

Design Principles of Policy Languages for Path Vector Protocols*

Timothy G. Griffin
AT&T Labs – Research
Florham Park, NJ, USA
griffin@research.att.com

Aaron D. Jaggard[†]
Dept. of Mathematics
Tulane University
New Orleans, LA, USA
adj@math.tulane.edu

Vijay Ramachandran[‡]
Dept. of Computer Science
Yale University
New Haven, CT, USA
vijayr@cs.yale.edu

ABSTRACT

BGP is unique among IP-routing protocols in that routing is determined using semantically rich routing policies. However, this expressiveness has come with hidden risks. The interaction of locally defined routing policies can lead to unexpected global routing anomalies, which can be very difficult to identify and correct in the decentralized and competitive Internet environment. These risks increase as the complexity of local policies increase, which is precisely the current trend. BGP policy languages have evolved in a rather organic fashion with little effort to avoid policy-interaction problems. We believe that researchers should start to consider how to *design* policy languages for path-vector protocols that 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 attempt to do this in a general way that is not constrained by the details of BGP.

Categories and Subject Descriptors

C.2 [Computer Systems Organization]: Computer-Communication Networks—*Network Protocols, Internetworking*

General Terms

Design, Languages, Theory

*This work was partially supported by the U.S. Department of Defense (DoD) University Research Initiative (URI) program administered by the Office of Naval Research (ONR).

[†]Partially supported by ONR Grant N00014-01-1-0795 and by ONR Grant N00014-01-1-0431. This work was done while at the Dept. of Mathematics, University of Pennsylvania, Philadelphia, PA, USA.

[‡]Partially supported by a 2001–2004 U.S. DoD National Defense Science and Engineering Graduate (NDSEG) Fellowship and by ONR Grant N00014-01-1-0795.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM'03, August 25–29, 2003, Karlsruhe, Germany.
Copyright 2003 ACM 1-58113-735-4/03/0008 ...\$5.00.

Keywords

Interdomain routing, path-vector protocols, Border Gateway Protocol (BGP), Stable Paths Problem (SPP), routing-policy languages

1. INTRODUCTION

The Border Gateway Protocol (BGP) is the dynamic routing protocol used to connect autonomously administered networks on the Internet [12, 21, 24]. This contrasts with other, more familiar IP-routing protocols such as OSPF and IS-IS, whose main task is to establish and maintain connectivity *within* a single administrative domain [14].

BGP is unique among IP-routing protocols in that routing is determined using semantically rich routing policies. It is important to note that the languages and techniques for specifying BGP routing policies are not actually a part of the protocol. The BGP specification (RFC 1771 [21]) merely describes the low-level binary formats of BGP update messages, the intended meaning of the fields included in update messages, and the correct operation of a BGP-speaking router. On the other hand, routing-policy languages have been developed by vendors and have evolved through interactions with network engineers in an environment lacking vendor-independent standards. Router vendors typically provide hundreds of special commands for use in the configuration of BGP policies. In addition, BGP communities (RFC 1997 [3]) allow policy writers to selectively attach tags to routes and use these to signal policy information to other BGP-speaking routers. Routing policies can then condition their behavior on the presence or absence of specific community values. These developments have more and more given the task of writing BGP configurations aspects associated with open-ended programming. 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.

However, this rich expressiveness has come with hidden risks. The interaction of locally defined routing policies can lead to unexpected global routing anomalies such as non-deterministic routing and protocol divergence [10, 25]. If the interacting 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 autonomous system to implement “cold-potato routing” can cause protocol divergence in a neighboring autonomous system [4, 18]. We suspect that such problems will only become more common as BGP

continues to evolve with richer policy expressiveness. For example, extended communities [20] provide an even more flexible means of signaling information within and between autonomous systems than RFC 1997 communities do. At the same time, applications of communities by network operators are evolving to address complex issues of interdomain traffic engineering [2].

We believe that the root cause of “BGP-configuration problems” is a lack of design for the policy languages that are used to configure this protocol. 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 for path-vector protocols that 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 attempt to do this in a general way that is not constrained by the details of BGP. In this way, 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 [22], Telephony Routing over IP (TRIP) [23], and of various proposals for interdomain routing of optical paths [19, 26].

1.1 Overview of the Design Space

We feel that our main contribution is in the identification of the *design goals* of policy languages for path-vector protocols. In addition, we formalize these goals and path-vector implementations in a way that allows inherent trade-offs to be rigorously characterized.

We identify six important design goals for any path-vector 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 relationships imposed by the pervasive economic roles of customer, provider, and peer [15, 16]. The challenge then is to design policy languages that are as expressive as possible without sacrificing other design goals.

Robustness. We require predictability, *i.e.*, that any non-determinism in routing policies is not the result of unwanted policy interactions, and that a routing solution always exists and is 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.

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. For us, generalized autonomy will mean the ability to define a partition on routes and freely rank these partitions 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 oper-

ators would have to continually “tweak” their policies to compensate for unseen changes made to policies elsewhere.

We present one notion of autonomy specific to 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.

