Yale University
Department of Computer Science

Robustness of Class-Based Path-Vector Systems

Aaron D. Jaggard Vijay Ramachandran

YALEU/DCS/TR-1296
December 2004

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). A shortened form of this work has been published as a conference paper [10].

Robustness of Class-Based Path-Vector Systems

Aaron D. Jaggard* Vijay Ramachandran†

Abstract

Griffin, Jaggard, and Ramachandran [5] introduced a framework for studying design principles for path-vector protocols, such as the Border Gateway Protocol (BGP) used for inter-domain routing in the Internet. They outlined how their framework could describe Hierarchical-BGP-like systems in which routing at a node is determined by the relationship with the next-hop node on a path (*e.g.*, an ISP-peering relationship) and some additional scoping rules (*e.g.*, the use of backup routes). The robustness of these *class-based path-vector systems* depends on the presence of a global constraint on the system, but an adequate constraint has not yet been given in general. In this paper, we give the best-known sufficient constraint that guarantees robust convergence. We show how to generate this constraint from the design specification of the path-vector system. We also give centralized and distributed algorithms to enforce this constraint, discuss applications of these algorithms, and compare them to algorithms given in previous work on path-vector protocol design.

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). A shortened form of this work has been published as a conference paper [10].

*Dept. of Mathematics, Tulane University, New Orleans, LA, USA. E-mail: `adj@math.tulane.edu`. Partially supported by National Science Foundation (NSF) Grant DMS–0239996 and by ONR Grant N00014–01–1–0795.

†Dept. of Computer Science, Yale University, New Haven, CT, USA. E-mail: `vijayr@cs.yale.edu`. Partially supported by a 2001–2004 DoD National Defense Science and Engineering Graduate (NDSEG) Fellowship, by ONR Grant N00014–01–1–0795, and by NSF Grant ITR–0219018.

1 Introduction

The standard Internet inter-domain routing protocol, the Border Gateway Protocol (BGP), determines routes using independently configured policies in autonomously administered networks. Little global coordination of policies takes place between domains, or autonomous systems (ASes), because (1) ASes are reluctant to reveal details about internal routing configuration, and (2) BGP contains no reliable mechanism to permanently attach information to a route as it is shared throughout the network. However, without global coordination, interaction of locally configured policies can lead to global routing anomalies [1, 4, 7, 11, 13], *e.g.*, route oscillation and inconsistent recovery from link failures. Because the techniques used to configure policies and the protocol's specification have evolved separately, there is an inherent trade-off between, on one hand, maintaining rich semantic expressiveness available with current vendor-developed configuration languages and autonomy in policy configuration, and, on the other hand, guaranteeing that the protocol will converge robustly, *i.e.*, predictably, even in the presence of link and node failures. Griffin, Jaggard, and Ramachandran [5] showed that achieving all three of these design goals requires a non-trivial global constraint on the network, but they left open the question of how to identify and enforce the constraint.

This paper answers this question in the context of class-based path-vector systems. Path-vector systems, introduced in [5], provide a formal model for design and analysis of path-vector protocols and their policy-configuration languages. 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. The canonical example of such a system 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 this "Hierarchical BGP" system; the framework can also be used to build and analyze systems with complete autonomy and those that allow arbitrary next-hop preference routing. Furthermore, any protocol specification that can be described by a countable-weight, monotone path-vector algebra [12] can also be described by a class-based path-vector system [9].

In this paper, 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.

1.1 Related Work

Several papers have presented policy-configuration guidelines or design principles to prevent global routing anomalies in path-vector routing. Gao and Rexford [3] showed that route preferences and scoping consistent with the Hierarchical-BGP example mentioned above give stable path-vector routing without global coordination. Griffin, Shepherd, and Wilfong [6] defined an abstraction of path-vector routing and identified a sufficient condition for local policies that prevents policy-induced route oscillation; Griffin and Wilfong [8] used those conditions to give a simple path-vector routing algorithm that dynamically detects policy-induced route oscillations. In addition, Gao, Griffin, and Rexford [2] combined the results from these papers to modify a simplified version of BGP to perform stable back-up routing.

