} \neq -1$, create a list T' of classes in $\mathbf{m}(C_i)$. Add the pair (j, t) for all $t \in T'$ to the list T.

2. For all $1 \leq i, j \leq c$, examine M_{ij}. If $M_{ij} = 1$ then add the pair (i, j) to T.

Note that for any pair $(t_1, t_2) \in T$, $\mathbb{P}(C_{t_1}, C_{t_2})$ holds.

6.3 CENTRALIZED DISPUTE-WHEEL PREVENTION

Although we can now identify the constraint for a given class description, we have not yet mentioned how to enforce this condition. In this section, we describe a centralized algorithm that operates on an instance graph and detects violations of Constraint 6.2.18.

6.3.1 CYCLE-DETECTION ALGORITHM

Given an instance with undirected network $G = (V, E)$ and nodes' class assignments, we want to identify all cycles in which $\mathbb{P}$ holds on all consecutive pairs of edges around the cycle. We do this as follows.

ALGORITHM 6.3.1.

1. Construct a digraph $G_S = (V, E_S)$ using the same vertices as in the network G. For every edge in $\{u, v\} \in E$, E_S contains the directed edges (u, v) and (v, u).

2. Construct a new digraph $G_L = (V_L, E_L)$ from G_S as follows. Let $V_L = E_S$. E_L contains an edge from (u, v) to (w, x) iff $v = w$ and $\mathbb{P}(\mathbf{c}(v, u), \mathbf{c}(v, x))$ holds.

3. Do a directed depth-first search of G_L. Any directed cycles found correspond to potential dispute wheels.

The following proposition asserts the correctness of the algorithm.

PROPOSITION 6.3.2. *Any cycle in the graph G_L generated by Algorithm 6.3.1 corresponds to a potential dispute wheel in the original network, and every dispute wheel in the network produces a directed cycle in G_L.*

Proof. Let $e_0, e_1, \ldots, e_k$ be a directed cycle in G_L, with $e_i = ((u_i, v_i), (u_{i+1}, v_{i+1}))$ for $u_i, v_i \in V$ and $v_i = u_{i+1}$ for $0 \leq i \leq k$ (subscripts modulo $k + 1$). By construction of G_L, $\mathbb{P}(\mathbf{c}(v_i, u_i), \mathbf{c}(v_i, v_{i+1}))$ holds for every i, so the cycle $\{u_0, v_0\}, \{u_1, v_1\}, \ldots, \{u_k, v_k\}$ violates Constraint 6.2.18 (*i.e.*, it is a potential dispute wheel).

By Proposition 6.2.3, the rim $\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_k, v_0\}$ of a dispute wheel in G is such that $\mathbb{P}(\mathbf{c}(v_i, v_{i-1}), \mathbf{c}(v_i, v_{i+1}))$ holds for all i. Thus E_L contains $((v_{i-1}, v_i), (v_i, v_{i+1}))$ for every i, producing a directed cycle in G_L. $\square$

The following proposition gives an upper bound on the running time of the algorithm.

PROPOSITION 6.3.3. *Algorithm 6.3.1 has running time $O(\Delta|E|)$ on a network $G = (V, E)$ with maximum vertex degree Δ.*

Proof. Construction of G_S takes $O(|V| + |E|)$ time, construction of G_L takes $O(|V_L| + |E_L|) = O(|E| + \Delta|E|)$ time, and cycle checking G_L by depth-first search takes $O(|V_L| + |E_L|) = O(|E| + \Delta|E|)$ time. Therefore, the total running time is $O(\Delta|E|)$. $\square$

6.3.2 CHECKING NEXT-HOP PREFERENCES

Suppose we have a network running a path-vector protocol for which each node v specifies a partial order $\trianglelefteq$ on neighbors such that for two neighbors u and w, if $u \triangleleft w$, then routes imported from u must be ranked lower (*i.e.*, more preferred) than routes imported from w, and if $u = w$, then no relative preference is forced between routes imported from u and w. Furthermore, we allow nodes to describe scoping rules for these neighbors (under what conditions, if any, routes can be exported). These policies are called *next-hop preferences*, because the relative preference and scoping rules for routes are determined by the next hop along the path, *i.e.*, the neighbor from which the path descriptor is imported.

Given a network and next-hop preferences, we can construct a class-based system consistent with the nodes' relative preference and scoping rules. Define a class description in which there is a class for every directed signaling edge in the network, and assign every neighbor the class corresponding to the edge between the node and that neighbor. Then set the entries of W to be consistent with the partial order for next-hop preferences, and set the entries of M to be consistent with the scoping rules. Most of the entries of W and M are irrelevant because not all edges in the graph are adjacent, so comparisons will never have to be made between all possible pairs of classes. Essentially, each node's next-hop preferences define sub-matrices of W and M.

By creating this consistent class-based system, we can use the robustness checks developed in this paper to see whether this network with its next-hop preferences has any potential dispute wheels. Because there is a class for every directed edge in the signaling graph, there will be $c = 2|E|$ classes, and generation of the constraint pairs satisfying $\mathbb{P}$ using Algorithm 6.2.19 will take $O(c^3) = O(|E|^3)$ time. However, not all pairs of classes correspond to adjacent edges; assuming that Δ is the maximum degree of any vertex in the graph, each node in V gives less than Δ^2 pairs of directed signaling edges. For each pair we must check one entry in M and at most Δ in W, so $\mathbb{P}$ may actually be generated in $O(\Delta^3|V|)$ time. Running the centralized cycle-check (Algorithm 6.3.1) takes $O(\Delta|E|)$

time. The total time is thus $O(\Delta^3|V| + \Delta|E|)$, but because $|E| = O(\Delta|V|)$, the running time is simply $O(\Delta^3|V|)$.

Below we give the full algorithm that executes the constraint-generating and centralized constraint-checking procedures with the running time discussed above for a network $G = (V,\ E)$ with next-hop preferences (relative preference and scoping rules at each node for all neighbors of that node).

ALGORITHM 6.3.4.

1. Construct a set $T \subset V^3$. For every node $v \in G$, repeat the following for each neighbor x of v: For all neighbors $u \neq x$ of v, if a descriptor at v imported from u can be exported to x (without level increase), then:

   (a) add the triple $(v,\ u,\ x)$ to the set T; and

   (b) for all neighbors w such that $w \trianglelefteq u$, add the triple $(v,\ w,\ x)$ to the set T.

   After iterating through all nodes v and pairs of neighbors x and u, note that elements of the set T correspond to pairs of class assignments satisfying the predicate $\mathbb{P}$ in the next-hop-preferences sense.

2. Construct a new directed graph $G_L = (V_L,\ E_L)$. The vertex set

$$V_L = \{(u,\ v) \text{ and } (v,\ u) \mid \{u,\ v\} \in E\}\,;$$

   i.e., there is one vertex for every directed signaling edge. The edge set

$$E_L = \{((u,\ v),\ (v,\ x)) \mid (v,\ u,\ x) \in T\}\,;$$

   i.e., there is a directed edge from $(u,\ v)$ to $(v,\ w)$ iff these two signaling edges can be adjacent on a dispute-wheel rim.

3. Cycle-check G_L using directed depth-first search. Any directed cycles found correspond to potential dispute wheels.

PROPOSITION 6.3.5. *Any directed cycle in the graph G_L generated by Algorithm 6.3.4 corresponds to a potential dispute wheel in the original network, and every dispute wheel in the network produces a directed cycle in G_L.*

Proof. The graph G_L checked for cycles in Algorithm 6.3.4 is similar to the graph G_L checked for cycles in Algorithm 6.3.1; the difference is that the edges are based on next-hop-preference conditions instead of a predefined $\hat{W}$ and $\hat{M}$. Therefore, this result follows from Proposition 6.3.2 if we show that $(v,\ u,\ x) \in T$ iff $\mathbb{P}(\mathbf{c}(v,u),\ \mathbf{c}(v,x))$ would hold for the $\hat{W}$ and $\hat{M}$ created from the next-hop preferences.

However, this is simple to derive from the construction of T in step 1 of Algorithm 6.3.4: $(v,\ u,\ x) \in T$ iff (1) either an import over $(u,\ v)$ can be exported over $(v,\ x)$ without level increase, thus $\hat{M}_{\mathbf{c}(v,u)\ \mathbf{c}(v,x)} = 1$; or (2) there exists some neighbor y of v such that $u \trianglelefteq y$ and a descriptor learned from y can be exported to x without a level increase; thus $\exists\, y \in \mathbf{m}^{-1}(\mathbf{c}(v,x)) : \hat{W}_{\mathbf{c}(v,y)\ \mathbf{c}(v,u)} \neq -1$. These conditions are exactly equivalent to $\mathbb{P}(\mathbf{c}(v,u),\ \mathbf{c}(v,x))$. $\square$

6.3.3 ALGORITHMS IN PREVIOUS WORK

Section 6.3 in [SOB03] gives a check for protocol convergence on a network given a path-vector algebra—much like our centralized algorithm given the constraint generated from the class description. Translated to the class-based framework, the class-aware constraint-generating algorithm and convergence-checking algorithm from [SOB03] take time $O(c^3)$ and $O(c \cdot (|V| + |E|))$, respectively, where c is the number of classes, assuming that matrix W is consistent with a linear order on C. The performance of this algorithm on a network compared to Algorithm 6.3.1 will depend on how the number of classes c compares to the vertex degrees in a network and how sparse the network is. Also note that the algorithm from the algebra framework might give some false positives: it identifies some cycles as troublesome that are not actually potential dispute wheels (*i.e.*, the constraint checked is stronger than necessary and stronger than Constraint 6.2.18). We discuss this difference below and show how to implement this stronger, but sometimes faster, convergence check in our framework.

In the algebra framework, routing policy is captured by the assignment of a *label* to each edge in the signaling graph. Changes to the attributes of a path descriptor when it is shared between neighboring nodes are modeled by an operation that depends on (1) the label on the signaling edge between the neighbors, and (2) the *signature* of the path before it is shared—this corresponds to the original path descriptor; the result is a new signature, or path descriptor, that can be ranked at the importing node. Because the algebra framework does not separately model import and export transformations, the label assigned to the edge must capture the policy decisions made at both import and export. In the class-based framework, policy decisions depend on class assignments between neighbors. Therefore, when using the algebra framework to model a class-based system, the labels on signaling edges must capture the class assignment in both directions—both nodes' view of the other node—in order to capture the relative-preference and scoping rules that eventually affect the rank or availability of path descriptors. Thus, if c is the number of classes, there are c^2 possible labels.

The robustness algorithm in [SOB03] first generates a set of labels with which to check cycles (this corresponds to the generation of our constraint involving pairs of classes): it identifies sets of labels L_w that come from pairs of labels satisfying an equivalent notion of $\mathbb{P}$. (The variable w indexes these sets, ordering them based on the relative-preference order of the labels falling into the sets.) The algorithm then uses these sets to check the *freeness* constraint from Definition 5.1.7: it checks the graph for cycles formed by edges whose labels all belong to one of these sets L_w (for some value w). Because the sets L_w are missing information that could be used to detect that some cycles would not actually be dispute wheels, some instances are flagged as problematic even though they are robust.

The following scenario causes the algorithm to produce a false positive. We will refer to labels with a pair of classes indicating the class assignments in both directions along a signaling edge. Suppose that $(C_\alpha, C_{\alpha'})$ and $(C_\beta, C_{\beta'})$ is the only pair of labels involving $(C_\beta, C_{\beta'})$ that satisfies the equivalent notion of $\mathbb{P}$ (*i.e.*, this pair of edge types could be on a dispute-wheel rim). The algorithm in [SOB03] will add $(C_\beta, C_{\beta'})$ to the appropriate set L_w. Suppose that the label $(C_\gamma, C_{\gamma'})$ also belongs to L_w because it, too, is part of a pair of labels satisfying the equivalent notion of $\mathbb{P}$. The algorithm would then remove all cycles in which all edge labels belong to L_w. Consider such a cycle with two adjacent edges labeled $(C_\gamma, C_{\gamma'})$ and $(C_\beta, C_{\beta'})$. It may be the case that $X_{\gamma\alpha'} \neq 1$, *i.e.*, these edges could not actually participate in a dispute because doing so would violate class-consistency. The storage of labels in L_w basically throws away one half of the pair satisfying $\mathbb{P}$, in this case, the label $(C_\alpha, C_{\alpha'})$. As a result, cycles that could not have class consistency on the overlapping edges and be dispute cycles at the same time are still flagged.