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 *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 similar methods. Of course, the simpler and 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 [17]). 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 partial exposure of just enough information to ensure robustness while at the same time allowing for a sufficient amount of information hiding.

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. Section 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.

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 2.3 defines the intended role of policy languages in path-vector-system configuration.

The notions of expressiveness and robustness are formalized in Sections 4 and 5. For both we employ the Stable Paths Problem (SPP) [10] as a semantic model of path-vector systems. We identify one class of robust systems as our target for expressiveness (Definition 19 and Theorem 5). Autonomy and transparency are formalized in Sections 6.1 and 6.2. Global consistency is discussed in Section 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

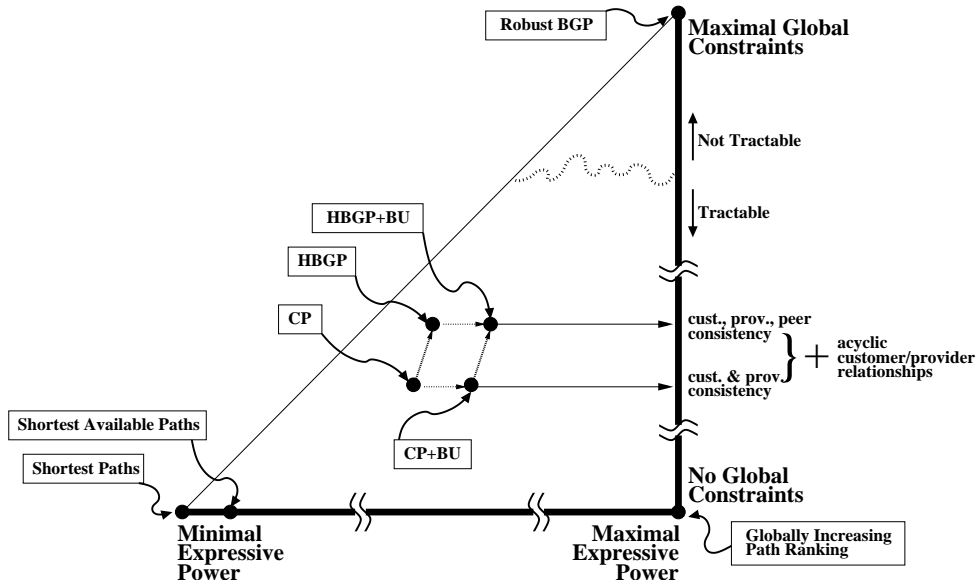


Figure 1: Design space for robust and transparent path-vector systems.

either autonomy of neighbor ranking or transparency (or both) (Theorem 7).

- 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 8).

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.

Figure 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 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-Path” 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 3.) We take “maximal” expressiveness to be the expressive power of a natural class of robust systems that we define in Section 5.3. The system “Globally Increasing Path Ranking” achieves maximal expressiveness with no global constraint, but it sacrifices other desirable properties (Section 6.3). 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) [1] in the manner suggested in [7] to accomplish this. Many practical issues make this scenario unlikely; furthermore, it was shown in [8] that, in the worst case, checking various global-consistency constraints is *NP-hard*.

The Hierarchical BGP systems (inspired by [5, 6]) provide examples from today’s commercial Internet. The system CP is a BGP-like system where 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 in Section 8. In the real world, we note that Internet economics seems to be sufficient in ensuring that networks behave in close approximation to the rules enforced by the above local and global constraints.

Due to a lack of space, all proofs have been deferred to the technical-report version of this paper [9].

2. 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.

2.1 Dynamics of Path-Vector Routing

We first briefly discuss the intended dynamics of routing using a path-vector system, as this motivates the system components we define in our framework.

Informally, let each node in the network be a protocol-speaking router responsible for its autonomous domain. A node advertises destinations in its network to its neighbors, and they further transmit this information to their neigh-

bors, *etc.* Whenever a router gets new information about a destination, it determines the best route to that destination given all the up-to-date information it has collected. We expect that routers will influence these decisions by modifying route attributes. This can be done on *export*, when routes are advertised to neighbors (or possibly filtered out altogether), or on *import*, when data is collected and stored for decision-making.

Therefore, we assume that there is some data structure to store and exchange route information, and that transformations to these data structures are made on import and export as dictated by routers' policy configurations. The exchange of these data structures between neighbors as described above will eventually permeate the network with knowledge about the various destinations *originated* by routers. Comparing these data structures gives a "best" route to a destination.

2.2 Formal Definition of Path-Vector Systems

As we develop this definition, 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 μ_{bgp} .

2.2.1 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 notify its neighbors of the path so that they can do the same.

The path-vector-system specification includes a description of the components in a path descriptor and a map that ranks them in 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. 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 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 1. Although the mechanics of determining "best" routes will be discussed in Section 2.6, 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

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

where \mathbb{N} is the set of natural numbers, $\text{Seq}(\mathbb{N})$ is the set of finite sequences of natural numbers, and \mathcal{C} is the set $\{\text{red, blue, green}\}$. 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 [3].