Building on this work, the papers by Griffin, Jaggard, and Ramachandran [5] and Sobrinho [12] developed formal models for path-vector routing so that properties of path-vector protocols can be studied without involving details of protocol dynamics or of actual networks. Both papers prove results about protocol convergence based on the design specification of the protocol itself. They present an application of their frameworks that generalizes systems like Hierarchical BGP (mentioned above), incorporating the policy guidelines from [2, 3, 6] into the design of protocol systems. This paper completes the analysis of this application: It answers the questions left open in [5] and gives results that can be used in the general design of protocols and policy-configuration languages.

2 The Class-Based Framework

Class-based systems, which generalize Hierarchical-BGP-like systems, are a type of *globally constrained path-vector policy systems* (PVPSes). A PVPS describes a protocol that nodes use to exchange path descriptors (instances of the protocol's data structure describing routes in the network). A PVPS specifies a *ranking function* ω from the set $\mathcal{R}$ of all path descriptors into some totally ordered set; if r and r' describe two different paths from a node v to a network destination, v will prefer the one described by r if $\omega(r) < \omega(r')$. The PVPS also defines allowable import and export *policy functions* that nodes use to modify path descriptors in order to affect their rank; these are usually compiled from policies written by network operators at each node in some high-level policy language. We will recall other relevant definitions and PVPS results as we proceed; for a fuller exposition of PVPSes see [5].

In a class-based system, a path descriptor r is a quadruple (d, g, l, P), where d is the forwarding destination of the path and P is the path itself. The attributes g and l (whose values in the descriptor r are denoted $g(r)$ and $l(r)$) are the *level* and *local preference*, respectively, of the route; these take values in $\mathbb{N}$. Level is shared between nodes but local preference is not. The ranking function depends on these attributes: $\omega(r) = (g(r), -l(r))$, with lexicographic ordering on the values of ω so that r will be preferred over r' if $g(r) < g(r')$ or if $g(r) = g(r')$ and $l(r) > l(r')$. Policies in class-based systems may filter routes, weakly increase the level attribute, and change the local preference as desired, but nothing else.

Class-based systems incorporate global *relative preference* and *scoping* rules using these two attributes; the former affect relative rank of paths imported from different classes of neighbors, while the latter specify the classes to which certain routes may be exported. In BGP, it is common practice today for a router to always prefer a route learned from a customer router over a route (to the same destination) learned from a peer router and for a router to always export routes learned from its customer(s) to its provider(s). The first is an example of a relative preference rule, the second of a scoping rule.

A class-based PVPS is almost entirely described by a *class description* $CD = (C,\ X,\ W,\ M)$, which contains a set of c classes and three $c \times c$ matrices that describe the relationship, preference, and scoping rules. The *classes* in $C = \{C_1,\ C_2,\ \ldots,\ C_c\}$, *e.g.* "customer" or "peer," are used to describe the relationship between one node and another. In an instance of a class-based system, each node v will assign a class $C^v(u)$ to each of its neighbors u; $C^v(u)$ indicates how v views the relationship between nodes u and v. 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 $X_{ij} = 1$. 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 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 $\omega(r_i) \bullet \omega(r_j)$. 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 abovementioned 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]

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.

[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.

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.

In our analysis here, 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 2.1 (**$\{0,1\}$-Description Matrices).**

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.

Example 2.2 (Hierarchical BGP with Back-up Routes). This is the canonical example of a class-based system, motivated by the design guidelines in [2, 3]. 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:

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

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

The corresponding $\{0,1\}$-matrices are:

$$\hat{W} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix},\quad \hat{M} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Omitting the ability to mark routes for backup use (and generally ignoring the level attribute) yields the simpler system of Hierarchical-BGP; both of these systems are discussed in [5].

