Policy Disputes in Path-Vector Protocols

Timothy G. Griffin F. Bruce Shepherd Gordon Wilfong
{griffin, bshep, gtw}@research.bell-labs.com
Bell Laboratories, Lucent Technologies

Abstract

The Border Gateway Protocol, BGP, is currently the only interdomain routing protocol employed on the Internet. As required of any interdomain protocol, BGP allows policy-based metrics to override distance-based metrics and enables each autonomous system to independently define its routing policies with little or no global coordination. Varadhan et al. [11] have shown that there are collections of routing policies that together are not safe *in the sense that they can cause BGP to diverge. That is, an unsafe collection of routing policies can result in some autonomous systems exchanging BGP routing messages indefinitely, without ever converging to a set of stable routes. In this paper we present sufficient conditions on routing policies that guarantee BGP safety. We use a new formalism, called the* Simple Path Vector Protocol *(SPVP), that is designed to capture the underlying semantics of any path vector protocol such as BGP. We identify a certain circular set of relationships between routing policies at various autonomous systems that we call a* dispute cycle. *We show that systems with no dispute cycles are guaranteed to be safe. While these include systems whose policies are consistent with shortest paths under some link metric, the class of systems with no dispute cycles is strictly larger.*

1 Introduction

BGP [9, 7, 10] allows each autonomous system to independently formulate its routing policies, and it allows these policies to override distance metrics in favor of policy concerns. In contrast to pure distance-vector protocols such as RIP [8], Varadhan *et al.* [11] have shown that there are routing policies that are *unsafe* in the sense that they can cause BGP to diverge. Although it seems that BGP divergence has not yet occurred in practice, it could potentially introduce a great deal of instability into the global routing system.

The goal of this paper is to clarify the nature of BGP policy inconsistencies that can give rise to protocol divergence. Our main contributions can be summarized as follows. We introduce a general formalism, called the *Simple Path Vector Protocol* (SPVP), that is designed to capture the underlying semantics of any path vector protocol such as BGP. We define a *dispute cycle* whose arcs correspond to a certain type of policy dispute. We show that systems with no dispute cycles are guaranteed to be safe. While these include systems whose policies are consistent with shortest paths under some link metric, the class of systems with no dispute cycles is strictly larger.

The paper is organized as follows. In Section 2 we define the *Simplified Path-Vector Protocol* (SPVP). In an SPVP specification each node (representing an autonomous system) has a list of permitted paths to a destination, together with a ranking of these paths. Solutions to such specifications are defined to be routing trees that satisfy certain stability conditions. We define a model of dynamic evaluation whose stable states correspond to such solutions.

In Section 3 we define two equivalent structures, the *dispute wheel* and the *dispute cycle*, that capture a certain type of circular policy inconsistency. We show that an SPVP specification with no dispute cycle (no dispute wheel) always has a unique solution. In addition, we show that such a specification is safe and so its dynamic evaluation will always arrive at a stable state.

BGP is different from shortest path routing for several reasons. First, the relative ranking of routes in BGP is not, in general, based on path lengths, or any other universally agreed upon cost function. Second, each autonomous system can reject paths arbitrarily (even shortest paths) based on policy considerations. Even so, it seems a natural question to ask which policies are consistent with an edge cost function. We explore this question in Section 4. We introduce the concept of an SPVP specification being consistent with a given edge cost function. Even in this case, one may find routing trees that are not shortest path trees with respect to the cost function. However, we show that any SPVP specification that is consistent with a cost function without non-positive cycles will be safe. An immediate consequence of this is that BGP configurations that are simply based on "hop count" are safe (even with "padding" of AS-paths). On the other hand, we show that BGP-like systems can actually violate "distance metrics" and remain safe. Finally, Section 5 suggests some problems for future work. Full proofs for those omitted here can be found in [5].

1.1 Related Work

Bertsekas *et al.* [1] prove convergence for a distributed version of the Bellman-Ford shortest path algorithm. Because of the differences between BGP and shortest path routing mentioned above, these results do not directly apply to a protocol such as BGP.