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:

$$\begin{aligned} & l > l'; \text{ or} \\ & l = l', m < m'; \text{ or} \\ & l = l', m = m', n < n'. \end{aligned}$$

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.

2.2.2 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. The policies would ideally 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 define the goal of the path-vector protocol; *e.g.*, if policies can only add a positive value to a path-cost attribute that solely determines path rank, the path-vector system implements lowest-cost-path routing.

Formally, let elements of the function space $2^{\mathcal{R}} \rightarrow 2^{\mathcal{R}}$ 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 2. Let the triple $\mathcal{C} = (\mathbb{L}^{in}, \mathbb{L}^{out}, \circ)$ be the *local-constraints* portion of the path-vector-system specification. \mathbb{L}^{in} and \mathbb{L}^{out} are predicates on import and export policy functions, respectively. If $\mathbb{L}^{in}(f)$ or $\mathbb{L}^{out}(f)$ holds, then f is a *legal* local-policy function. Furthermore, we assume that if either $\mathbb{L}^{in}(f)$ or $\mathbb{L}^{out}(f)$ holds, then f satisfies:

- (1) for each $X \subseteq \mathcal{R}$, if $|X| = 1$ then $|f(X)| \leq 1$;
- (2) for each X , $f(X) = \bigcup_{r \in X} f(\{r\})$; and
- (3) for each r_1, r_2 , if $f(\{r_1\}) = \{r_2\}$, then $dest(r_1) = dest(r_2)$.

0 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 $o(X)$ 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 $L_{\mu bgp}^{in}(f)$ and $L_{\mu bgp}^{out}(f)$ hold if and only if f satisfies conditions (1)–(3) above as well as

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

The only path descriptors which may be originated by nodes are those with an empty AS path and a default local preference of 0, so we let $o_{\mu bgp}(X)$ be true if and only if $(d, l, P, n, S) \in X$ implies $l = 0$ and $P = \epsilon$, the empty path.

2.2.3 Application of Policies

Although import and export policies allow router operators to configure their routers, it is important to recognize that the router (or the protocol itself) actually implements the application of 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

Definition 3. 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 $(\mathbb{N} \times \mathbb{N} \times (2^{\mathcal{R}} \rightarrow 2^{\mathcal{R}}) \times 2^{\mathcal{R}}) \rightarrow 2^{\mathcal{R}}$; 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. If u and v are nodes, f is u 's export policy function for v , and X the set of path descriptors at u , then we let

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

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 $t_{\mu bgp}^{out}(u, v, f, X)$) and g is v 's import policy function for u , then we let

$$t_{\mu bgp}^{in}(v, u, g, Y) = \{g(r) \mid r \in Y, P \text{ is a simple path}\}.$$

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

2.2.4 Path-Vector System

Definition 4. 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 1–3.

2.3 Policy Languages and Configurations

Of course, policy writers don't 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 5. 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 $\mathcal{M}(p) = (F^{in}, F^{out}, F^{orig})$ of functions. The policy functions $F^{in}(v, u)$ and $F^{out}(v, u)$ are called the *import* and *export policy functions* at v for u , respectively. If u and v are node identifiers, then $L^{in}(F^{in}(v, u))$ and $L^{out}(F^{out}(v, u))$ hold whenever these policy functions are defined. Finally, the function F^{orig} maps node identifiers v to finite subsets of \mathcal{R} such that $o(F^{orig}(v))$ holds whenever $F^{orig}(v)$ is defined.

Running Example, Part 4. For our running example, we define a simple policy language $PL_{\mu bgp}$. A policy configuration in this language is a list of *declarations* of the form

$$\begin{aligned} \text{export from } v \text{ to } W & : \text{rule}; \text{ or} \\ \text{import at } v \text{ from } W & : \text{rule}; \text{ or} \\ \text{originate from } v & : (d, 0, \epsilon, v, S), \end{aligned}$$

where 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*:

$$\begin{aligned} C_1 & \implies A_1 \\ C_2 & \implies A_2 \\ & \vdots \\ C_n & \implies A_n \end{aligned}$$

where each C_i is a boolean predicate over the attributes of a path descriptor 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 below.

2.4 Instances of Path-Vector Systems

Definition 6. An *instance* of a path-vector system PV with respect to a policy language PL (or an *instance* of (PV, PL)) is a pair $I = (G, F)$, where $G = (V, E)$ is an undirected graph, called the *signaling graph*, and the *configuration function* F maps nodes $v \in V$ to policy configurations $F(v) = p$ in the policy language PL . When $F(v) = p$ and $\mathcal{M}(p) = (F^{in}, F^{out}, F^{orig})$, we require that $F^{orig}(v)$ is defined, and that, for every $\{v, u\} \in E$, both $F^{in}(v, u)$ and $F^{out}(v, u)$ are defined. We will assume that the vertex set V is a subset of \mathbb{N} .

Definition 7. Given two instances $I = (G, F)$ and $I' = (G', F')$ 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 F' is equal to F when restricted to G' . For example, given any instance $I = (G, F)$ and G' , a subgraph of G , the instance $I' = (G', F)$ is a sub-instance of I .