Remark 2.3. A proof that this system robust (with the additional global constraint that there are no customer-provider cycles) can be found in [5]. We will return to this example to show that our more general results regarding robustness are consistent with what is already known about these systems.

3 Robustness and Global Constraints

Having reviewed the class-based framework, we now present the contributions of this paper in detail. In this section, we motivate and describe the constraint that guarantees robust convergence for protocols modeled by the framework; we prove its correctness and show how to generate it given only the class-based design specification of a protocol. Then we show how to check this constraint in two ways: by using a centralized algorithm (Sec. 4), and by using a distributed algorithm (Sec. 5) that additionally fixes detected constraint violations.

3.1 Robustness and Dispute Wheels

The major goal of a routing system is the assurance that every instance of a network and local policies will be *robust*, *i.e.*, that every legal instance of the system will have a unique routing solution as will all sub-instances (these might arise due to the failure of any set of links and nodes). The additional design goals of *autonomy*, *transparency*, and *expressiveness* were formally defined for PVPSes (and thus class-based systems) in [5]. One of the major results in that paper (Thm. 8, [5]) is that any robust, reasonably autonomous, transparent, and moderately expressive PVPS must have a *global constraint*—a predicate defined on instances of the PVPS whose truth value on an instance determines whether that instance is "legal"—that is not identically true. Class-based systems, by their definition, satisfy these design goals of PVPSes; thus they must include a non-trivial global constraint if they are to be robust. A sufficient global constraint for the Hierarchical-BGP systems in Ex. 2.2 is that the signaling graph does not contain any customer-provider cycles (*i.e.*, cycles in which each node views one of its neighbors in the cycle as a customer and the other as a provider) [3]. Our goal here is to find good global constraints for class-based systems in general. As a starting point, we use the broadest-known condition for robustness of path-vector systems, that of *dispute-wheel freeness*; this was presented in [6] and was then adopted into the PVPS framework in [5]. We now turn to a simplified definition of this condition, including just the details needed for class-based systems, after some notation.

Definition 3.1 (Descriptor for a Realizable Path). Let $P_n = v_n v_{n-1} \cdots v_0$ be a putative route in a path-vector instance, such that v_n has learned about P_n from its neighbor v_{n-1}, who learned about $P_{n-1} = v_{n-1} v_{n-2} \cdots v_0$ from its neighbor v_{n-2}, *etc.* This process of route exchange began with node v_0 originating a path descriptor r_0 and exporting it to v_1. At each step along the way, a subpath P_i of P_n was imported at node v_i. We call this a *realizable path* because it can be selected as a valid forwarding route from v_i.

Given an edge $\{u,\ v\}$ in the network, we define the *arc-policy function* for the signaling edge $(u,\ v)$ to be the application of u's export policy function $F^{out}_{(u,v)}$ for v followed by the application of v's import policy function $F^{in}_{(v,u)}$ for u. Before the descriptor is imported at v, the protocol updates the path attribute to reflect the added edge $(u,\ v)$, filters out any paths which contain loops, and hides the local-preference value used by u.[2] Formally, if R is a set of path descriptors known to u, then the arc-policy function for the signaling edge $(u,\ v)$ applied to R is

$$f_{(v,u)}(R) = F^{in}_{(v,u)}(\{(d, g, 0, vP) \\ \mid (d, g, l, P) \in F^{out}_{(u,v)}(R),\ vP \text{ is a simple path}\});$$

this is the set of path descriptors that v receives along the edge $(u,\ v)$. Using arc-policy functions, we can iteratively define the path descriptor associated with path P_n and its subpaths, assuming that it originated at v_0 with path descriptor r_0. Let

$$r(P_0) = r_0,$$

and for $i,\ 1 \leq i \leq n$, let

$$r(P_i) = f_{(v_i, v_{i-1})}(r(P_{i-1})).$$

Definition 3.2 (Dispute Wheel). An instance of a path-vector system contains a *dispute wheel* (see Fig. 1) if there is a set of nodes $\{v_0, v_1, \ldots, v_n\} \subset V$ such that:

1. node v_0 advertises a destination and nodes $v_1, \ldots, v_n$ attempt to find routes to that destination;
2. there exists a signaling path Q_i for each i, $1 \leq i \leq n$, from v_0 to v_i (the Q_i are not necessarily disjoint); let $Q_0 = Q_n$;
3. there exists a signaling path R_i from each v_i to v_{i+1} for all i, $1 \leq i \leq n$ (where $v_{n+1} = v_1$; the R_i are not necessarily disjoint); let $R_0 = R_n$;
4. Q_i and $R_{i-1}Q_{i-1}$ are realizable paths from node v_i to v_0 for all i, $1 \leq i \leq n$; and
5. $\omega(r(R_{i-1}Q_{i-1})) < \omega(r(Q_i))$ for all i, $1 \leq i \leq n$; *i.e.*, every node v_i prefers to reach v_0 by using the path $R_{i-1}Q_{i-1}$ through v_{i-1} instead of the path Q_i.

[2] In the general PVPS framework, policy application is explicitly defined using *policy application functions*; for class-based systems, these functions are responsible for the updating and filtering as well as applying the policies in the manner described.

The paths Q_i are *spokes* of the dispute wheel, the paths R_i constitute the *rim* of the wheel, and the nodes on the rim of the wheel are its *rim nodes*.

The importance of dispute wheels is shown by the following theorem.

Theorem 3.3 (Sec. V, [6] and Thm. 5, [5]). *Any dispute-wheel-free PVPS instance is robust.*

An *instance* of a PVPS is a network on which the PVPS protocol will run, combined with nodes' import and export policies that are permitted by the PVPS specification. Dispute wheels in instances of class-based systems have the following property, which we will use to detect and avoid dispute-wheels in this work.

Proposition 3.4 (Lemma 8.5, extended version of [5]). *The level-attribute values of all the path descriptors $r(Q_i)$ and $r(R_iQ_i)$ involved in a dispute wheel are equal.*

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

Definition 3.5 (Edge-Class Notation).

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 3.6 (Homogeneity and Heterogeneity). 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, $\mathbf{c}(e) \in \{C_{k_1}, C_{k_2}, \ldots\}$.

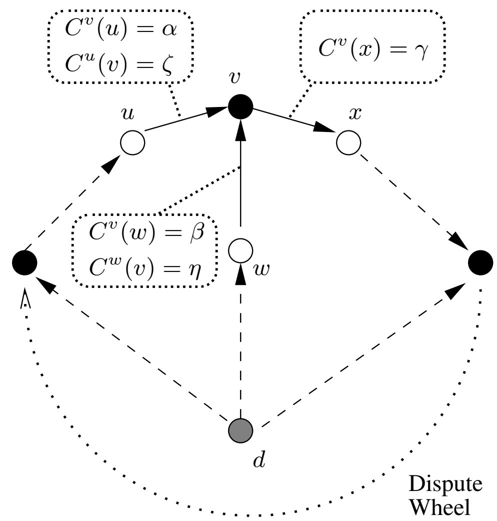

Figure 1: Active node v of a dispute wheel.

3.2 Generating a Global Constraint

PVPS solvability, and thus checking robustness, is *NP*-hard in general [6]; 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. Prop. 3.4 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 Thm. 3.3 tells us that precluding dispute wheels guarantees 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 Fig. 1, 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 Fig. 1 have arrows in the direction of signaling or export.