In Varadhan *et al.* [11], the convergence properties of an abstraction of BGP is studied. They describe a system similar to our BAD GADGET, for instance, as a simple example of policies which lead to divergence. In their setting, a node must update each time it receives a new route-to-origin "advertisement" from one of its neighbors. This is in contrast to our model where the update sequence determines when nodes process their neighbor's path choices. They also define the notion of an auxiliary graph, called a *return graph*, to study convergence. Return graphs are defined only for systems with a particular topology, namely a ring topology, and a restricted set of allowable paths at each node, namely only counterclockwise paths. A return graph is defined as follows. For a node v and two permitted paths P, Q from v to 0, we define an arc (P, Q) if when storing P at v, and updating the nodes clockwise around the ring, the node v adopts Q when v is once again reached.

Gouda and Schneider [2, 3] have studied metrics which always have a *maximal tree*, that is, a tree in which every node has its 'favorite' path to the origin, contained in the tree. Their notion of a maximal tree is different from the central notion of a *stable* tree introduced in Section 2.2. The latter is based on reaching a *local*, as opposed to a *global*, equilibrium. Roughly speaking, a *metric* in their work corresponds to a method for ranking paths based on edge costs. They characterize metrics which admit a maximal tree for any graph and any possible cost function.

2 A Simple Path-Vector Protocol

This section introduces a framework, called the *Simple Path Vector Protocol* (SPVP), that is designed to capture the underlying semantics of any path vector protocol such as BGP. The intent is to strip away all but the essentials from BGP, leaving only the basic notions of *permitted paths* to a destination and the *ranking* of those paths. We seek to study the safety of routing policies in a manner independent of the details used to implement those policies (for example, BGP attributes and import and export transformations). In modeling BGP we make several simplifying assumptions. First, we ignore all issues relating to internal BGP (iBGP). As a corollary to this, we assume that there is at most one link between any two autonomous systems. Second, we ignore address aggregation. We believe that these simplifications are not of fundamental importance, and we adopt them in order to improve the clarity of the statements and proofs.

2.1 BGP Route Selection

In order to motivate the SPVP formalism, we briefly review the route selection process of BGP [9, 7, 10]. BGP employs a large number of attributes to convey information about each destination. For example, one BGP attribute records the path of all autonomous systems that the route announcement has traversed. For these reasons BGP is often referred to as a *path vector* protocol. The BGP attributes are used by *import policies* and *export policies* at each autonomous system in order to implement its *routing policies*.

In BGP, route announcements are records that include the following attributes.

$\mathbf{nlri}$	:	network layer reachability information (address block for a set of destinations)
$\mathbf{next\_hop}$	:	next hop (address of next hop router)
$\mathbf{as\_path}$	:	ordered list of vertices traversed
$\mathbf{local\_pref}$	:	local preference
$\mathbf{med}$	:	multi-exit discriminator
$\mathbf{c\_set}$	:	set of community tags

The local preference attribute $\mathbf{local\_pref}$ is not passed between autonomous systems, but is used internally within an autonomous system to assign a local degree of preference.

Each record r is associated with a 4-tuple, rank-tuple(r), defined as

$$\langle r.\mathbf{local\_pref},\ \frac{1}{\mid r.\mathbf{as\_path} \mid},\ \frac{1}{r.\mathbf{med}},\ \frac{1}{r.\mathbf{next\_hop}}\rangle.$$

For a given destination d, the records r with $d = r.\mathbf{nlri}$ are ranked using lexical ordering on rank-tuple(r). The best route selection procedure for BGP [9] picks routes with the highest rank. In other words, if two route records share the same $\mathbf{nlri}$ value, then the record with the highest local preference is most preferred. If local preference values are equal, then the record with the shortest $\mathbf{as\_path}$ is preferred. If these paths have the same length, then the record with the lowest $\mathbf{med}$ value is preferred. Finally, ties are broken with preference given to the record with the lowest IP address for its $\mathbf{next\_hop}$ value. Note that this ordering is "strict" in the sense that if two records r_1, r_2 are ranked equally, then $r_1.\mathbf{next\_hop} = r_2.\mathbf{next\_hop}$. Route selection based on highest rank is always deterministic since at any time there is at most one route record with a given $\mathbf{nlri}$ and a given $\mathbf{next\_hop}$.