Running Example, Part 5. We now give an instance of $(PV_{\mu bgp}, PL_{\mu bgp})$. Figure 2 shows a graph with five vertices. A policy configuration covering all 5 nodes in this graph is shown in Figure 3.

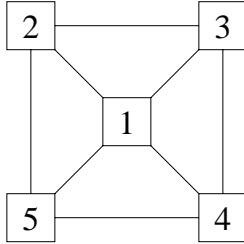


Figure 2: A simple 5 node graph.

2.5 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 descriptors 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 8. 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 $X \subseteq \mathcal{R}$,

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

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 2 only need to hold for the functions $\{F_{(v,u)} \mid \{v, u\} \in E\}$; however since t^{out}

```

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

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

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

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

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

```

Figure 3: Example policy configuration in $PL_{\mu bgp}$.

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 $P = vx_1 \dots x_k w$ 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 $P = vx_1 \dots x_k w = vP'$ and let $r(vP', r_w) = F_{(v,x_1)}(r(P', r_w))$.

Definition 9. 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. Formally:

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

2.6 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 10. A *path assignment* ρ is a mapping from V to $2^{\mathcal{R}}$. Given a path assignment ρ , define the set $C(\rho, v)$ of

candidates at node v to be

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

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 11. For $X \subseteq \mathcal{R}$, define the set $\min(X)$ as

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

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 $\text{sol}(I)$ be set of solutions for I . Note that it may be the case that $\text{sol}(I) = \emptyset$. If for every $d \in \mathcal{D}$, every instance I of PV , and every $\rho \in \text{sol}(I)$ there is at most one $r \in \rho(v)$ with $\text{dest}(r) = d$, then PV is a *single-path vector system*. Otherwise, we say that PV is a *multi-path vector system*.

Running Example, Part 6. The unique solution $\rho_{\mu bgp}$ to the instance from part 5 of our running example is shown in Table 1. Note that the sub-instance obtained by deleting the edge $\{1, 5\}$ from the graph has no solution.

v	$\rho_{\mu bgp}(v)$
1	$\{(d, 0, \epsilon, 1, \emptyset)\}$
2	$\{(d, 50, (1), 1, \{red\})\}$
3	$\{(d, 500, (4, 5, 1), 4, \{green\})\}$
4	$\{(d, 25, (5, 1), 5, \{green\})\}$
5	$\{(d, 1, (1), 1, \{green\})\}$

Table 1: Unique solution for our running example.

3. PATH-VECTOR-SYSTEM EXAMPLES

We briefly discuss two points in the design space that were mentioned in the overview.

Example 1. (Shortest Paths) Let $\mathcal{R}_{sp} = \mathcal{D}_{sp} \times \mathbb{N} \times \text{Seq}(\mathbb{N})$; the second component is a non-negative length associated with the path listed in the third component. The length is the sole determiner of rank such that lower-length paths are preferred. We permit nodes to increment the length of a path on import or export, so that $L^{in} = L^{out} = L_{sp}$ where L_{sp} is the constraint that there exists $n \in \mathbb{N}$, $n > 0$ such that for all $d \in \mathcal{D}_{sp}$, $m \in \mathbb{N}$, $P \in \text{Seq}(\mathbb{N})$, we have $f(\{(d, m, P)\}) = \{(d, m + n, P)\}$.

We define the export policy application $t_{sp}^{out}(u, v, f, X)$ to produce the set

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

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

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

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

Remark 2. Note that by replacing $\text{Seq}(\mathbb{N})$ with \mathbb{N} we could model “distance vector” protocols similar to RIP [13]. However, we will restrict our attention to those systems that do not allow signaling paths of arbitrary length.

Example 2. (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 $L_{sap}(f)$ means that there exists $n \in \mathbb{N}$, $n > 0$ such that for all $d \in \mathcal{D}_{sp}$, $m \in \mathbb{N}$, $P \in \text{Seq}(\mathbb{N})$, either $f(\{(d, m, P)\}) = \emptyset$ or $f(\{(d, m, P)\}) = \{(d, m + n, P)\}$.

4. EXPRESSIVENESS

To rigorously capture the *expressive power* of path-vector systems, we use a variant of the Stable Paths Problem (SPP) [10] as a semantic domain. After reviewing the SPP framework, 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.

4.1 The Stable Paths Problem (SPP)

Definition 12. 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 13. A *path assignment* for S is any mapping π from V to subsets of \mathcal{P} such that $\pi(v) \subseteq \mathcal{P}^v$. The set of paths candidates(u, π) represents all permitted paths at u that can be formed by extending the paths assigned to the neighbors of u . For $u = u_0$, this set is candidates(u, π) = $\{u\}$, and for $u \neq u_0$, the set candidates(u, π) is

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

A path assignment π is a *solution* for an SPP if for every node u we have $\pi(u) = \min(\lambda^u, \text{candidates}(u, \pi))$.

Remark 3. The definition for SPP given here is a bit more general than that of [10] 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.