Below, we describe the translated algorithm to check the algebraic robustness constraint and give bounds for the running time of each step.

ALGORITHM 6.3.6. Steps 1–3 generate the algebraic robustness constraint, the sets L_w; steps 4–5 check that constraint.

1. By assumption, W is consistent with a linear order $<_C$ on the classes C. To each

class C_i assign a value $w(C_i) \in \{1, \ldots, c\}$ such that $w(C_i) < w(C_j)$ if $C_i <_C C_j$. Let $w^* = \max_i w(C_i)$; thus $w^* = O(c)$.

2. Use Algorithm 6.2.19 to generate pairs of classes on which $\mathbb{P}$ holds. This takes $O(c^3)$ time.

3. For each w, $1 \le w \le w^*$, construct the set of labels

$$L_w = \{(C_\beta, C_{\beta'}) \in C \times C \mid \exists\, C_\alpha \in C : \\ \mathbb{P}(C_\alpha, C_\beta) \wedge X_{\beta\beta'} = 1 \wedge w(C_{\beta'}) = w\}.$$

   This takes $O(c^3)$ time because the sets can be built by examining the $O(c^2)$ pairs (C_α, C_β) satisfying $\mathbb{P}$ and, for each, examining the $O(c)$ classes $C_{\beta'}$ so that if $X_{\beta\beta'} = 1$, $(C_\beta, C_{\beta'})$ is added to the set $L_{w(C_{\beta'})}$.

4. Given the network $G = (V,\ E)$, construct the graph $G_S = (V,\ E_S)$ as in Algorithm 6.3.1. Then for each w, $1 \le w \le w^*$, construct the graph $G_S[w] = (V,\ E_w)$ where $e \in E_w$ iff $\mathbf{c}(e) = C_\beta$, $\mathbf{c}(e') = C_{\beta'}$ such that $(C_\beta, C_{\beta'}) \in L_w$. This takes $O(c|E|)$ time, total.

5. Cycle-check each $G_S[w]$; any cycle is a potential dispute wheel. This takes $O(c\cdot(|V| + |E|))$ time by depth-first search.

6.4 DISTRIBUTED DISPUTE-WHEEL PREVENTION

Although the Internet graph and node relationships do not change haphazardly, a centralized algorithm running on a snapshot of the Internet graph is still somewhat infeasible: A central source would need to collect information about the network topology as well as, in a potentially harder and/or privacy-invading task, information about node relationships throughout the network. In this section, we first present a distributed algorithm for detecting potential dispute wheels and then contrast this algorithm with one given in earlier work.

6.4.1 DISTRIBUTED CYCLE-CHECK

Our distributed algorithm (Algorithm 6.4.1) detects potential dispute wheels that include a specified edge on their rim. The algorithm is administered by the two nodes connected by the edge in question; it sends at most three messages across each edge in the graph and does not require that the graph, minus the edge in question, is dispute-wheel-free. Furthermore, the algorithm reveals little about the relationships between nodes in the graph – a node *may* learn possibilities for its neighbors' relationships with other nodes, but nothing about other relationships in the graph.

If the algorithm detects the edge as problematic, either the edge can be removed from the signaling graph (*i.e.*, the edge is not used to advertise routes) or some tweaks to local policy can be applied to prevent a dispute wheel. These tweaks are included in Algorithm 6.4.1 and allow the edge to exist as-is for the purpose of signaling routes that would never cause a dispute. This algorithm could be run, *e.g.*, by two nodes before adding a signaling link to the Internet graph to see what policy tweaks must be enforced to prevent route oscillations.

In summary, node u starts the algorithm by sending out a forward token $[N, F]$ to v. N is a nonce, which prevents interference between parallel executions of the algorithm, and F indicates that this is a (forward) token. Any node w along the way, including v, that receives this token from some node x passes a copy of the token along to a neighbor y if $\mathbb{P}(\mathbf{c}(w, x), \mathbf{c}(w, y))$ holds and w has not already forwarded the token to y. In this way, the token traverses all pairs of edges that could be part of a potential dispute wheel. If a cycle of edges is traversed, *i.e.*, u receives its starting token $[N, F]$ and would forward it to v, then u knows that the edge (u, v) participates in a potential dispute-wheel rim. Token-traversal paths end when there are no neighbors y that should be forwarded the token; in that case, a receipt, or "backwards" message, $[N, B]$ is sent to the neighbor from whom the token was received. If a node w receives receipts from all neighbors to whom it forwarded the token, w then sends a receipt to the neighbor from whom it received the token. Note that only one forward token needs to be sent along any edge; any duplicate tokens sent along an edge will take the same route as the original token, and this has no effect on cycle detection from u's perspective. We thus know that all token-traversal paths will terminate—in the worst case, after the token has traversed every edge once. The algorithm essentially ends when u receives a receipt from v, indicating that all token traversals have ended. Node u then sends out an all-clear message $[N, C]$, which gets forwarded along the token-traversal paths, so that other nodes can delete any data structures used for that instance of the algorithm. Once the algorithm has ended, if u detected a problem, then u can either refuse to signal along (u, v) or tweak policies so that a dispute wheel could never form along the cycle.

ALGORITHM 6.4.1. A node u should start the following procedure to check the signaling edge (u, v); when checking the network edge $\{u, v\}$, v should separately check the signaling edge (v, u) in the opposite direction. Assume that nodes have a list L_Q for storing nonces from different, parallel executions of this algorithm. Let the in-neighbors of v be denoted $\mathrm{in}(v)$ and the out-neighbors be denoted $\mathrm{out}(v)$.

For node u:

1. Choose and store a nonce N. Create an empty list of nodes L_B.

2. If $\exists\, w \in \mathrm{in}(u)$ such that $\mathbb{P}(\mathbf{c}(u, w),\ \mathbf{c}(u, v))$ holds, then u sends the message $[N, F]$ to v along (u, v).

3. Whenever u receives $[N, F]$ from $w \in \text{in}(u)$, send the message $[N, B]$ to w. If $\mathbb{P}(\mathbf{c}(u, w),\ \mathbf{c}(u, v))$ holds, then add the node w to the list L_B.

4. When u receives the message $[N, B]$ from v, u should send $[N, C]$ to v. Node u may now start routing along (u, v) after applying the appropriate policy-tweak rules below to nodes in list L_B.

5. Node u should ignore any $[N, C]$ messages.

For all nodes $w \neq u$:

1. If w receives the message $[N, F]$ from $x \in \text{in}(w)$:

   (a) If $N \notin L_Q$, add N to list L_Q and create an array A_N of lists of type $(V \times \{0, 1\})$ indexed by the elements of $\text{in}(w)$.

   (b) For each $y \in \text{out}(w)$, if $\mathbb{P}(\mathbf{c}(w, x),\ \mathbf{c}(w, y))$ and $(y, 0),\ (y, 1) \notin A_N(z)$ for all $z \in \text{in}(w)$, then send $[N, F]$ to y and add $(y, 0)$ to $A_N(x)$.

   (c) If no $[N, F]$ messages were sent above in step (b), then send $[N, B]$ to x.

2. If w receives $[N, C]$ and $N \in L_Q$, then for each $z \in \text{in}(w)$, send $[N, C]$ to each y such that $(y, 1) \in A_N(z)$. Delete A_N and remove N from L_Q.

3. If w receives $[N, B]$ from $y \in \text{out}(w)$, then replace $(y, 0)$ with $(y, 1)$ in A_N. If $((y, i) \in A_N(x) \Rightarrow (i = 1))$, *i.e.*, if y is the last node in the list $A_N(x)$ from which $[N, B]$ was received for some $x \in \text{in}(w)$, then send $[N, B]$ to x.

The following are the policy-tweak rules for u if, at the end of the algorithm, L_B is not empty. Let

$$Y^u(w) = \{y \in \text{in}(u) \mid C^u(y) \in \mathbf{m}^{-1}(C^u(v)) \ \wedge\ \hat{W}_{C^u(y)C^u(w)} \neq -1\,\}.$$

Node u should depreference routes from $w \in L_B$ with respect to all $y \in Y^u(w)$. This is only possible iff

$$\nexists(w, w') : (w' \in Y^u(w)) \wedge (w \in Y^u(w')) \tag{6.6}$$

because two neighbors cannot both be depreferenced with respect to each other. If (6.6) does not hold, then:

1. Pick some $w \in L_B : Y^u(w) \neq \emptyset$. For all $w' \neq w$, if $Y^u(w') \neq \emptyset$, then increase the level attribute on import from w' and remove w' from L_B.
2. Depreference w with respect to all other $y \in Y^u(w)$.
3. For all $w \in L_B$ remaining, increase the level on routes imported from w when exported to v.

The following propositions assert various properties of the algorithm, including correctness.

LEMMA 6.4.2. *In Algorithm 6.4.1, at most one* $[N, F]$ *token is sent along each signaling edge.*

Proof. For every node $w \neq u$, an $[N, F]$ message is only sent to a neighbor if an $[N, F]$ message is received, but the $[N, F]$ message is not sent if the neighbor has already received an $[N, F]$ message from w, regardless of how many $[N, F]$ messages are received at w. Because this process starts with u sending one $[N, F]$ message to v, it is clear that at most one $[N, F]$ message is sent between every pair of nodes. □

PROPOSITION 6.4.3. *The algorithm terminates, i.e., every node that sends to* y *or receives from* x *a message* $[N, F]$ *receives from* y *or sends to* x*, respectively, a message* $[N, B]$.

Proof. We must show that if the $[N, F]$ token is sent along some edge $(x,\ y)$, then an $[N, B]$ receipt is sent back from y to x. Consider a graph $G_T = (V_T,\ E_T)$ constructed from the target network $G = (V,\ E)$. The vertex set V_T is the set of directed signaling edges of G, and there is an edge in E_T from $(x,\ y)$ to $(y,\ z)$ if the receipt of a token $[N, F]$ at y from x causes y to send the token $[N, F]$ to z.

Note that the connected component of G_T is a tree rooted at $(u,\ v)$; this is because Lem. 6.4.2 tells us that an $[N, F]$ token is sent at most once along a signaling edge, and we can see from the algorithm that an $[N, F]$ token is only sent by a node after receiving one itself. Therefore every node in G_T has either no ancestors (if the node is $(u,\ v)$ or $[N, F]$ is never sent along the edge) or exactly one ancestor.

First consider the leaf nodes of the tree; these are edges $(x,\ y)$ such that receipt of the $[N, F]$ token at y does not cause an $[N, F]$ token to be sent. According to the algorithm, this happens in two cases: (1) $y = u$; or (2) there does not exist a neighbor z of y, for which $\mathbb{P}(\mathbf{c}(y, x),\ \mathbf{c}(y, z))$ holds, that has not already received a token. In both of these cases, an $[N, B]$ message is sent back to x from y.

Next consider the ancestors of leaf nodes in the tree. Given the argument above, we know that for such an edge $(x,\ y)$, node y receives an $[N, B]$ message from all neighbors z such that $(y,\ z)$ is a descendant of $(x,\ y)$. According to the algorithm, once this has happened, every entry in the list $A_N(x)$ at y will be of the form $(z, 1)$; thus y will send an $[N, B]$ receipt to x. This argument can be repeated for the ancestors of these nodes, *etc.*, so that all tokens are eventually acknowledged with receipts.

The $[N, C]$ messages terminating the algorithm follow the path of the tree, so that every node that initially received the token will destroy any data associated with the nonce N. □

PROPOSITION 6.4.4. *At the end of the algorithm, if $L_B \neq \emptyset$ at u, then $(u,\ v)$ is part of a cycle violating Constraint 6.2.18.*

Proof. If $L_B \neq \emptyset$, then there exists some $w \in L_B$ such that w sends u the message $[N, F]$.