A *route transformation* $\mathcal{T}$ is a function on route records, $\mathcal{T}(r) = r'$, that operates by deleting, inserting, or modifying the attribute values of r. If $\mathcal{T}(r) = \langle\rangle$ (the empty record), then we say that r has been *filtered out* by $\mathcal{T}$.

Suppose u and w are autonomous systems with a BGP peering relationship. As a record r moves from w to u it undergoes three transformations. First, $r_1 = \text{export}(u \leftarrow w, r)$ represents the application of *export policies* (defined by w) to r. Second, $r_2 = \text{PVT}(u \leftarrow w, r_1)$ is the BGP-specific *path vector* transformation that adds w to the $\mathbf{as\_path}$ of r_1 and filters out the record if its $\mathbf{as\_path}$ contains u. Finally, $r_3 = \text{import}(u \leftarrow w, r_2)$ represents the application of import policies (defined at u) to r_2. In particular, this is the function that assigns a $\mathbf{local\_pref}$ value for r_3. We call the composition of these transformations the *peering transformation*, $\text{pt}(u \leftarrow w, r)$, defined as

$$\text{import}(u \leftarrow w, \text{PVT}(u \leftarrow w, \text{export}(u \leftarrow w, r))).$$

Suppose autonomous system u_0 is originating a destination d by sending a route record r_0 with $r_0.\mathbf{nlri} = d$ to (some of) its peers. If u_k is an autonomous system and $P = u_k\ u_{k-1} \cdots u_1\ u_0$ is a simple path where each pair of autonomous systems u_{i+1}, u_i are BGP peers, then we define $r(P)$, *the route record received at u_k from u_0 along path P*, to be

$$\text{pt}(u_k \leftarrow u_{k-1}, \text{pt}(u_{k-1} \leftarrow u_{k-2}, \cdots \text{pt}(u_1 \leftarrow u_0, r_o))).$$

We say that P is *permitted* at u_k when $r(P) \neq \langle\rangle$. We can then define a *ranking function*, $\lambda^{u_k}(P)$, on AS-paths permitted at u_k as $\lambda^{u_k}(P) = \text{lex-rank}(\text{rank-tuple}(r(P)))$.

The SPVP formalism defined below is based on the notion of permitted paths and ranking functions on these paths. In terms of BGP, we can think of SPVP as capturing the semantics that translate the *apparent* routing policies at autonomous system u_k (the import and export policies defined at u_k) into the *actual* routing policies at u_k. Note that the actual routing policies at u_k are the result of the interaction between routing policies of many, possibly distant, autonomous systems.

2.2 SPVP Specifications

A simple, undirected, connected graph $G = (V, E)$ represents a network of nodes $V = \{0, 1, 2, \ldots, n\}$ connected by edges E. For any node u, $\text{peers}(u) = \{w \mid \{u, w\} \in E\}$ is the set of *peers* for u. We assume that node 0, called the *origin*, is special in that it is the destination to which all other nodes will attempt to establish a path.

A *path* in G is a sequence, $v_k,\ v_{k-1},\ \ldots\ v_1,\ v_0$, of nodes such that for each $i > 0$, $\{v_i, v_{i-1}\}$ is an edge in E. The empty path is written ϵ. We assume that all non-empty paths $P = v_k, v_{k-1}, \ldots, v_1, v_0$ implicitly have a direction from the *first node* v_k to the *last node* v_0. Suppose $e = \{u, v\}$ is an edge in E. If node v is the first node of path P, then $(u, v)P$ denotes the path that starts at node u, traverses edge e, and then follows path P.

If P and Q are non-empty paths such that the first node in Q is the same as the last node in P, then PQ denotes the path formed by the *concatenation* of these paths. We extend this with the convention that $\epsilon P = P\epsilon = P$, for any path P. For a simple path $P = v_k, v_{k-1}, \ldots, v_1, v_0$ and for any i, j with $k \geq i > j \geq 0$ we denote by $P[v_i, v_j]$ the subpath $v_i, v_{i-1}, \ldots, v_j$.