Prop. 3.4 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 Def. 3.2(5) (a necessary condition for a dispute wheel). In particular, any of the following conditions at one active node would preclude the dispute wheel in Fig. 1; 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 3.7 (Rim Necessary Condition). *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. (Sketch.) Condition 1 follows directly from Prop. 3.4. Condition 2 follows directly from Lemma 8.6 in the extended version of [5]. □

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. This was the constraint suggested in [5], but it 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 the constraint in [5] 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 3.8. $S = X\hat{M}$ (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 3.9 (Existence of Homogeneous Dispute Wheels). *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 3.10 (Homogeneous Dispute Wheels in HBGP). For the system in Ex. 2.2, 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. Thus, as shown in [3, 5, 6], 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 3.11 (Simple Heterogeneous Dispute Wheel). *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, 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 3.12 (Non-permutation Heterogeneity). *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, 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 3.13 (Reverse Heterogeneity). *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, 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 Prop. 3.11. □

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 3.14 (Dispute Predicate). 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 3.15 (Exact Condition for Dispute Wheel Creation). *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 Prop. 3.4); 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 Prop. 3.4); thus, it must assign the neighboring spoke node w a class $C^{v_i}(w) = C_\gamma$ such that

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

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

$$\hat{W}_{\gamma k_{i'}} \neq -1 \tag{2}$$

so that the rim preference is allowed. (1) and (2) 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

$$\exists C_\gamma \in \mathbf{m}^{-1}(C_{k_{i+1}}) : \hat{W}_{\gamma k_i'} \neq -1. \tag{3}$$

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 (3) 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 (3). 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 (3). 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. □

Thm. 3.15 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 3.16 (Class-Based Robustness). 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 Thm. 3.3, 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 3.17 (Generation of Robustness Constraint). 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.

4 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 3.16.

4.1 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 4.1 (Centralized Cycle Detection).

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.

Proposition 4.2 (Correctness). *Any cycle in G_L found by Alg. 4.1 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 3.16 (*i.e.*, it is a potential dispute wheel).

By Prop. 3.4, 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. □

Proposition 4.3 (Running Time). *Alg. 4.1 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 $O(\Delta|E|)$. □

4.2 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 submatrices 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 Alg. 3.17 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 (Alg. 4.1) 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 4.4 (Check for Next-Hop Preferences).

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 4.5 (Correctness of Next-Hop-Preferences Check). *Any cycle in G_L found by Alg. 4.4 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 Alg. 4.4 is similar to the graph G_L checked for cycles in Alg. 4.1; 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 Prop. 4.2 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 Alg. 4.4: $(v,\ u,\ x) \in T$ 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 $\mathbb{P}(\mathbf{c}(v,u),\ \mathbf{c}(v,x))$. $\square$

4.3 Algorithms in Previous Work

Sobrinho [12] presents an algebraic formalism for path-vector routing that is shown in [9] to be essentially equivalent to PVPSes. Sec. 6.3 in [12] gives a check for protocol convergence on a network given the abstract design specification of the protocol—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 [12] take time $O(c^3)$ and $O(c \cdot (|V| + |E|))$, where c is the number of classes, assuming that matrix W is consistent with a linear order on C. The performance of this algorithm compared to Alg. 4.1 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 3.16). 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 [12] 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: 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 [12] 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.

Algorithm 4.6 (Algebraic Robustness Check, adapted from [12]). 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 Alg. 3.17 to generate pairs of classes on which $\mathbb{P}$ holds. This takes $O(c^3)$ time.

3. For each w, $1 \leq w \leq w^*$, 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 Alg. 4.1. Then for each w, $1 \leq w \leq w^*$, 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.

5 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.

5.1 Distributed Cycle-Check

Our distributed algorithm (Alg. 5.1) 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 Alg. 5.1 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 5.1 (Distributed Edge Check). 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 $w \in \mathrm{in}(u)$, 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

$$\nexists(w, w') : (w' \in Y^u(w)) \wedge (w \in Y^u(w')) \qquad (4)$$

because two neighbors cannot both be depreferenced with respect to each other. If (4) 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 1 (Number of Tokens). *In Alg. 5.1, 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 5.2 (Termination). *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. 1 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 5.3 (Cycle Participation). *At the end of the algorithm, if $L_B \neq \emptyset$ at u, then $(u,\ v)$ is part of a cycle violating Constraint 3.16.*

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 5.4 (Number of Messages). *The algorithm sends either* 0 *or* 3 *messages per signaling-graph link.*

Proof. By Lem. 1, 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 Prop. 5.2 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 $[N, F]$ is sent) or 3 ($[N, F]$, $[N, B]$ 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).