This means that u originated the message $[N, F]$, sending it to v, and there is a set of nodes $\{v = x_1,\ x_2,\ \ldots,\ x_n = w\}$ such that every node x_i sends $[N, F]$ to x_{i+1} (assume $x_{n+1} = u$). According to the algorithm, this only happens if: (1) $\mathbb{P}(\mathbf{c}(u, w),\ \mathbf{c}(u, v))$ holds; and (2) if $\mathbb{P}(\mathbf{c}(x_i, x_{i-1}),\ \mathbf{c}(x_i, x_{i+1}))$ holds for all $1 \leq i \leq n$. We then have a cycle of edges

$$(u,\ v),\ (v,\ x_2),\ (x_2,\ x_3),\ \ldots,\ (x_{n-1},\ w),\ (w,\ u)$$

where $\mathbb{P}$ holds pairwise along adjacent edges; this is a potential dispute wheel containing $(u,\ v)$. □

PROPOSITION 6.4.5. *The algorithm sends either* 0 *or* 3 *messages per signaling-graph link.*

Proof. By Lemma 6.4.2, at most one $[N, F]$ message is sent along a link. If an $[N, F]$ message is sent, it is clear from the algorithm and the proof of Proposition 6.4.3 that one $[N, B]$ message and one $[N, C]$ message are sent along the link, and that no other messages are generated as a result of the $[N, F]$ message. Therefore, the total number of messages sent along a given link is either 0 (no $[N, F]$ message is sent) or 3 (the messages $[N, F]$, $[N, B]$, and $[N, C]$ are sent). □

Depending on the structure of the graph, a token-traversal path might include all the edges in a graph; but, in this case, this will be the only token-traversal path (because tokens are only sent once per edge).

Algorithm 6.4.1 preserves privacy in the following ways. As the messages involved contain only a nonce and message type, the edge being checked by a run of the algorithm is not revealed to nodes other than u. Furthermore, because Proposition 6.4.3 tells us that every $[N, F]$ message sent is acknowledged with an $[N, B]$-message reply, nothing in the algorithm tells any of the other nodes whether or not a potential dispute wheel has been detected—only u knows this. The only information learned is that if a node w receives an $[N, F]$ message from x, it knows that there exists some neighbor z of x such that $\mathbb{P}(\mathbf{c}(x, z), \mathbf{c}(x, w))$ holds. w might then narrow down the possibilities for the assignments $C^x(z)$ and $C^x(w)$, although the latter is already restricted by $C^w(x)$ and the matrix X. Node w does not learn any other information about nodes' class assignments.

There is an inherent trade-off between the number of messages sent by the algorithm and the state retained at each node. Algorithm 6.4.1 can be modified—without sacrificing privacy—to delete the state for nonce N early, instead of waiting for an $[N, C]$ message. There are two possible modifications:

1. The list $A_N(z)$ can be deleted once an $[N, B]$ message is sent to in-neighbor z; or
2. The array of lists A_N can be deleted once $[N, B]$ messages are sent to all in-neighbors z such that $A_N(z)$ is nonempty, *i.e.*, once all tokens have been acknowledged.

While these modifications eliminate the need for the $[N, C]$ message, they lose some benefit of aggregating token-traversal paths. Consider a node w that has acknowledged all tokens with receipts. Using either modification, w now retains no state for this run of the algorithm. However, it could receive the forward token from a neighbor that had not yet sent it a token (*e.g.*, because of network delays) or from a neighbor that previously sent it a token (*e.g.*, because it deleted state as well). As a result of either of these (and certainly in the latter case), w might send a token to a neighbor it had previously sent a token, thus duplicating a token-traversal path and increasing the total number of messages used by the algorithm. Given the first modification, this could happen even when there are outstanding tokens that have not been acknowledged. Therefore, these modifications may be appropriate in certain networks where sending duplicate tokens is unlikely and in networks where maintaining state or sending the $[N, C]$ message is expensive; however, in most cases, these modifications will result in duplicating messages along token-traversal paths with little added benefit.

6.4.2 ALGORITHMS IN PREVIOUS WORK

The algorithm SPVP_3 in [GW00] is a distributed path-vector routing algorithm that detects local-policy-based routing oscillations while running. SPVP_3 essentially adds a *path-history attribute* to path descriptors: it stores the changes in best-route choices that cause the descriptor to be advertised. If there is a cycle in these changes, then the descriptor being advertised is contributing to a route oscillation due to local policies. These path-history cycles are shown in [GW00] to correspond exactly to dispute wheels. Therefore, in the process of routing, SPVP_3 detects the actual policy conflicts forming a dispute wheel.

The algorithms in this chapter take a different approach – they attempt to detect and/or prevent potential dispute wheels before a routing dispute ever occurs. But more importantly, the idea of including a constraint as part of the system specification allows us to prove properties about the system at design-time. The centralized and distributed algorithms, then, essentially implement constraint enforcement rather than find modified solutions on-the-fly. The class-based path-vector routing protocol will work as expected as long as the system has been designed to prevent bad policy interactions.

One final difference is that SPVP_3 essentially removes the rim paths on a dispute wheel from the choices that *all* rim nodes have for paths to a destination. This is unnecessary, because the dispute wheel only needs to be "broken" at one active node by tweaking preferences. SPVP_3 prevents multiple nodes (rather than one) from being assigned its more-preferred path, whereas our distributed algorithm tweaks policies at one node to correct a potential dispute wheel. It is also worth noting that SPVP_3 requires potentially large routing messages (the length of the path history will be on the order of the size of the dispute-wheel rim). Also, the entire detection process might be repeated for the same cycle and slightly modified spoke paths (or a different destination), whereas the cycle-detection algorithms will detect or fix a potential dispute-wheel rim before it is used for routing, so that this cycle does not cause oscillation for *any* destination or set of spoke paths. The downside of this, however, is that some policy tweaks or filtering might be used to fix the potential dispute-wheel rim even if no route oscillation actually occurs, whereas SPVP_3 deals with the oscillations as they happen dynamically without affecting policies for other routes.

CHAPTER 7

GENERALIZED PATH-VECTOR SYSTEMS AND INDEPENDENT ROUTE RANKING*

Chapters 4–6 ignore the complexities of sharing inter-domain routes within an AS; in particular, the model of the Internet assumed that every AS can be represented by one node in a graph with a single routing policy and a single link to each neighboring node. (In reality, ASes are made up of several routers that maintain BGP sessions to share inter-domain routes; these sessions often connect links to different neighboring ASes and provide multiple inter-connections between the same ASes. See Chapter 2 for more information.) Doing so allowed the development of simple convergence conditions for eBGP.

On the other hand, previous work that included these complexities [BOR+02, GW02A, GW02B] – *e.g.*, violations of independent route ranking, including MEDs in BGP (see Section 2.4) – has been unable to prove general guidelines for convergence analogous to those for eBGP. In this chapter, we bridge this gap by providing a complete, rigorous model that not only permits analysis of the MED attribute but also includes more general route-selection procedures that may violate the independent-route-ranking property. We present an extension of the SPP framework that covers these instances of the inter-domain routing problem and derive a constraint for policy configuration that guarantees robust conver-

*This chapter has appeared previously in joint work with Aaron D. Jaggard [JR05B].

gence; as it is more general, it applies to instances of the limited, original SPP model as well. The generalized SPP is included in the PVPS framework to broaden their application and to give a much cleaner measure for protocol expressiveness.

7.1 A GENERALIZED FRAMEWORK FOR INTER-DOMAIN ROUTING

We first define general route-selection functions and independent route ranking (IRR), explaining the difference between these definitions of route selection and the more specific definitions used in the original SPP and PVPS frameworks. We then present the Generalized Stable Paths Problem (GSPP) and the Generalized Path-Vector Policy System (GPVPS), both of which incorporate the generalized version of route selection. In doing so, we provide an example GSPP demonstrating a MED-induced oscillation.

7.1.1 ROUTE-SELECTION FUNCTIONS AND INDEPENDENT ROUTE RANKING

We begin with a general model of selecting best routes from a routing table.

DEFINITION 7.1.1. A *route-selection function* σ_v maps a set of paths R to a set $S \subseteq R$ that is a set of "best" routes at node v; we write $\sigma_v(R) = S$. When we restrict the selection to a particular destination, we will write $\sigma_v^d(R) = S^d$ such that all paths S^d have destination d.

In most cases, including BGP, $|\sigma^d(R)| \leq 1$ for a set of paths R and some destination d (*i.e.*, for each destination, at most one best path is chosen). Furthermore, we assume that choosing some permitted path is preferred to choosing no path, although some paths are filtered by local policy such that they are never considered as part of the selection process. Assuming that these filtered paths are not stored in the routing table R, then for all $R^d \subset R$ to a particular destination d, $R^d \neq \emptyset$ implies $\sigma^d(R^d) \neq \emptyset$.

Independent route ranking (IRR) means that the preference of a path relative to other paths depends only on that path alone (and any information in that path's routing-table entry) and not knowledge of other paths.

DEFINITION 7.1.2. A selection function σ obeys *Independent Route Ranking* iff, for all sets of routes R_1 and R_2 and destinations d, the following two conditions hold:

1. $\sigma^d(R_1) = S$ implies $\sigma^d(R_1 \cup R_2) \cap (R_1 \setminus S) = \emptyset$; and

2. $\sigma^d(R_1) = S$ and $\sigma^d(R_1 \cup R_2) \cap S \neq \emptyset$ implies $\sigma^d(R_1 \cup R_2) \supseteq S$.

We call violations of the first condition *type-1 IRR violations* and those of the second condition *type-2 IRR violations*. In the case of single-route selection functions, the above definition of IRR is equivalent to the following: if path P_1 is chosen over all paths in P as best, then additional knowledge of a route $P_2 \notin P$ does not then permit another route $P_3 \neq P_1$ in P to be chosen as best; only P_1 or P_2 may be chosen relative to $P \cup \{P_2\}$. (Condition 2 is not relevant for single-valued selection functions.)

The original SPP and PVPS frameworks only modeled selection functions that independently assign a *rank* to each route and choose the path of minimal (or maximal) rank. Selection functions written in this way are called *linear selection functions*; at each node, the preference order on unfiltered (permitted) paths is consistent with a linear order. Because the protocol-convergence conditions described in Chapters 3–6 depend on this notion of rank, they do not apply to the more general setting involving arbitrary selection functions. We note the relationship between linear selection functions and IRR below.

DEFINITION 7.1.3. A selection function σ is a *linear selection function* iff there exists a map $\omega : \mathcal{P} \to \mathcal{U}$ from permitted paths $\mathcal{P}$ to a totally ordered set $\mathcal{U}$ such that

$$\forall\, R \subset \mathcal{P},\;\; \sigma(R) = \{P \mid \forall\, P' \in R,\, \omega(P) \leq \omega(P')\}\,.$$

PROPOSITION 7.1.4. *A selection function has no IRR violations iff it can be written as a linear selection function.*

Proof. First assume that σ is a linear selection function; then there exists some ranking function ω such that $\sigma(R) = \{P \mid \omega(P) \leq \omega(P') \, \forall \, P' \in R\}$. If there were a type-1 IRR violation, then there exist R_1, R_2 such that $\sigma(R_1) = S_1$ and $\sigma(R_1 \cup R_2) = S_2$ such that $S_2 \subset (R_1 \setminus S_1)$. But then for all $P \in S_2$, it must be that $\omega(P) \leq \omega(P')$ for all $P' \in (R_1 \cup R_2)$, but this implies that $\omega(P) \leq \omega(P')$ for all $P' \in R_1$, thus $P \in S_1$, which contradicts $S_2 \subset (R_1 \setminus S_1)$. If there were a type-2 IRR violation, then there exist R_1, R_2 such that $\sigma(R_1) = S_1$ and $\sigma(R_1 \cup R_2) = S_2$ such that $S_2 \cap S_1 \neq S_1$ and $S_2 \cap S_1 \neq \emptyset$. Let $P \in S_1 \setminus S_2$; then $\omega(P) \leq \omega(P')$ for all $P' \in R_1$ but there exists $P'' \in (R_1 \cup R_2)$ such that $\omega(P) > \omega(P'')$. Thus $P'' \in R_2$, but by transitivity, if $\omega(P) \leq \omega(P')$ for $P' \in R_1$, then $\omega(P'') < \omega(P')$; this means that no route $P' \in R_1$ could be chosen by $\sigma(R_1 \cup R_2)$, which contradicts $S_2 \cap S_1 \neq \emptyset$. Therefore, any linear selection function has no IRR violations.