For each $v \in V - \{0\}$, let $\mathcal{P}^v$ be the set of *permitted paths* from v to the origin (node 0). If P is a path from v to the origin and $P \notin \mathcal{P}^v$, then P is said to be *rejected* at node v. If $P = v, v_k, \ldots, v_1, v_0 = 0$ is in $\mathcal{P}^v$, then the node v_k is called the *next hop* for path P. Let $\mathcal{P} = \bigcup_{v \in V} \mathcal{P}^v$ be the set of all permitted paths to the origin.

For each $v \in V - \{0\}$, there is a non-negative, integer-valued *ranking function* $\lambda^v : \mathcal{P}^v$ representing how node v ranks its permitted paths. If $P_1, P_2 \in \mathcal{P}^v$ and $\lambda^v(P_1) < \lambda^v(P_2)$, then P_2 is said to be *preferred over* P_1. Let $\Lambda = \{\lambda^v \mid v \in V - \{0\}\}$. An *SPVP specification*, $S = (G, \mathcal{P}, \Lambda)$, is a graph together with the permitted paths at each non-zero node and the ranking functions for each non-zero node.

We impose the following restrictions on Λ and $\mathcal{P}$.

(empty path is permitted) for each $v \in V$, $\epsilon \in \mathcal{P}^v$,

(empty path is lowest ranked) for each $v \in V$, $\lambda^v(\epsilon) = 0$,

(strictness) if $\lambda^v(P_1) = \lambda^v(P_2)$, then $P_1 = P_2$ or there is a u such that $P_1 = (v, u)P_1'$ and $P_2 = (v, u)P_2'$ (paths P_1 and P_2 have the same next-hop).

(simplicity) if path $P \in \mathcal{P}^v$, then P is a simple path (no repeated nodes),

(consistency of permitted paths) If $P \in \mathcal{P}^v$ and $w \neq 0$ is in P, then $P[w, 0] \in \mathcal{P}^w$.

Paths correspond to BGP's $\mathbf{as\_path}$ attribute. Unlike BGP however, in SPVP the number u is prepended to all paths at node u. This merely simplifies the exposition. The "consistency of permitted paths" is also not an essential condition, since any specification that does not satisfy it can easily be transformed into one that does.

Let $S = (G, \mathcal{P}, \Lambda)$ be an SPVP specification. A *routing tree* $T = (P_1, P_2, \cdots, P_n)$ is a vector of paths with $P_i \in \mathcal{P}^i$, such that the union of these paths is a tree. Note that some of the P_i may be the empty path. Node i is *stable* with respect to this tree if $\lambda^i((i, j)P_j) \leq \lambda^i(P_i)$ whenever $(i, j)P_j \in \mathcal{P}^i$. A tree $T = (P_1, P_2, \cdots, P_n)$ is *stable* if every node is stable.

An SPVP specification $S = (G, \mathcal{P}, \Lambda)$ is called *solvable* if there exists a stable routing tree for S. Otherwise, S is called *unsolvable*. A stable routing tree T is called a *solution* for the specification S.

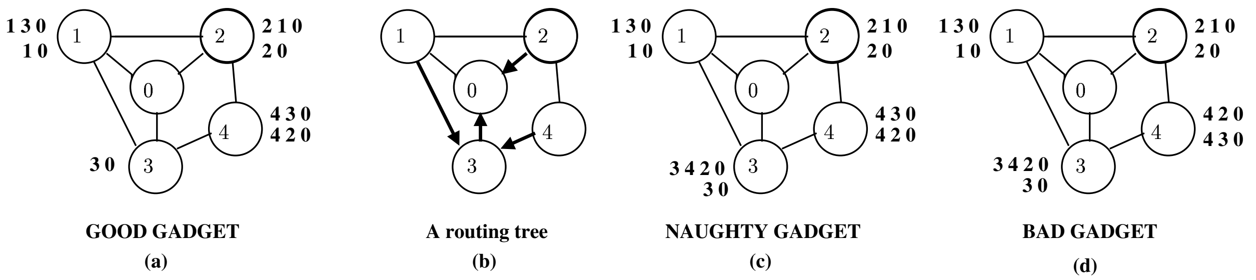

Figure 1. Examples of SPVP specifications.