Alg. 5.1 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 Prop. 5.2 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. Alg. 5.1 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.

5.2 Algorithms in Previous Work

The algorithm SPVP_3 in [8] is a distributed path-vector routing algorithm that detects local-policy-based routing oscillation 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 [8] 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 paper 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.

6 Conclusion and Future Work

In this paper, we have extended previous work on path-vector policy systems by focusing on class-based systems, which generalize Hierarchical-BGP and related protocols. In particular, we show 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 provide centralized and distributed algorithms to enforce it. However, our constraint is not likely to be the most general, tractable constraint for all path-vector systems. To date, class-based systems seem to be the only path-vector systems well-characterized enough that exact constraints for them can be proven, but studying other characterizations may yield constraints and enforcement mechanisms that guarantee robustness for a larger set of path-vector protocols. In addition, the question of how to efficiently run our distributed algorithm in parallel remains open: in particular, token-traversal paths from separate instances of the algorithm can probably be combined to find and fix all potential dispute wheels at once.

Acknowledgements

We are grateful to Joan Feigenbaum, Dana Angluin, Stanley Eisenstat, and Paul Hudak for helpful discussions and to the *ICNP'04* referees for their useful suggestions.

References

[1] Cisco Field Note. Endless BGP Convergence Problem in Cisco IOS Software Releases. Oct. 2001. `http://www.cisco.com/warp/public/770/fn12942.html`

[2] L. Gao, T. G. Griffin, and J. Rexford. Inherently Safe Backup Routing with BGP. In *Proc. INFOCOM 2001*, pp. 547–556, Apr. 2001.

[3] L. Gao and J. Rexford. Stable Internet Routing Without Global Coordination. *ACM/IEEE Trans. on Networking*, 9(6): 681–692, Dec. 2001.

[4] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W. Lee. An Architecture for Stable, Analyzable Internet Routing. *IEEE Network Magazine*, Jan./Feb. 1999.

[5] T. G. Griffin, A. D. Jaggard, and V. Ramachandran. Design Principles of Policy Languages for Path-Vector Protocols. In *Proc. ACM SIGCOMM'03*, pp. 61–72, Aug. 2003. Extended version: Yale Univ. Tech. Report YALEU/DCS/TR-1250, Apr. 2004. `ftp://ftp.cs.yale.edu/pub/TR/tr1250.pdf`.

[6] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The Stable Paths Problem and Interdomain Routing. *ACM/IEEE Trans. on Networking*, 10(2):232–243, Apr. 2002.

[7] T. G. Griffin and G. Wilfong. An Analysis of BGP Convergence Properties. In *Proc. ACM SIGCOMM'99*, pp. 277–288, Sept. 1999.

[8] T. G. Griffin and G. Wilfong. A Safe Path-Vector Protocol. In *Proc. IEEE INFOCOM 2000*, pp. 490–499, Mar. 2000.

[9] A. D. Jaggard and V. Ramachandran. Relating Two Formal Models of Path-Vector Routing. In *Proc. IEEE INFOCOM 2005*, to appear Mar. 2005.

[10] A. D. Jaggard and V. Ramachandran. Robustness of Class-Based Path-Vector Systems (conference version). In *Proc. ICNP'04*, pp. 61–72, Aug. 2004.

[11] D. McPherson, V. Gill, D. Walton, and A. Retana. BGP Persistent Route Oscillation Condition. newblock RFC 3345, Aug. 2002.

[12] J. L. Sobrinho. Network Routing with Path Vector Protocols: Theory and Applications. In *Proc. ACM SIGCOMM'03*, pp. 49–60, Aug. 2003.

[13] K. Varadhan, R. Govindan, and D. Estrin. Persistent Route Oscillations in Inter-domain Routing. *Computer Networks*, 32(1):1–16, Jan. 2000.