4.2 Mapping Path-Vector Systems to SPPs

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 14. Define $I(w, r_w)$ to be a *restriction* of instance I where the only descriptor originated is r_w at node w . For each, define the *corresponding SPP instance* $S_{(I, w, r_w)}$ as described below, and let $\mathcal{S}(I)$ be the set of SPP instances corresponding to all the possible restrictions 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 $v \in V$, we define the ranking function as follows: $\lambda_{(I, w, r_w)}^v(P_1) \leq \lambda_{(I, w, r_w)}^v(P_2)$ 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 $\lambda_{(I,w,r_w)}^v(P_1) = \lambda_{(I,w,r_w)}^v(P_2)$ 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 1. *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 $I(w, r_w)$.

THEOREM 2. *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 $S_{(I,w,r_w)}$.

THEOREM 3. $\pi_{\rho_\pi} = \pi$ and $\rho_{\pi_\rho} = \rho$.

Running Example, Part 7. An SPP associated with our running example is presented in Figure 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; so, this figure can be taken to represent an entire equivalence class of SPPs.

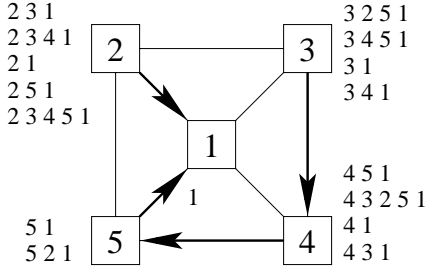


Figure 4: SPP for running example.

4.3 Expressive Power of a Path-Vector System

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) 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 15. 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 16. 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. It is not hard to show that

$$\mathcal{M}(PV_{sp}) \subset \mathcal{M}(PV_{sap}) \subset \mathcal{M}(PV_{\mu bsp}).$$

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.

5.1 Definition of Robustness

Definition 17. 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 18. In a similar manner, we can define robustness of SPP instances. Define the set \mathcal{RSPP} as

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

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 1. *For every (PV, PL) , if $\mathcal{M}(PV, PL) \subseteq \mathcal{RSPP}$, then there exists an $\mathcal{E}(S) \in \mathcal{RSPP}$ such that $\mathcal{E}(S) \not\subseteq \mathcal{M}(PV, PL)$. In other words, no path-vector policy system can capture exactly all robust systems.*

5.2 A Natural Class of Robust Systems

Definition 19. The SPP S is *almost-partially ordered* if \odot_S is reflexive, transitive, and obeys the following rule:

SPP Antisymmetry: $P_1 \odot_S P_2$ 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.

THEOREM 4. *If an SPP instance S is almost-partially ordered, then it is robust.*

THEOREM 5. *If $\mathcal{M}(PV, PL) \subseteq \mathcal{APOSPP}$, then the path-vector policy system (PV, PL) is robust.*

Remark 5. The above theorems give the broadest-known sufficient condition for robustness and are consistent with the results in [10].

5.3 Increasing Path-Vector Systems

Definition 20. The SPP instance S is *increasing* if, for all edges $\{u, v\}$ with path Q permitted at u and path vQ permitted at v , that $\lambda^u(Q) < \lambda^v(vQ)$. (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.

THEOREM 6. $\mathcal{APOSPP} = \mathcal{ISPP}$.

Ideally, we would like to construct a (PV, PL) pair such that $\mathcal{M}(PV, PL) = \mathcal{ISPP}$, thus obtaining expressiveness and robustness. There are two natural ways in which we might modify $PV_{\mu bgp}$ in order to do this; we give these in the following abbreviated examples.

Example 3. PV_{up} further constrains policy functions to satisfy $\omega(f(\{r\})) > \omega(\{r\})$ for every r , and leaves local preference unchanged in the protocol transformations (instead of setting it to 0).

Example 4. PV_{force} modifies both protocol transformations so that they filter out descriptors whose ω -rank does not increase under the application of the policy function in question.

We note that $\mathcal{M}(PV_{up}) = \mathcal{M}(PV_{force}) = \mathcal{ISPP}$. As we see in the next section, each of these systems lacks some desirable property, a conflict which is in fact unavoidable (Theorem 7).

6. AUTONOMY AND TRANSPARENCY

6.1 Autonomy

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

We first define a general notion of autonomy. Suppose that we have a collection of predicates on path descriptors, defined via simple boolean connectives and simple atomic predicates on path-descriptor attributes, such that exactly one predicate holds for each descriptor; call this a *partition* of path descriptors. Furthermore, suppose there is a partial order on the predicates, thus inducing an ordering on path descriptors. A path-vector policy system is *autonomous with respect to a partition* if there exists a legal import policy that ranks routes consistent with the partition's induced ordering on descriptors. (For example, a policy writer may want to prefer routes marked with a specific attribute value over routes that are not marked with that value. To do so, the system must be autonomous with respect to a collection of predicates testing 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.

Although possible in our framework, we will not formalize this general notion of autonomy in this paper. Instead, we 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. (This is compatible with the above general definition; classification by type of neighbor can be thought of as the form of the predicates on path descriptors.)

Definition 21. 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, \dots, 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 all j , v always prefers routes sent from partition C_j over those sent from partition C_{j+1} .