Now assume that we have an IRR selection function σ. For all $R \subseteq \mathcal{P}$, if $\sigma(R) = S$, then assign path ranks such that $\omega(P) \leq \omega(P')$ for $P \in S$, $P' \in R$. Suppose this ordering of path ranks is not consistent with a linear order; then there exist two permitted paths P_1, P_2 such that more than one of $\omega(P_1) < \omega(P_2)$, $\omega(P_1) > \omega(P_2)$, and $\omega(P_1) = \omega(P_2)$ are true. Suppose that $\sigma(\{P_1, P_2\}) = \{P_1\}$ so that $\omega(P_1) < \omega(P_2)$, but $\omega(P_2) \leq \omega(P_1)$: then there exists $R \subset \mathcal{P}$ such that $\sigma(R) \cap \{P_1, P_2\} = \{P_2\}$ and $R \supset \{P_1, P_2\}$. But then let $R_1 = \{P_1, P_2\}$ and $R_2 = R \setminus \{P_1, P_2\}$; then σ has a type-1 IRR violation with R_1 and R_2, which contradicts our IRR assumption. (By symmetry, there is also a contradiction if $\omega(P_1) > \omega(P_2)$ but $\omega(P_2) \not< \omega(P_1)$.) Suppose that $\sigma(\{P_1, P_2\}) = \{P_1, P_2\}$ so that $\omega(P_1) = \omega(P_2)$ but $\omega(P_2) \neq \omega(P_1)$: then there exists $R \subset \mathcal{P}$ such that $\sigma(R) \not\supset \{P_1, P_2\}$ and $R \supset \{P_1, P_2\}$. But then let $R_1 = \{P_1, P_2\}$ and $R_2 = R \setminus \{P_1, P_2\}$; then σ has a type-2

IRR violation with R_1 and R_2, again contradicting our IRR assumption. Therefore, any IRR selection function can be written as a linear selection function. □

Previous work has conjectured that IRR violations are a major cause of protocol oscillations [BOR+02, GW02A]. Below we prove that given one IRR violation at a node, we can construct a simple network on which the protocol diverges, but in which the other nodes' selection functions obey IRR and could satisfy previously established convergence conditions.

THEOREM 7.1.5. *Suppose σ_v is an IRR-violating (nonlinear) selection function. Then there exists an oscillating network instance containing node v in which all other nodes have IRR (linear) selection functions.*

Proof. Assume single-valued selection; the argument below generalizes. If σ violates IRR, then there exists a set Y of paths to d containing at least vP_1, vP_2 such that $\sigma_v^d(Y) = vP_1$ and for some set of paths $Z \neq \emptyset$ such that $Z \cap Y = \emptyset$, $\sigma_v^d(Z \cup Y) = vP_2$. Let the next hops on $vP_1, vP_2, \ldots \in Y$ be $v_1, v_2, \ldots$, respectively; assume that these paths are fixed, *i.e.*, $\forall\, R \ni P_1$, $\sigma_{v_1}^d(R) = P_1$ and analogously for the other $\sigma_{v_i}^d$. Let the next hops on $vZ_1, vZ_2, \ldots \in Z$ be $z_1, z_2, \ldots$, respectively. For each z_i, let $\sigma_{z_i}^d(R \cup \{Z_i\})$ be Z_i if $R \not\ni z_i v P_2$ and $z_i v P_2$ if $R \ni z_i v P_2$; but, assume that Z_i are fixed, *i.e.*, these paths are always broadcast.

This instance diverges. Assume that the links between all routers use first-in-first-out (FIFO) communication, though the network may be asynchronous. Consider a snapshot in time in which v has chosen vP_2 as best; this only happens if it also learns of all paths in Z, which means that all z_i have selected Z_i as their best paths. After this, v will broadcast vP_2 to its neighbors, eventually reaching the z_i. These nodes will thus switch to $z_i v P_2$, withdrawing Z_i. When the withdrawal reaches v, it will switch to choosing vP_1 as best, withdrawing vP_2. This withdrawal will eventually reach the z_i (after the first broadcast of vP_2) causing these nodes to switch back to Z_i; the broadcast of these choices back to v returns us to the original state when all the z_i switch, thus producing an oscillation.

If we examine the network when v has chosen vP_1 as best, there are two possibilities: (1) All z_i have chosen Z_i as best but the broadcasts of these choices have not yet reached v; or (2) some z_i (possibly all) have not chosen Z_i as best. In case (1), the broadcasts of Z_i will eventually reach v causing a choice of vP_2 as best; this leads to the starting state above. In case (2), if Z_i is not best at some z_i, z_ivP_2 must be available because Z_i is fixed; thus, v must have chosen vP_2 at some previous time. In this case, choose that snapshot of time as a starting point, and it is clear that the above oscillation will occur. $\square$

7.1.2 THE GENERALIZED STABLE-PATHS PROBLEM

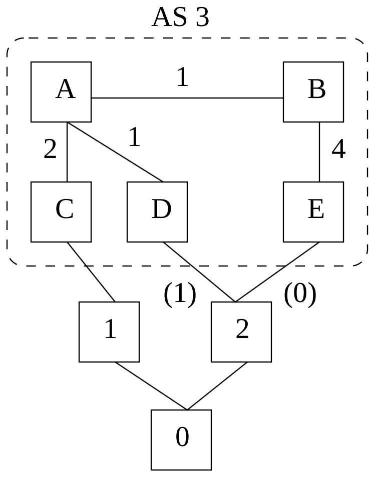
SPP limited nodes' route-selection functions to linear selection functions. We now present the generalized version first discussed in [GW02A] to accommodate modeling attributes in BGP that are inconsistent with independent route ranking.

DEFINITION 7.1.6. An instance of the *Generalized Stable Paths Problem (GSPP)* is a network $G = (V, E)$ and a set of permitted paths $\mathcal{P}$ in G to a fixed destination node $v_0 \in V$. (The set $\mathcal{P}$ of permitted paths can be partitioned into sets $\mathcal{P}_v$, $v \in V$, which are the permitted paths at node v, *i.e.*, starting at v and ending at v_0.) All nodes $v \neq v_0$ have a route-selection function $\sigma_v^{v_0} : 2^{\mathcal{P}_v} \to \mathcal{P}_v$. A *path assignment* $\pi : V \to \mathcal{P}$ is a solution to GSPP iff $\pi(v_0) = (v_0)$ and for every $v \neq v_0 \in V$, $\pi(v) = \sigma_v^{v_0}(\{vP \in \mathcal{P} \mid P = \pi(u) \text{ and } \{u, v\} \in E\})$.

REMARK 7.1.7. GSPP is *NP*-complete. This is because GSPP is in *NP*—given a solution, it is easy to check whether it is stable—and because SPP, an *NP*-complete problem [GSW02], trivially reduces to GSPP by writing its path preferences as (linear) selection functions. Also note that this version of the problem assumes single-valued selection functions.

EXAMPLE 7.1.8. Figure 7.1 shows an example GSPP given in [GW02A, MGWR02]. This instance models the route-selection procedure of BGP running on a network in which the Multi-Exit Discriminator (MED) attribute is used. (We refer the reader to Section 2.4 for a review of BGP with MEDs.) The network is shown from the perspective of AS 3.

In Figure 7.1, IGP distances are listed as numbers next to links; MED values are listed next to inter-AS connections in parentheses. Let the fixed destination be AS 0, and assume that all paths have the same local-preference value assigned at AS 3. The selection functions for the internal routers A and B are also shown.



Selection functions for routers A and B:

$$\sigma_A^0(AC10, AD20) = AD20$$
$$\sigma_A^0(AD20, ABE20) = ABE20$$
$$\sigma_A^0(AC10, ABE20) = AC10$$
$$\sigma_A^0(AC10, AD20, ABE20) = AC10$$

$$\sigma_B^0(BAD20, BE20) = BE20$$
$$\sigma_B^0(BAC10, BE20) = BAC10$$

Figure 7.1: The GSPP MED-EVIL.

This instance, called MED-EVIL, was first given in [GW02A, MGWR02] as an example of a MED-induced oscillation. It is important to note that both selection functions have IRR violations because of the MED values set by AS 2; thus, the paths cannot be ranked and this configuration cannot be represented as a standard SPP.

We now briefly describe why this GSPP has no solution. First assume that A and B have not advertised routes to each other; then they will choose $AD20$ and $BE20$, respectively, because of minimal IGP distances. If these nodes share these choices, B will still choose $BE20$ because, even though $BAD20$ has a shorter IGP path length, its MED value is higher than $BE20$ and both paths lead to AS 2. Router A, upon learning of $ABE20$, will no longer consider $AD20$ because of its higher MED value and will choose $AC10$ instead (because its IGP path length is shorter than $ABE20$, the other viable option). When A's new choice is broadcast to B, router B will choose $BAC10$ because of its shorter IGP distance (over $BE20$), withdrawing $BE20$. However, this withdrawal removes the path through AS 2 with lower MED value, causing A to choose $AD20$ again, withdrawing $AC10$. Thus, we have an oscillation similar to that in the proof above.

7.1.3 GENERALIZED PATH-VECTOR POLICY SYSTEMS

In this section, we expand PVPSes to include arbitrary selection functions, and we incorporate GSPP as a new measure of protocol expressiveness. The expanded framework can then be used to design and analyze non-IRR path-vector protocols.

DEFINITION 7.1.9. A *generalized path-vector policy system (GPVPS)* is a triple comprising: PV, the *path-vector system* that models the underlying message-passing system for route advertisements and signaling; PL, a *policy language* used to configure local-policy inputs; and $\textsc{k}$, a *global constraint* on network instances assumed to be true for executions of the protocol modeled by PV.

A GPVPS path-vector system PV has one more component than a PVPS PV (Section 4.2.1), a constraint $\textsc{l}^\sigma$ on nodes' route-selection functions.

REMARK 7.1.10. By fixing some constraints, a GPVPS can be restricted to model a linear selection function and a standard PVPS. In particular, assume there is some totally ordered set $\mathcal{U}$ and some map $\omega : \mathcal{R} \to \mathcal{U}$, and let

$$\textsc{l}^\sigma(\sigma) \Rightarrow \forall\, d,\ \sigma^d(R) = \min\{P \mid \omega(P) \leq \omega(P')\ \forall P' \in R\}.$$

DEFINITION 7.1.11. An *instance* of a GPVPS is a pair $I = (G,\ P)$ of a network $G = (V,\ E)$ and a *configuration function* P similar to Definition 4.2.7, except that nodes have an additional configuration-function component σ_v that is the route-selection function to be used on routing tables for node v such that $\text{L}^\sigma(\sigma_v)$ holds.

A *path assignment* $\rho : V \to 2^\mathcal{R}$ is a *solution* to the instance I iff

$$\rho(v) = \sigma_v \left(F^{orig}(v) \cup \left(\bigcup_{\{u,v\} \in E} F_{(u,v)}(\rho(u)) \right) \right) ,$$

where $F_{(u,v)}$ is the arc-policy function for signaling edge (u, v) as in Definition 4.2.10.

A solution therefore must consist of consistent, best paths to each destination.

REMARK 7.1.12. We assume, just as for original PVPSes, that policy functions are *separable*, *i.e.*, if p is a policy function and R is a set of path descriptors, then $p(R) = \{p(r) \mid r \in R\}$. Separability is preserved by policy application, thus arc-policy functions are separable:

$$\forall\, (u, v) \in E,\ F_{(u,v)}(R) = \left\{F_{(u,v)}(r) \mid r \in R\right\}.$$

Because the GPVPS framework is general enough to model various types of inputs and constraints on those inputs, the implementation of intended protocol behavior is left to the protocol designer; several methods may achieve the same types of permitted routing configurations. For example, the general notion of "routing-policy expressiveness" might be captured by either allowing broad choices of selection functions for individual routers, or by defining a simple route-selection procedure and limiting path-descriptor transformations via import- and export-policy constraints.