The system PV_{up} in Example 3 does not support autonomy of neighbor ranking. However the system PV_{force} in Example 4 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.

6.2 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 22. 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 6. The system PV_{force} is not transparent, but the systems PV_{up} and $PV_{\mu bgp}$ are.

6.3 A Design Trade-off

THEOREM 7. *If (PV, PL) is any path-vector policy system with $\mathcal{M}(PV, PL) = \mathcal{APOSPP}$, then either (PV, PL) does not support autonomy of neighbor ranking or PV is not transparent, or both.*

7. GLOBAL CONSTRAINTS

Theorem 7 shows that the expressive power of \mathcal{APOSPP} can be reached only if a path-vector policy system gives up either autonomy or transparency. 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 23. A *globally constrained* path-vector policy system is a triple (PV, PL, κ) , where κ is a global constraint for (PV, PL) . I is a *legal instance* of (PV, PL, κ) if I is an instance of (PV, PL) and $\kappa(I) = \text{TRUE}$.

Definition 24. Let $\mathcal{M}(PV, PL, \kappa)$ be the set $\{\mathcal{E}(S) \mid S \in \mathcal{S}(I) \text{ for some legal } (PV, PL) \text{ instance } I\}$.

Definition 25. Define the constraint κ_{ppo} as

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

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

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

THEOREM 8. *Suppose the global constraint κ is robust for a transparent (PV, PL) allowing autonomy of neighbor ranking such that $\mathcal{M}(PV_{sp}) \subseteq \mathcal{M}(PV, PL, \kappa)$ (i.e., at least as expressive as shortest paths). Then κ must be non-trivial.*

8. AN APPLICATION: CLASS-BASED PATH-VECTOR POLICY SYSTEMS

The Hierarchical-BGP points in the design space (HBGP, etc.), motivated by [5, 6], are examples of a general class of transparent systems where some type of autonomy of neighbor ranking is relevant: route transformations depend on the partition of neighbors into *classes*. We will refer to systems that use a generalized version of such a policy language as *class-based systems*. Theorem 8 tells us that such systems require a nontrivial global constraint; in this section we sketch design guidelines for these systems.

8.1 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 [5, 6]). 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 $PV_{\mu bgp}$, 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 preferred class. Therefore, we extend the path descriptor r , following [5], 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 7. 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 [5] discussed these same issues.

The components of the path-vector system PV_{cb} that we fix for class-based applications appears in Figure 5. Note that L_{cb}^{in} and L_{cb}^{out} guarantee that the level attribute is non-decreasing and that t_{cb}^{out} guarantees that local preference is not shared. When ranking, a lower 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 22.

\mathcal{R}_{cb}	$= \mathcal{D}_{cb} \times \mathbb{N} \times \mathbb{N} \times \text{Seq}(\mathbb{N})$
\mathcal{U}_{cb}	$= \mathbb{N} \times \mathbb{Z}$ (lexically ordered)
$dest_{cb}(d, g, l, P)$	$= d$
$\omega_{cb}(d, g, l, P)$	$= (g, -l)$
$O_{cb}(X)$	$= (r \in X) \Rightarrow (\exists d \in \mathcal{D}_{cb}, m \in \mathbb{N} \text{ such that } r = (d, 0, 0, m))$
$L_{cb}^{in}(f)$	$= (((d', g', l', P') = f(d, g, l, P)) \Rightarrow (g \leq g' \wedge P = P')) \wedge \text{MC}(f)$
$L_{cb}^{out}(f)$	$= (((d', g', l', P') = f(d, g, l, P)) \Rightarrow (g \leq g' \wedge P = P')) \wedge \text{MC}(f)$
$t_{cb}^{in}(u, v, f, X)$	$= \{(d, g, l, P) \in f(X) \mid P \text{ is a simple path}\}$
$t_{cb}^{out}(u, v, f, X)$	$= \{(d, g, 0, uP) \mid (d, g, l, P) \in f(X)\}$

Figure 5: Components of PV_{cb} .

8.2 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 relationships. Let $C = \{C_1, C_2, \dots, C_c\}$ 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. Here, a node v with neighbors u, w, x has assigned classes C_k, C_i, C_j to these neighbors, respectively.

Class assignments might require some consistency, e.g., that “customer” and “provider” assignments occur in consistent pairs; such requirements are expressed by the *cross-class matrix* $X = \{0, 1\}_{c \times c}$. For any pair of nodes $u, v \in V$, if $C^v(u) = C_i$, then $X_{ij} = 1$ if $C^u(v)$ is permitted to be C_j ; otherwise, $X_{ij} = 0$.

Let (\bullet) be the set of *order operators*, e.g., $=, <, \leq$, etc., and \top , which means “any relationship,” so that $z_1 \top z_2$ is true

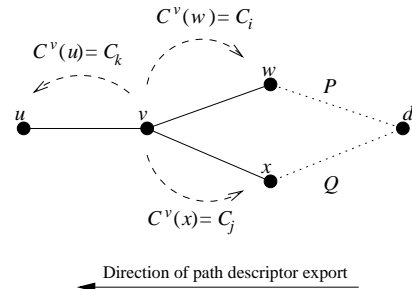


Figure 6: Class assignments to neighbors of node v and paths to a destination node d .

for any z_1, z_2 in the same ordered set. Relative preference between classes will be described by the *preference matrix* $W = (\bullet)_{c \times c}$ so that if $\bullet = W_{ij}$, then $\omega(r_i) \bullet \omega(r_j)$ for path descriptors r_i, r_j imported from neighbors of classes C_i, C_j , respectively; *e.g.*, in Figure 6, if $W_{ij} <$, then node v should prefer the path P over the path Q . The policy-language compiler can enforce this as a constraint on local-preference-attribute values set by import policies.

Scope will be described by the *level matrix* $M = ((\bullet) \cup \{\perp\})_{c \times c}$. For any node v and neighbors w, u with $C^v(w) = C_i$ and $C^v(u) = C_k$, if $M_{ik} = \perp$ then for any path descriptor r imported from w , $F^{out}(v, u)(\{r\}) = \emptyset$. This setting is used to prevent the exchange of routes between classes altogether (filtering); *e.g.*, in Figure 6, if $M_{ik} = \perp$, then v would not export to u any routes it learned from w . Other scoping conditions can be described by allowing or enforcing a change in the level attribute; *e.g.*, 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. This situation can be sketched using our example Figure 6: Formally, for any node v and two neighbors w, u with $C^v(w) = C_i$ and $C^v(u) = C_k$, assume there is a path P from w to some destination d . Let r_w be the path descriptor at v for the path vP , and let r_u be the path descriptor for the path vP exported to u , *i.e.*, $\{r_w\} = F^{out}(v, u)(\{r_w\})$. The policy compiler should enforce through constraints on level-attribute values set in export policies that, if $\bullet = M_{ik}$, then $g(r_w) \bullet g(r_u)$.

Example 5. For HBGP+BU, let $C = \{C_1, C_2, C_3\}$, where C_1 can be interpreted “customer,” C_2 as “peer,” and C_3 as “upstream provider.” X should enforce consistent customer-provider and peer-peer relationships; W should enforce that customer routes are preferred over peer routes, and both are preferred over upstream routes; M should enforce that customer routes are shared with all neighbors, and that peer and upstream routes are only shared with customers. In addition, M should permit nodes to flag routes as backup routes so that they are less preferred even if relative preference rules would dictate otherwise. The resulting matrices X, W , and M are as follows.

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \quad W = \begin{bmatrix} \top & < & < \\ > & \top & < \\ > & > & \top \end{bmatrix}$$

$$M = \begin{bmatrix} \leq & \leq & \leq \\ \leq & < & < \\ \leq & < & \perp \end{bmatrix}$$

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. The tuples will honor the scope and relative preference rules described by CD if the compiler does the following at each node v for its specified 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 $\omega(r_u) \bullet \omega(r_w)$.

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 8. Class-based systems are not autonomous with respect to neighbor ranking; however, the system is autonomous with respect to predicates on path descriptors that test level-attribute values and the class of neighbor exporting the route.

8.3 Class-Based Global Constraints

Let the class-consistency constraint C be defined as

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

We are left to define a robustness check; let $\kappa_{cb} = C \wedge J$, where J is a constraint such that κ_{cb} is robust for PV_{cb} with respect to some PL of the form described above.

The constraint κ_{cb} will be robust for (PV_{cb}, PL) if J can guarantee that realizable paths are almost-partially ordered. Intuitively, J must enforce additional conditions that build on the partial ordering on paths derived from the scope and relative-preference rules in CD . The level attribute plays an important role in the ordering of paths because this attribute flags routes that do not obey standard preference and filtering rules.

Definition 26. A *level-equality cycle* is a cycle of nodes $v_1, v_2, \dots, v_k, v_{k+1} = v_1$ in a class-based instance such that when v_i exports to v_{i-1} a path descriptor imported from v_{i+1} , the level attribute can stay the same.

We can then obtain the following sufficient condition for robustness. It is not necessarily the minimal global constraint to guarantee robustness, but it can provide a good starting point in design.

THEOREM 9. *Let CD be a class description for a policy language PL and let C be the class-consistency constraint. If $J(I)$ implies that all instances I of (PV, PL) contain no level-equality cycles, then $(PV_{cb}, PL, \kappa_{cb})$, where $\kappa_{cb}(I) = C(I) \wedge J(I)$, is robust.*

We note that for certain systems, if J implies that instances contain no level-equality cycles, then J must check an unreasonable number of possibilities.

Example 6. For the system HBGP+BU, J reduces to a check that no customer-provider cycles exist: A simple case-by-case analysis of possible class assignments in a level-equality cycle, given the constraints in matrices C and M , shows that the only level-equality cycles possible are cycles in the customer-provider relationship graph. (See the technical-report version [9] for the full proof.) Checking for these is tractable; furthermore, the basic economics of the current commercial Internet naturally enforces κ_{cb} .