EXAMPLE 7.1.13. To model BGP, let the set of path descriptors be the set of data records used in BGP update messages and routing tables (including attributes such as local preference and MED). Set the policy-application functions to hide (or zero-out) all private attributes on eBGP export and extend the AS-path entry, checking for and filtering loops. Then set the import and export policy constraints to limit attribute changes as described in the BGP specification [RL95], *e.g.*, local preference is an integer value in some range. Set the origination constraint to check for the proper form of path descriptors for local paths. Finally, set the selection function constraint so that all routers use the BGP selection procedure in Section 2.4.

The original PVPS framework used SPP as a semantic domain to measure the *expressiveness* of a PVPS, *i.e.*, the types of routing configurations that could be expressed using a PVPS given its constraints. Because we allow arbitrary selection functions for GPVPSes, we define a new measure of expressiveness incorporating GSPP.

DEFINITION 7.1.14. Suppose that I_r is a *restriction* of a GPVPS instance I in which the only path descriptor originated is r for some single destination d. Define the GSPP $\mathcal{S}(I_r)$ to have the set of permitted paths at each node be the realizable paths at that node, *i.e.*, $\mathcal{P} = \bigcup_{v \in V}\{P = v \cdots d \mid r(P) \neq \emptyset\}$. (These are the unfiltered paths.) Let the selection functions be the same as in the GPVPS instance. Then $\mathcal{S}(I_r)$ represents the *routing configuration* for that destination.

The *expressive power* of a GPVPS PV is the set of all allowable routing configurations (all allowable instances), *i.e.*,

$$\mathcal{M}(PV) = \{\mathcal{S}(I_r) \mid I_r \text{ is a restriction of a legal instance } I \text{ of } PV\}.$$

REMARK 7.1.15. We note that any solution π for the GSPP $\mathcal{S}(I_r)$ corresponds to a solution ρ for the restricted GPVPS instance I_r and *vice versa*. The proofs of these facts are mostly algebraic manipulation and exactly mirror the analogous proofs in Section 4.4.1 for the original SPP and PVPS.

GSPP can also be used as a better measure of expressiveness for the original PVPS; all GSPPs in this case will have linear selection functions because the original PVPSes do not allow IRR violations. The original framework defined expressiveness in terms of equivalence classes of SPPs because only the relative preference ordering of paths at each node, not the actual integer ranking function used to order paths, is important in capturing a routing configuration. Therefore, any SPP S belongs to an equivalence class of SPPs $\mathcal{E}(S)$ in which all SPPs may have different integer rank functions but have the same ordering of permitted paths at each node. We note that there is an injection from SPP equivalence classes to GSPPs; there is a canonical GSPP whose selection function orders paths in the same way as each of the SPPs in the equivalence class.

This is also clear from the definition of linear selection functions (Definition 7.1.3). Given a linear selection function, there are any number of rank maps ω that preserve the choices of the selection function. Every one of these rank assignments corresponds to a possible SPP instance (because nodes have functions assigning ranks to paths), but these SPPs all share the same relative preference between paths and thus belong to the same equivalence class. However, there is only one (canonical) GSPP for a set of selection functions assigned to nodes.

7.2 CONDITIONS FOR GENERALIZED PROTOCOL CONVERGENCE

We begin by providing the analogous definition of evaluation digraphs for GSPPs that incorporate route-selection functions. (Given this definition, we use analogous versions of the SPP convergence properties given in Section 3.2.1.)

DEFINITION 7.2.1. The *evaluation digraph* of a GSPP instance S is a directed graph $\mathcal{T}(S) = (V_{\mathcal{T}}, E_{\mathcal{T}})$ in which the nodes represent *protocol selection states*, and the edges represent transitions between states. A selection state is a path assignment $\pi \in \left(\prod_{v \in V} \mathcal{P}_v\right)$; if $\alpha \in V_{\mathcal{T}}$, then we write the path assignment corresponding to this node as π_α. The *start state* is the node corresponding to the empty path assignment, in which $\pi(v_0) = (v_0)$ and, for $v \neq v_0$, $\pi(v) = \epsilon$, the empty path. The directed edge (α, β) is present in $E_{\mathcal{T}}$ iff for all $v \neq v_0 \in V$

$$\pi_\beta(v) = \sigma_v^{v_0}\left(\bigcup_{\{u,v\} \in E} \{v\pi_\alpha(u)\}\right).$$

Given a set of routing-policy inputs, we can study the corresponding GSPP's evaluation digraph to see how they affect path-vector-protocol execution. However, an evaluation digraph is both large and complex; it is impractical to construct it as doing so requires simulating all possible update sequences in the GSPP instance. Therefore, we introduce a new version of dispute wheels and prove that it adequately captures oscillations in generalized SPPs. From that discussion, we are then able to describe oscillations in terms of a new partial order on permitted paths described by local-policy configurations. (Because our generalized version of the problem does not have a notion of rank, we must nontrivially change the components of the order in Section 4.4.2 to correctly describe the generalized robustness condition.)

7.2.1 GENERALIZED DISPUTE WHEELS

DEFINITION 7.2.2. A *generalized dispute wheel* (see Figure 7.2) contains *active nodes* $v_0, \ldots, v_k$ (with all subscripts interpreted modulo $k+1$) such that v_i has a *spoke path* Q_i to the destination d, and v_i and v_{i+1} are connected by a *rim segment* R_{i+1} such that either:

1. $\exists\, S \supseteq \{Q_i,\ R_{i+1}Q_{i+1}\}$ such that $\sigma_{v_i}^d(S) = R_{i+1}Q_{i+1}$; or

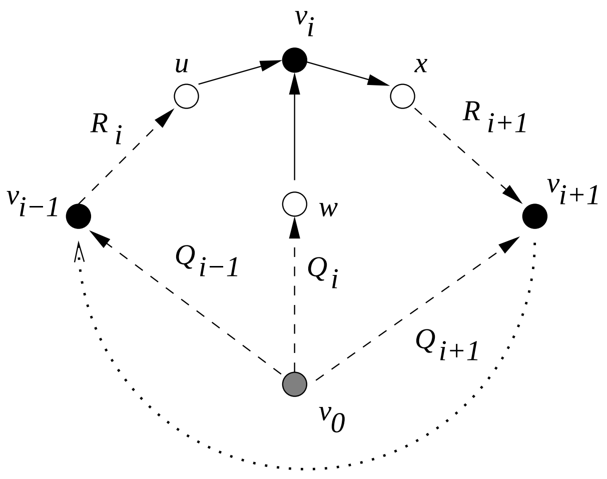


Figure 7.2: Generalized dispute wheel.

2. $\exists\, S \not\ni R_{i+1}Q_{i+1}$ such that

   (a) $\sigma^d_{v_i}\left(S \cup \{Q_i\}\right) \neq Q_i$ and

   (b) $\sigma^d_{v_i}\left(S \cup \{Q_i,\ R_{i+1}Q_{i+1}\}\right) = Q_i$; or

3. $\exists\, S \not\ni R_{i+1}Q_{i+1}$ such that

   (a) $\sigma^d_{v_i}\left(S \cup \{Q_i\}\right) = Q_i$ and

   (b) $\sigma^d_{v_i}\left(S \cup \{Q_i,\ R_{i+1}Q_{i+1}\}\right) \notin \{Q_i,\ R_{i+1}Q_{i+1}\}$.

REMARK 7.2.3. Note that of the three relationships between active nodes in a generalized dispute wheel, only condition (1) can occur for a linear selection function; conditions (2) and (3) imply the existence of an IRR violation. Condition (1) is analogous to the condition on rim segments found in the original definition of a dispute wheel for standard SPPs.

THEOREM 7.2.4. *If the evaluation digraph of a GSPP instance contains a cyclic trace, i.e., if a GSPP instance is not safe, then it contains a generalized dispute wheel.*

Proof. Let C be a cycle in the evaluation digraph of the instance, v_0 a node which does not select the same route throughout C, and Q_0 one of the paths that v_0 selects in C. Without loss of generality, we may assume that u is the last (and thus only) node on Q_0 that does not select the same route throughout C. Viewing Q_0 as one of the spokes of a generalized dispute wheel, we now attempt to construct another such spoke and a rim segment joining it to the spoke Q_0.

Let v_0P_1 be the next path that v_0 selects in C, and let x_1 be the first node on P_1. If x_1 oscillates its path selection in C, then let v_1 be the last cycling node on P_1, let $Q_1 = v_1 \cdots d$ be the next spoke, and $R_1 = v_0x_1 \cdots v_1$ be the rim segment connecting these two spokes. (Both Q_1 and R_1 are subpaths of P_1.) Because x_1 oscillates in C, it must broadcast and withdraw P_1 during the oscillation, and one of these actions causes the selection-state transition; thus the rim segment satisfies condition (1) in Definition 7.2.2.

If x_1 does not oscillate in C, let v_0P_2 be the path that v_0 selects in C after v_0P_1 and x_2 the first node on P_2. If x_2 cycles in C, we may proceed as above, otherwise we consider the path v_0P_3 that v_0 selects in C after v_0P_2, *etc.* Eventually, we either construct another spoke connected to Q_0 by a new rim segment or we progress through all of C and return to the path assignment in which v_0 selects v_0P_1. If the latter happens, then v_0 cycles through a sequence of paths in C, and each of these paths is learned from a neighbor who does not cycle in C. All of these paths are thus known to v_0 at all times, therefore all of the changes in path assignment to v_0 must be the result of IRR violations. (This is because a change in path assignment requires that v_0 know of different routes before and after the change. If the change selects a route that was already known but not chosen, by Definition 7.1.2, the selection function for v_0 has a type-1 IRR violation.)

In this case, assume that v_0's selection of Q_0 is the result of $\sigma^d_{v_0}(S) = Q_0$ and v_0's choice of v_0P_1 is the result of $\sigma^d_{v_0}(S_1) = v_0P_1$, with $Q_0, v_0P_1 \in (S \cap S_1)$. Because $S \,\Delta\, S_1 \neq \emptyset$, there is some route P_2 such that either learning or withdrawing v_0P_2 causes the transition from S to S_1 and Q_0 to v_0P_1. Let x be the first node on P_2 and v_1 be the last oscillating node on P_2. (There is such a node because P_2 is broadcast and withdrawn in the oscillation; otherwise we would not have this oscillation.) Then we can let $Q_1 = v_1 \cdots d$ be the next spoke, and $R_1 = v_0x \cdots v_1$ be the rim segment joining them such that either condition (2) – if P_2 is learned – or condition (3) – if P_2 is withdrawn – is satisfied.

Because the oscillation cycle is finite, we can repeat this process until we reach a selection state or path assignment that we have already visited. At this point, a subset of the spoke and rim segments will form a generalized dispute wheel. □

COROLLARY 7.2.5. *If a GSPP instance is not solvable, then it contains a generalized dispute wheel.*

Proof. An unsolvable GSPP has no sink state in its evaluation digraph; therefore all traces must contain cycles, and any of these cyclic traces produces a generalized dispute wheel by Theorem 7.2.4. □

PROPOSITION 7.2.6. *If a GSPP instance has multiple solutions, then it contains a generalized dispute wheel.*

Proof. We follow an analogous proof method in [GSW02]. Suppose π_1, π_2 are two solutions; we can view these as trees in the network, rooted at the destination v_0: $T_i = \bigcup_{v \in V} \pi_i(v)$. Then let $H = (V,\ E(T_1) \cap E(T_2))$ be the graph induced by the intersection of the trees and let T be the component of H including v_0. Note that $V - V(T)$ is nonempty – otherwise $T_1 = T_2$.