9. CONCLUSIONS AND OPEN PROBLEMS

We have defined the path-vector system framework: we identified and formalized dimensions of the protocol design space in a way that highlights the role of policy languages and individual protocol components.

Several issues that we discussed require additional work. First, either Conjecture 1 must be proven or a broader sufficient condition for robustness should be found. Second, the power of class-based systems must be investigated further; in particular, the robustness check presented in Theorem 9 can probably be tightened. Third, while we justify the inclusion of global constraints in protocol design, we do not discuss how they are enforced. Distributed algorithms, supplementary protocols, or economic incentives could check global consistency. We can also ask what level of expressiveness can be achieved by an autonomous, transparent, and robust system with an imposed global constraint that can be checked by one of the above methods in polynomial time. Finally, additional useful degrees of autonomy should be identified and analyzed (perhaps in the context of specific routing applications).

We have focused on the static semantics of path-vector systems rather than their dynamic behavior. 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. Furthermore, the static semantics of a path-vector system are independent of the algorithm used to find solutions; distributed approaches to this problem are of particular interest.

We have focused on the signaling of routes without discussion of how this corresponds to forwarding in the data plane. For example, in BGP, the signaling graph of Internal BGP (IBGP) need not have any relationship to the forwarding graph (IGP forwarding). Several routing anomalies have been described [11] that related to this independence in BGP. In general, there will be some interaction between the signaling graph, the physical network supporting this signaling, and the paths in the data plane which are controlled by the paths in the signaling plane. We need a general theory that describes this interaction for path-vector protocols.

10. REFERENCES

- [1] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizar. Routing Policy Specification Language (RPSL). RFC 2280, 1998.
- [2] O. Bonaventure and B. Quoitin. Common Utilizations of BGP Community Attribute. Manuscript, 2003.
- [3] R. Chandra, P. Traina, and T. Li. BGP Communities Attribute. RFC 1997, 1996.
- [4] Cisco Field Note. Endless BGP Convergence Problem in Cisco IOS Software Releases. October 2001. <http://www.cisco.com/warp/public/770/fn12942.html>
- [5] L. Gao, T. G. Griffin, and J. Rexford. Inherently Safe Backup Routing with BGP. In *Proc. IEEE INFOCOM 2001*, 1:547–556, April 2001.
- [6] L. Gao and J. Rexford. Stable Internet Routing without Global Coordination. In *Proc. ACM SIGMETRICS*, pages 307–317, June 2000.
- [7] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W. Lee. An Architecture for Stable, Analyzable Internet Routing. *IEEE Network*, 13(1):29–35, 1999.
- [8] T. Griffin and G. Wilfong. An Analysis of BGP Convergence Properties. In *Proc. ACM SIGCOMM'99*, pages 277–288, September 1999.
- [9] T. G. Griffin, A. D. Jaggard, and V. Ramachandran. Design Principles of Policy Languages for Path Vector Protocols. Technical Report 1250, Yale University Department of Computer Science, 2003.
- [10] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, April 2002.
- [11] T. G. Griffin and G. Wilfong. On the Correctness of IBGP Configuration. In *Proc. ACM SIGCOMM'02*, August 2002.
- [12] B. Halabi. *Internet Routing Architectures*. Cisco Press, 1997.
- [13] C. Hendrick. Routing Information Protocol (RIP). RFC 1058, 1988.
- [14] C. Huitema. *Routing in the Internet*. Prentice Hall, 1995.
- [15] G. Huston. Interconnection, Peering and Settlements: Part I. *Internet Protocol Journal*, 2(1):2–16, March 1999.
- [16] G. Huston. Interconnection, Peering and Settlements: Part II. *Internet Protocol Journal*, 2(2):2–23, June 1999.
- [17] G. Huston. Scaling Interdomain Routing—A View Forward. *Internet Protocol Journal*, 4(4):2–16, December 2001.
- [18] D. McPherson, V. Gill, D. Walton, and A. Retana. BGP Persistent Route Oscillation Condition. Manuscript, 2002.
- [19] B. Rajagopalan, J. Luciani, and D. Awduche. IP Over Optical Networks: A Framework. Manuscript, 2003.
- [20] S. Ramachandra and D. Tappan. BGP Extended Communities Attribute. Internet Draft, 2001. Work in progress.
- [21] Y. Rekhter and T. Li. A Border Gateway Protocol. RFC 1771 (BGP version 4), 1995.
- [22] E. Rosen and Y. Rekhter. BGP/MPLS VPNs. RFC 2547, 1999.
- [23] J. Rosenberg, H. Salma, and M. Squire. Telephony Routing Over IP (TRIP). RFC 3219. January 2002.
- [24] J. W. Stewart. *BGP4, Inter-domain Routing in the Internet*. Addison-Wesley, 1998.
- [25] K. Varadhan, R. Govindan, and D. Estrin. Persistent Route Oscillations in Inter-domain Routing. *Computer Networks*, 32:1–16, 2000.
- [26] Y. Xu, A. Basu, and Y. Xue. A BGP/GMPSL Solution for Inter-domain Optical Networking. Manuscript, 2002.