In the following process, assume that all nodes u_i are assigned paths in both solutions. Choose an edge $\{u_1, v_1\} \in T_1$ where $u_1 \notin V(T)$ and $v_1 \in V(T)$. Then $\pi_1(u_1) = u_1Q_1$, where Q_1 is the path in T from v_1 to d; $\pi_1(v_1) = \pi_2(v_1) = Q_1$ so that Q_1 is in both solutions because T is the intersection of both solutions. There is some other path $P_1 =$

$\pi_2(u_1)$ in T_2; this path is of the form R_2Q_2 where $R_2 = u_1 \cdots u_2$ contained in $T_2 \setminus H$ and $Q_2 = v_2 \cdots d$ contained in T. Note that $\pi_2(u_2) = u_2Q_2$, so we can repeat this process by examining the path $\pi_1(u_2)$. Continuing, we can alternate between both solutions until we repeat a node u_i.

The paths R_i, Q_i form a generalized dispute wheel. This is because for each i, there must exist some $S \subset \mathcal{P}_{u_i}$ such that $\sigma_{u_i}^{v_0}(S \cup \{R_{i+1}Q_{i+1}, u_iQ_i\}) = R_{i+1}Q_{i+1}$ because for either $i = 1$ or $i = 2$, $\pi_i(u_i) = R_{i+1}Q_{i+1}$ given the construction above. (If not, π_i is not a stable solution, because the path u_iQ_i must be available given that Q_i is in the intersection of both solutions.) This satisfies condition (1) in Definition 7.2.2. □

The contrapositive of the above three assertions forms a sufficient condition on GSPP instances that guarantees robust protocol convergence; we summarize this as the following.

PROPOSITION 7.2.7. *If a GSPP instance has no generalized dispute wheel, it is robust.*

7.2.2 PARTIALLY ORDERED GSPPS AND GENERALIZED DISPUTE DIGRAPHS

The three types of conditions described in Definition 7.2.2 that connect dispute-wheel spokes by rim segments can be used to define relations between permitted paths in a GSPP. These relations can, in turn, be used to define a graph structure on the paths in a GSPP that makes the relationship between paths based on policy interactions easy to visualize. The underlying compatibility of local-policy configurations can then be described as the existence of a consistent partial order on permitted paths using these relations. By rephrasing the sufficient condition from Proposition 7.2.7 using these terms, we can better understand how individual policy interactions (corresponding to the following path relations) constitute a global routing anomaly.

DEFINITION 7.2.8. Define the following four relations on permitted paths in a GSPP instance; assume that v_0 is the fixed destination node and that $u, v \in V$ are other network nodes.

SUBPATH $P_1 \ominus P_2$ iff

$$P_1 = v \cdots v_0, \quad P_2 = u \cdots v_0, \quad \text{and} \quad uP_1 = P_2$$

LINEAR SELECTION $P_1 \oslash P_2$ iff

$$P_1 = v \cdots v_0, \quad P_2 = u \cdots v_0, \quad \text{and} \quad \exists\, S : \sigma_u^{v_0}\left(\{uP_1,\ P_2\} \cup S\right) = uP_1$$

NONLINEAR SELECTION (FIRST TYPE) $P_1 \odot_1 P_2$ iff $P_1 = v \cdots v_0$, $P_2 = u \cdots v_0$, and

$$\exists S \not\ni uP_1 : \sigma_u^{v_0}\left(\{P_2\} \cup S\right) \neq P_2 \quad \text{and} \quad \sigma_u^{v_0}\left(\{uP_1,\ P_2\} \cup S\right) = P_2$$

NONLINEAR SELECTION (SECOND TYPE) $P_1 \odot_2 P_2$ iff $P_1 = v \cdots v_0$, $P_2 = u \cdots v_0$, and

$$\exists S \not\ni uP_1 : \sigma_u^{v_0}(S) = P_2 \quad \text{and} \quad \sigma_u^{v_0}\left(\{uP_1\} \cup S\right) \notin \{uP_1,\ P_2\}$$

We now define the following graph on the set of permitted paths using the above relations.

DEFINITION 7.2.9. Given a GSPP instance S, its *generalized dispute digraph* is the directed graph $\mathcal{D}(S) = (V_{\mathcal{D}}, E_{\mathcal{D}})$. The nodes $V_{\mathcal{D}} = \mathcal{P}$ are the permitted paths in the network. The directed edge $(P_1,\ P_2)$ is present in $E_{\mathcal{D}}$ iff one of $P_1 \ominus P_2$, $P_1 \oslash P_2$, $P_1 \odot_1 P_2$, or $P_1 \odot_2 P_2$ holds.

Note that the dispute digraph is smaller than the evaluation digraph as each node is labeled with a single network route rather than a set of network routes; it is also easy to build given the definition of each node's selection function.

Because the relations correspond to transitions in the evaluation digraph and connections between dispute-wheel spokes, we can prove the following.

THEOREM 7.2.10. *A GSPP instance has a generalized dispute wheel iff it has a cycle in its generalized dispute digraph.*

Proof. First assume that the instance has a generalized dispute wheel. Its rim gives a cycle in the generalized dispute digraph as follows, because the pair of paths from adjacent rim nodes to the destination each belong to one of the four relations in Definition 7.2.8. Begin with any active node v_i on the rim; let r_1 be the next node on the rim segment R_i. From the construction of the dispute wheel, $r_1 Q_i = r_1 v_i \cdots d$ is an extension of Q_i, so $Q_i \ominus rQ_i$; this relation holds for further extensions along the rim, such that $(r_i \cdots r_1 Q_i) \ominus (r_{i+1} r_i \cdots r_1 Q_i)$. Let R_i^* be the rim segment up to, but not including, v_{i-1}; using these relations, we see there is a path from Q_i to $R_i^* Q_i$ in the dispute digraph for each active node v_i in the dispute wheel. Call these paths D_i. Then, for every $R_i Q_i$ and Q_{i-1}, one of the three conditions in Definition 7.2.2 holds. In the case of condition (1), $\exists\, S : \sigma^d_{v_{i-1}}(S \cup \{R_i Q_i,\ Q_{i-1}\}) = R_i Q_i$; thus $R_i^* Q_i \oslash Q_{i-1}$, corresponding to the edge $(R_i^* Q_i,\ Q_{i-1})$ connecting D_i and D_{i-1}. In the case of condition (2), learning $R_i Q_i$ at v_{i-1} forces another route to be selected over Q_{i-1}; thus $R_i^* Q_i \odot_2 Q_{i-1}$, also corresponding to the edge $(R_i^* Q_i,\ Q_{i-1})$ connecting D_i and D_{i-1}. Finally, in the case of condition (3), withdrawing some route at v_{i-1} forces Q_{i-1} to be chosen; thus $R_i^* Q_i \odot_1 Q_{i-1}$, corresponding to the same edge connecting D_i and D_{i-1}. Therefore the dispute-digraph edges corresponding to pairwise relations between paths starting at adjacent rim nodes form a cycle.

In the other direction, assume that we have a cycle in the dispute digraph. Consider any edge $(P_1,\ P_2)$ and examine the relation between P_1 and P_2. If $P_1 \ominus P_2$, then let the first node of P_1 be a rim node and connect it to the first node of P_2 as an adjacent rim node (counterclockwise, referencing Figure 7.2.). If $P_1 \oslash P_2$, $P_1 \odot_1 P_2$, or $P_1 \odot_2 P_2$, then let P_2 be a spoke Q_i and connect the first node of P_2 to the first node of P_1 on the rim segment R_{i+1}; the subpath of P_1 from the first node to the last oscillating node will be the rim segment R_{i+1} and the remainder of P_1 will be the next spoke Q_{i+1}. The resulting structure will obey one of the three conditions in Definition 7.2.2 for rim segments connecting spokes and will have subpaths along individual rim segments (moving clockwise); therefore, this structure is the dispute wheel corresponding to the dispute-digraph cycle. □

This immediately leads to the following corollary, which provides an equivalent sufficient condition to Proposition 7.2.7 using the transitive closure of path relations (local conditions) instead of dispute-wheel freeness (a global condition).

COROLLARY 7.2.11. *Given a GSPP instance, if there is a cycle in its evaluation digraph, then the corresponding relation* $\bigcirc = (\ominus \cup \oslash \cup \odot_1 \cup \odot_2)^*$ *on permitted paths is not a partial order.*

Proof. If $P_1 \bigcirc P_2$, then there is a path in the dispute digraph from P_1 to P_2 (as in the proof of Theorem 7.2.10). If the relation $\bigcirc$ is not a partial order, there are two paths $P_1 \neq P_2$ such that $P_1 \bigcirc P_2$ and $P_2 \bigcirc P_1$; the two corresponding paths form a dispute-digraph cycle. Analogously, if there is a cycle in the dispute digraph, there are paths $P_1 \neq P_2$ corresponding to nodes in this cycle such that $P_1 \bigcirc P_2$ and $P_2 \bigcirc P_1$; thus $\bigcirc$ cannot be a partial order. The result then follows directly from Theorems 7.2.4 and 7.2.10. □

REMARK 7.2.12. The original set of relations defined in Section 4.4.2 for SPP partial ordering could assume linear selection functions; thus both types of the "nonlinear selection" relation were not used. However, recall that the "linear selection" relation was also different, defined as follows: assuming that ω is a ranking function, $P_1 \oslash P_2$ iff $\omega(P_1) \leq \omega(P_2)$. In this version of the definition, both paths begin at the same node, and the extension of

P_1 to u in Definition 7.2.8 was captured in the transitive closure of $\oslash$ with the subpath relation $\ominus$. If we defined a similar selection relation, *i.e.*, $P_1 \oslash P_2$ iff there exists some S such that $\sigma(\{P_1,\ P_2\} \cup S) = P_1$, then any IRR violation would automatically introduce a cycle in the dispute digraph (this fact follows directly from Definition 7.1.2). Because not all such IRR violations cause protocol oscillations (given other nodes' policies), subsuming one subpath relation into the three selection relations eliminates these spurious cycles from dispute digraphs. Consequently, the example dispute cycles below will appear different than the disputes in [GSW02].

Using the generalized dispute digraph, one can diagnose the cause of oscillations: cycles that involve nonlinear-selection edges are the result of IRR violations, and cycles only involving linear-selection and subpath edges are manifestations of basic policy disputes not based on IRR violations.

7.2.3 EXAMPLE GSPPS AND DISPUTE DIGRAPHS

In the following diagrams of generalized dispute digraphs, we will use the following convention for edges: solid lines correspond to subpath relations, dashed lines correspond to linear-selection relations, and dotted-and-dashed lines correspond to nonlinear-selection relations. Of these, those with solid arrowheads are of the first type while those with white arrowheads are of the second type.

EXAMPLE 7.2.13. We begin with the generalized dispute digraph for MED-EVIL, the GSPP from Example 7.1.8. To simplify the diagram, we have condensed ASes 1, 2, and 0 into a single AS 0 connected to routers C, D, and E; we can write analogous selection functions that maintain the policies and MED-induced oscillation in the original MED-EVIL. The graph is shown in Figure 7.3.

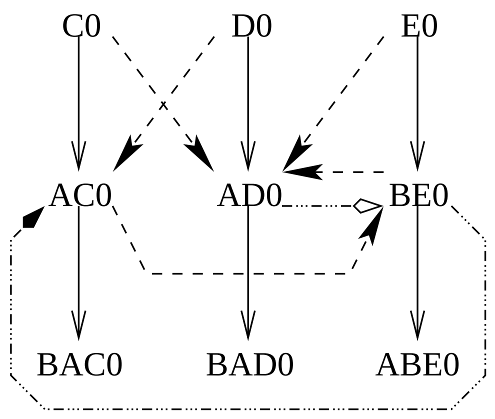


Figure 7.3: Generalized dispute digraph for MED-EVIL.

This digraph has two cycles, $AC0 - BE0$ and $AD0 - BE0$; as expected, both of these involve nonlinear selection edges and the paths that cause IRR violations. The policies of MED-EVIL create no oscillation when MEDs are ignored: note that the linear-selection edges do not form any cycles. Involving MEDs creates relations between paths that are not consistent with a partial order. To achieve a partial order, we can attempt to change local policies to change the relations, *i.e.*, break the cycle; *e.g.*, we can force node A to always choose the path $AC0$.

EXAMPLE 7.2.14. We now revisit the two canonical policy-induced oscillations discussed in Section 3.1. The instance DISAGREE is shown in Figure 7.4; it contains two stable solutions but does not predictably converge to either one, thus its dispute digraph contains a cycle. The instance BAD GADGET is shown in Figure 7.5; it has no solution, so its dispute digraph also contains a cycle. Because these instances have linear selection functions, they are shown as standard SPPs.

7.3 APPLICATIONS TO PROTOCOL DESIGN

In this section we examine some strategies for constraining policies to guarantee robust protocol convergence. Although dispute wheels and dispute digraphs are useful tools for

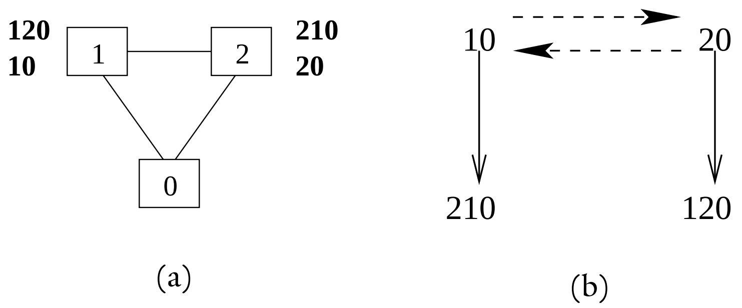


Figure 7.4: (a) The SPP instance DISAGREE and (b) its corresponding generalized dispute digraph.

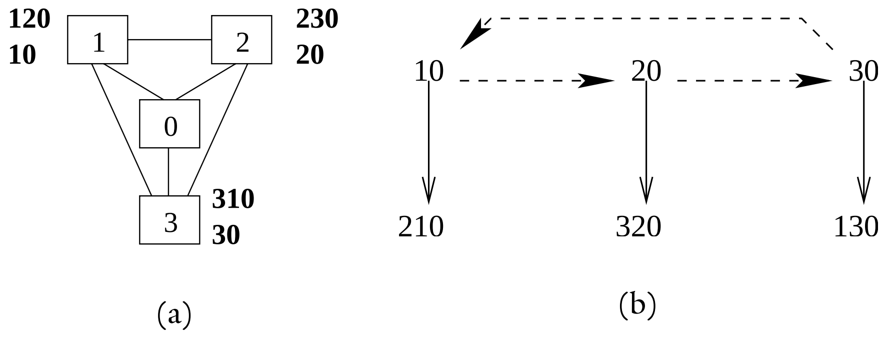


Figure 7.5: (a) The SPP instance BAD GADGET and (b) its corresponding generalized dispute digraph.

studying policy interactions, using them can be impractical for real network configurations. The dispute digraph has size proportional to the number of loopless paths in a network; checking for dispute wheels is at least as hard, because there is no known way to directly produce a dispute wheel without an instance's dispute digraph or evaluation digraph. Furthermore, it is almost impossible to obtain Internet-wide policy information to generate these structures, and the structures will be different every time nodes make policy changes. Ideally, we want constraints on the protocol specification or policy-configuration language that applies to a broad set of networks and routing configurations—we would like to use the sufficient condition from the previous section while allowing for as much policy expressiveness as possible.

Unfortunately, generalizing the types of constraints in Chapters 4–6 is hard because our new model allows for nonlinear selection functions, which removes any notion of path-rank values, and because our model broadens the notion of the protocol's route-selection procedure arbitrarily.

Some obvious, draconian constraints, *e.g.*, preventing the advertisement of any route that causes an IRR violation, can be trivially shown to prevent routing anomalies, but these are very strict and harshly limit expressive power. Below we review a specific proposal to review MED-induced oscillations in BGP, and we use our tools to suggest an improvement. In the following subsections, we discuss two other conjectured solutions and, using the results from the PVPS and GPVPS frameworks, prove them to be true.

7.3.1 MULTIPLE-PATH BROADCAST

Basu *et al.* [BOR$^+$02] and Musunuri and Cobb [MC04] proved that a modification to BGP's update messages will prevent MED-induced oscillations. They suggested that nodes broadcast not only best routes, but any route that remains after step 3 in the BGP route-selection process (see Example 7.1.8), *i.e.*, all routes with minimal MED values, possibly one for each AS, are broadcast, not only the one with minimal IGP distance to the egress point. This prevents routes that cause IRR violations from being broadcast and withdrawn repeatedly. In the case of MED-EVIL in Example 7.1.8,[1] node B would then always broadcast the route $BE20$, even though it would never select it. Because the route is never chosen elsewhere due to its high MED value, this introduces no consistency problems. However, it (1) allows other nodes to make the correct choice of routes with respect to MED values and (2) stops the oscillation by making that choice stable.

We can see the effect of such a change by examining the cyclic traces in the evaluation

[1]As in Example 7.2.13, we simplify the instance by condensing AS 1, AS 2, and AS 0 into a single AS 0 and modifying the selection functions accordingly.

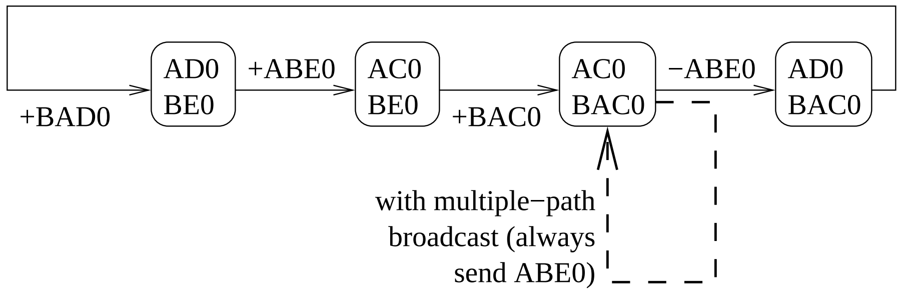


Figure 7.6: Cycle in the evaluation digraph of MED-EVIL.

digraph. The MED-induced cycle of MED-EVIL is shown in Figure 7.6. The nodes show the selections of nodes A and B, and the labels on arrows show the causes of transitions (routes being advertised or withdrawn). Note the IRR violation is clear in the transition between the first and second states; node A switches from $AD0$ to $AC0$ by learning a different route, $ABE0$. With multiple-path broadcast, the withdrawal of $ABE0$ never takes place; therefore the state $(AC0, BAC0)$ becomes a sink state and a stable assignment.

This effect easily generalizes to all GSPP instances involving MEDs: any MED-induced oscillation corresponds to an evaluation-digraph cycle of the above form, and preventing a route withdrawal by broadcasting additional routes will break the cycle by preventing one (or more) of the cycle's transitions. In addition, because more routes are always broadcast, nodes will not choose higher-MED-valued routes when lower-MED-valued routes are available, thus preserving the intended behavior of the MED attribute. More generally, if routes causing IRR violations are always broadcast, the resulting route-selection functions with restricted domain have no IRR violations.

Multiple-path broadcast can increase the size of routing tables and update messages. However, we propose that IRR violations can be detected dynamically, precisely when a newly learned route causes a switch in selection without selecting the new route. Requesting that the new route always be broadcast will prevent a future oscillation due to withdrawal of that route without any route inconsistencies. Maintaining one extra route as needed is more storage-efficient than the multiple-path broadcast proposed by [BOR+02, MC04]. Although this solution requires further modification to BGP, dynamic detection of IRR violations is possible by examining protocol-execution traces. In practice, whenever a BGP update message is received, the route selection before and after the update message can be compared. If the new selection is neither the old selection or the newly learned route, the route points to an IRR violation (this is clear from Definition 7.1.2. Requesting this IRR-violating route to be broadcast as long as it is available prevents any induced oscillations because the route essentially becomes fixed, breaking the cycle of withdrawals and advertisements in the evaluation digraph. Formally, we have the following.

PROPOSITION 7.3.1. *An oscillation due to an IRR violation can be dynamically detected and stopped by requesting one additional route to be broadcast permanently.*

Proof. Given a cycle in the evaluation digraph involving an IRR violation, there are transitions in this cycle involving an advertisement or withdrawal of a route that is never selected. This route can be detected by comparing path assignments in the states adjacent to these transitions. If the withdrawal transition is prevented by forcing the route to be advertised as long as it is available, even if it is not chosen, the withdrawal transition cannot take place and the cycle is broken. □

This procedure breaks cycles in a "snapshot of time," *i.e.*, for a static routing configuration that induces a protocol oscillation. If changes occur and routes are introduced or withdrawn for legitimate causes, the resulting GSPP instance will have a different evaluation digraph; however, the relevant IRR-violating routes can be detected for this new configuration in the same way. If the IRR-violating route is no longer available, the broadcasting node can send the appropriate withdrawal—this still allows the receiving node to detect new IRR violations involving other routes. Furthermore, if any IRR-violating selections are superseded by learning new routes that are always more preferred or by other IRR-violating routes, the original routes are not needed and the broadcast can be stopped.

7.3.2 COMPARE ALL MEDS

Some routers have an option to change the route-selection procedure involving MEDs: In step 3 of the BGP procedure described in Example 7.1.8, instead of eliminating multiple paths to the same AS by choosing the one with lowest MED value, MED values are compared across all paths so that, regardless of AS next-hop, only paths with the lowest MED values are retained for possible selection.

This option essentially changes the route-selection procedure so that it is linear: for each path, the preference of that path depends, in order, on its local preference, then path length, then MED value, and finally IGP distance. Therefore, IRR violations are no longer possible, and previous convergence constraints apply. In fact, because local-preference, AS-path length, and MED values do not change during intra-domain BGP (iBGP) sessions, and because IGP distances increase as paths are extended, the absolute rank value associated with paths increases on extension. This obeys the strict-monotonicity constraints of [GJR03, SOB03], so MED-induced oscillations cannot occur. (Of course, more general policy-induced oscillations due to, *e.g.*, local-preference settings, can still occur.)

Formally, comparing all MEDs changes the route-selection procedure such that selection functions are linear, compatible with the rank map $\omega(l, P, m, d) = (-l, |P|, m, d)$, lexically ordered, where l is the local preference, P is the AS path (path vector), m is the MED value, and d is the IGP distance.

7.3.3 AS-DISTINCT LOCAL-PREFERENCE SETTINGS

McPherson *et al.* in [MGWR02] suggest a workaround for MED-induced oscillations that prevents BGP from having a conflict when it reaches the MED step. If only routes from one AS remain when MEDs are considered, then all routes have their MED values compared and, similar to above, IRR violations are not possible. One simple way to do this is to assign local-preference values such that no two routes from different ASes have the same value; then the first step of the BGP selection process will automatically eliminate all routes except those from a single AS. (One can also assign distinct local-preference values to equidistant ASes; then the first two steps eliminate all routes but those from one AS.)

This route-selection procedure is consistent with linear selection functions because, just as above, the rank of a route independently depends on four criteria in order. Once the MED value is considered, all remaining routes have the same local preference, path length, next-hop AS, and MED value, again leaving the strictly monotonic IGP distance to be used to break ties. Therefore, this modification to BGP prevents MED-induced anomalies.

CHAPTER 8

CONCLUSIONS AND OPEN QUESTIONS

The path-vector policy system (PVPS) framework is a rigorous model that can be used to understand the behavior of inter-domain routing protocols. Previous work has given point solutions in the design space of robust routing protocols and has given some sufficient conditions on specific network instances or specific protocol implementations. This dissertation, however, uses the methodology of formal modeling to investigate the underlying convergence issues of path-vector protocols and inter-domain routing *in general*, without being constrained by the details of BGP. Doing so provides a complete description of the design space. In Chapter 4, we identified and rigorously defined several dimensions of this space corresponding to desirable protocol properties and demonstrated several inherent trade-offs in this space—these apply broadly to the design of inter-domain routing protocols. In particular, we showed the importance of including the development of a global constraint in the design of any routing system. Such design principles are required if we are to move beyond *ad hoc* fixes every time policy-interaction problems are encountered.

Our characterization of robust routing systems was not complete, however. Either Conjecture 4.5.3 must be proven or a broader sufficient condition for robustness should be found. Furthermore, we have not shown the simultaneous trade-offs among *all* dimensions of the design space. Design properties such as security, privacy, and policy opaqueness should be studied further. And, if we begin the design process with tractable global constraints as our starting point, we should be able to characterize the level of expressiveness that can be achieved with an autonomous, transparent, and robust system with an imposed global constraint that can be checked in polynomial time. Although we have addressed the balance of local and global constraints in the context of class-based systems in Chapter 6, we have yet to study this question in general.

Chapter 6 does, however, complete the analysis of these class-based systems that generalize Hierarchical-BGP and related protocols. In particular, we showed how to use the specification of a generic class-based system to generate a global constraint which guarantees the robust convergence of any network instance satisfying it. Our constraint is the best-known such constraint for these systems, and we provided centralized and distributed algorithms to enforce it. The question of how to *efficiently* run our distributed algorithm in parallel remains open, however: In particular, token-traversal paths from separate instances of the algorithm can probably be combined to find and fix all potential dispute wheels simultaneously.

To date, class-based systems seem to be the only path-vector systems well-characterized enough that an exact balance of local and global constraints for them can be proven. Other examples of path-vector policy systems should be studied in a similar way, hopefully yielding constraints and enforcement mechanisms that guarantee robustness for a larger set of path-vector protocols.

In Chapter 5, we showed that the results of the path-vector algebra framework [SOB03] are essentially equivalent to those in the PVPS framework. Because we can translate specifications between frameworks, the results discussed above can be applied to different levels of abstraction. However, we have yet to develop new design principles using a combination of frameworks; we do expect that the translation will aid continuation of work in this area, because the more suitable framework for a specific task can be used, and our translation can be applied to describe the result in the other framework when necessary.

Chapter 7 has fully extended the notion of convergence conditions on SPPs to the generalized version of the problem, allowing arbitrary route-selection procedures instead of those based on some notion of a linear path rank. Doing so allows us to fully understand the causes of policy-induced routing anomalies from the perspective of an underlying mathematical consistency between nodes' policies. The generalized version is able to capture the behavior of protocols, *e.g.*, BGP with MEDs, that do not satisfy independent route ranking (IRR). In examining these generalized policy interactions, we rigorously defined protocol-convergence properties and several useful graph structures that illustrate protocol behavior. We provided two equivalent sufficient conditions for robust convergence, both of which involve structures that are significantly smaller than examining the execution states of the protocol directly. We also discussed applications of these results to protocol design, including some simple (but strict) local-policy constraints and rigorous examinations of proposed solutions to MED-induced oscillations.

There are two obvious directions for future work left open by the results in Chapter 7. The first is the development of analogous constraints to those in Chapters 4–5 for arbitrary selection functions. The original increasing or monotonicity condition depends on path-rank values, which are not available in the generalized version. We have only been able to develop very simple local-policy constraints, and these limit expressiveness quite a bit. Because route-selection procedures are so broad, we expect generalizing previous notions of constraints to be difficult. One method that may prove fruitful is examining a restricted set of route-selection procedures, *e.g.*, the application of PVPSes to class-based systems in

Chapter 6. Similar work could be done for systems allowing IRR violations.

The second is a further broadening of the model to capture the static semantics of policy interactions when multiple-path broadcast is used; more generally, the results can be extended to SPPs with set-valued arbitrary selection and broadcast functions. The notion of a broadcast function is similar to a selection function: it returns a subset of paths, as dictated by the protocol specification, that are advertised to neighbors, given nodes' routing tables as inputs. (This allows more latitude in modeling step 3 of path-vector-protocol behavior discussed in Section 2.3.) Separating broadcast functions from export policies allows more freedom in implementing constraints at different parts of the route-advertisement process and keeps the separability property for policy functions. Unfortunately, it seems that the problem statement for arbitrary, set-valued selection and broadcast functions is difficult; in fact, the problem of finding a stable, consistent solution for such a routing configuration may not be in *NP*, because there may be a set of oscillating routing tables for each node that, given particular broadcast functions, do give at least one stable solution. This extension to the model could help design and evaluate protocols or protocol modifications, such as the ones reviewed briefly in Section 7.3.1, without having to study large evaluation digraphs.

We are able to capture the static semantics of deterministic routing using the SPP/PVPS and GSPP/GPVPS frameworks. However, in non-deterministic systems, the static and dynamic semantics may become intertwined; *e.g.*, a node might use some temporal condition to break ties between equally ranked routes from different neighbors in a BGP-like system—a system that prefers more recent routes will have very different semantics than one that prefers older routes. Both non-deterministic systems and their dynamic semantics should be investigated.

While security may be an orthogonal issue, it may be possible to use a similar formal-modeling approach to prove guarantees about secure protocol behavior. It seems unlikely that adding mechanisms for authenticating update messages or protecting routers will change protocol-convergence properties, but a formal proof of that fact could hasten deployment of a secured routing protocol.

Finally, we note that we have not addressed the actual development of deployable policy-configuration languages using our design principles. In the case of BGP, it is unclear that new languages can adequately enforce robustness constraints and provide desired expressiveness at the same time. However, there may be slight modifications to BGP and its languages that are safe, *e.g.*, hierarchical BGP with back-up routes [GGR01]. We have yet to design a next-generation routing protocol and policy-configuration languages from first principles, but we hope the results in this dissertation provide a foundation for doing so.

BIBLIOGRAPHY

[ABG$^+$98] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizar. Routing Policy Specification Language (RPSL). RFC 2280, January 1998.

[BCC00] T. Bates, R. Chandra, and E. Chen. BGP Route Reflection – An Alternative to Full Mesh IBGP. RFC 2796, April 2000.

[BOR$^+$02] Anindya Basu, Chih-Hao Luke Ong, April Rasala, F. Bruce Shepherd, and Gordon Wilfong. Route Oscillations in I-BGP with Route Reflection. In *Proceedings of ACM SIGCOMM'02*, pages 235–247, ACM Press, November 2002.

[BQ03] Olivier Bonaventure and Bruno Quoitin. Common Utilizations of BGP Community Attribute. `draft-bonaventure-quoitin-bgp-communities-00`, IETF Internet Draft. Work in progress, 2003.

[CIS01] Cisco Systems. Endless BGP Convergence Problem in Cisco IOS Software Releases. Field Note, `http://www.cisco.com/warp/public/770/fn12942.html`. October 2001.

[CTL96] R. Chandra, P. Traina, and T. Li. BGP Communities Attribute. RFC 1997, August 1996.

[DS98] Rohit Dube and John G. Scudder. Route Reflection Considered Harmful. `draft-dube-route-reflection-harmful-00`, IETF Internet Draft. Work in progress, November 1998.

[GAE$^+$99] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W.S. Lee. An Architecture for Stable, Analyzable Internet Routing. *IEEE Network*, 13(1):29–35, January / February 1999.

[GGR01] Lixin Gao, Timothy G. Griffin, and Jennifer Rexford. Inherently Safe Backup Routing with BGP. In *Proceedings of IEEE INFOCOM 2001*. IEEE Communications Society, IEEE Press, April 2001.

[GJR03] Timothy G. Griffin, Aaron D. Jaggard, and Vijay Ramachandran. Design Principles of Policy Languages for Path Vector Protocols. In *Proceedings of*

ACM SIGCOMM'03, pages 61–72. ACM Press, August 2003. Extended version available as Yale University Technical Report YALEU/DCS/TR-1250, `ftp://ftp.cs.yale.edu/pub/TR/tr1250.pdf`.

[GR01] Lixin Gao and Jennifer Rexford. Stable Internet Routing without Global Coordination. *ACM/IEEE Transactions on Networking*, 9(6):681–692, December 2001.

[GSW02] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The Stable Paths Problem and Interdomain Routing. *ACM/IEEE Transactions on Networking*, 10(2):232–243, April 2002.

[GW00] T. Griffin and G. Wilfong. A Safe Path Vector Protocol. In *Proceedings of IEEE INFOCOM 2000*, IEEE Communications Society, IEEE Press, March 2000.

[GW02A] Timothy G. Griffin and Gordon Wilfong. An Analysis of the MED Oscillation Problem in BGP. In *Proceedings of the 10th International Conference on Network Protocols (ICNP'02)*, pages 90–99, IEEE Press, November 2002.

[GW02B] Timothy G. Griffin and Gordon Wilfong. On the Correctness of IBGP Configuration. In *Proceedings of ACM SIGCOMM'02*, pages 17–29, ACM Press, August 2002.

[HEN88] C. Hendrick. Routing Information Protocol (RIP). RFC 1058, June 1988.

[HUS99A] Geoff Huston. Interconnection, Peering and Settlements: Part I. *Internet Protocol Journal*, 2(1), June 1999.

[HUS99B] Geoff Huston. Interconnection, Peering and Settlements: Part II. *Internet Protocol Journal*, 2(2), June 1999.

[HUS01] Geoff Huston. Scaling Interdomain Routing—A View Forward. *Internet Protocol Journal*, 4(4):2–16, December 2001.

[JR04] Aaron D. Jaggard and Vijay Ramachandran. Robustness of Class-Based Path-Vector Systems. In *Proceedings of the 12th International Conference on Network Protocols (ICNP'04)*, pages 84–93. IEEE Press, October 2004. Extended version available as Yale University Technical Report YALEU/DCS/TR-1296, `ftp://ftp.cs.yale.edu/pub/TR/tr1296.pdf`.

[JR05A] Aaron D. Jaggard and Vijay Ramachandran. Relating Two Formal Models of Path-Vector Routing. In *Proceedings of IEEE INFOCOM 2005* (electronic only). IEEE Press, March 2005.

[JR05B] Aaron D. Jaggard and Vijay Ramachandran. Robustness of Path-Vector Protocols without Independent Route Ranking. Technical Report YALEU/DCS/TR-1314, Yale University, April 2005. `ftp://ftp.cs.yale.edu/pub/TR/tr1314.pdf`.

[JR05C] Aaron D. Jaggard and Vijay Ramachandran. Towards the Design of Robust Inter-domain Routing Protocols. Manuscript, March 2005.

[MC04] Ravi Musunuri and Jorge A. Cobb. A Complete Solution for iBGP Stability. In *Proceedings of IEEE ICC-04*. IEEE Press, June 2004.

[MGWR02] D. McPherson, V. Gill, D. Walton, and A. Retana. Border Gateway Protocol (BGP) Persistent Route Oscillation Condition. RFC 3345, August 2002.

[POS80] Jon Postel. User Datagram Protocol. RFC 768, August 1980.

[POS81A] Jon Postel. Internet Protocol. RFC 791, September 1981.

[POS81B] Jon Postel. Transmission Control Protocol. RFC 793, September 1981.

[RL95] Y. Rehkter and T. Li. A Border Gateway Protocol (BGP version 4). RFC 1771, 1995.

[RLA04] B. Rajagopalan, J. Luciani, and D. Awduche. IP over Optical Networks: A Framework. RFC 3717, March 2004.

[RR99] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. RFC 2547, 1999.

[RSS02] Jonathan Rosenberg, Hussein Salma, and Matt Squire. Telephony Routing over IP (TRIP). RFC 3219, January 2002.

[SOB03] João L. Sobrinho. Network Routing with Path Vector Protocols: Theory and Applications. In *Proceedings of ACM SIGCOMM'03*, pages 49–60. ACM Press, August 2003.

[STA99] Richard P. Stanley. *Enumerative combinatorics. Vol. 2*. Cambridge University Press, Cambridge, 1999.

[STR05] Srihari R. Sangli, Daniel Tappan, and Yakov Rekhter. BGP Extended Communities Attribute. `draft-ietf-idr-bgp-ext-communities-08`, IETF Internet Draft. Work in progress, February 2005.

[VGE00] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent Route Oscillations in Inter-domain Routing. *Computer Networks*, 32(1):1–16, March 2000.

[WCRS02] Daniel Walton, David Cook, Alvaro Retana, and John Scudder. BGP Persistent Route Oscillation Solution. `draft-walton-bgp-route-oscillation-stop-00`, IETF Internet Draft. Work in progress, May 2002.

[XBX03] Y. Xu, A. Basu, and Y. Xue. A BGP/GMPSL Solution for Inter-domain Optical Networking. `draft-xu-bgp-gmpls-03`, IETF Internet Draft. Work in progress, September 2003.