Griffin and Wilfong [6] have shown that statically detecting solvability for real-world BGP is NP-hard. Similarly, [5] shows that the basic question of solvability is still NP-complete for the more abstract model of SPVP.

Figure 1 (a) presents an SPVP specification called GOOD GADGET. The ranking function for each non-zero node is depicted as a vertical list next to the node, with the highest ranked path at the top going down to the lowest ranked non-empty path at the bottom. The routing tree

$$(\ (1\ 3\ 0),\ (2\ 0),\ (3\ 0),\ (4\ 3\ 0)\ )$$

is a solution for this specification, and it is illustrated in Figure 1 (b). The dynamic model of SPVP, defined below, will always converge to this solution. Also, this is the only solution for GOOD GADGET since any other routing tree is not stable. For example, in the tree

$$(\ (1\ 0),\ (2\ 0),\ (3\ 0),\ (4\ 3\ 0)\ ),$$

nodes 1 and 2 would both prefer to change their paths to ones of higher rank. A modification of GOOD GADGET, called NAUGHTY GADGET, is shown in Figure 1 (c). NAUGHTY GADGET adds one permitted path $(3\ 4\ 2\ 0)$ for node 3, yet it has the same unique solution as GOOD GADGET. However, as is explained below, the dynamic evaluation of this specification can diverge. Finally, by reordering the ranking of paths at node 4, we produce a specification called BAD GADGET, presented in Figure 1 (d). This specification, which is similar to examples of [11], has no solution and its dynamic evaluation will always diverge.

2.3 A Simple Dynamic Model

We now present a simplified model of distributed evaluation that ignores the implementation details related to message queues and message passing. This model was used in [6], but with a different specification language. This model is equivalent to a special case of a more general message passing model, where it is assumed that each node performs an atomic step that processes all of its message queues, computes any changes to best routes, and sends update messages to all of its peers.

A *state* for a specification $S = (G, \mathcal{P}, \Lambda)$ is a vector $s = (P_1, \ldots, P_{n-1})$ of paths where $P_i \in \mathcal{P}^i$. Note that states do not always represent trees. We say that *path P_i is stored at node i* in this state, and we also say that the trivial path 0 is always stored at node 0. The system moves from state to state as nodes *update*. When a node updates it can replace its current path with a path of higher rank, if such a path is available. It can also lose its path, if it is no longer available from its next hop, and be forced to accept a path of lower rank.

To formalize this, we define the set of choices that a node has when it updates, how it chooses a best path, and how it updates. The choice of paths for node u in state $s = (P_1, \ldots, P_{n-1})$ is the set $\mathrm{Choices}(u, s)$, which is defined to be all $P \in \mathcal{P}^u$ such that either $P = (u, 0)$ and $\{u, 0\} \in E$ or $P = (u, v)P_v$ for some $\{u, v\} \in E$. There is one *best* choice in any state s, $\mathrm{Best}(u, s)$, defined to be the unique $P \in \mathrm{Choices}(u, s)$ such that $\lambda^u(P)$ is maximal, if $\mathrm{Choices}(u, s)$ is not empty. Otherwise, $\mathrm{Best}(u, s)$ is the empty path.

In order to model asynchronous routing processes we allow multiple nodes to update simultaneously. Let $A \subseteq V$ be non-empty, and $s = (P_1, \ldots, P_{n-1})$ be a state. Then state $s' = (P'_1, \ldots, P'_{n-1})$ is *reached from s by updating the nodes of A* if

$$P'_i = \begin{cases} P_i & \text{if } i \notin A, \text{ (i does not update)} \\ \mathrm{Best}(i, s) & \text{otherwise.} \end{cases}$$

We use the notation $s \xrightarrow{A} s'$ to denote this state transition. For example,

$$((1\ 0),\ (2\ 0),\ (3\ 0),\ \epsilon) \xrightarrow{\{1,2\}} ((1\ 3\ 0),\ (2\ 1\ 0),\ (3\ 0),\ \epsilon)$$

is a state transition for BAD GADGET.

A state $s = (P_1, \ldots, P_{n-1})$ is *stable* if for each $i \in V - \{0\}$ we have $\mathrm{Best}(i, s) = P_i$, or equivalently, if $s \xrightarrow{A} s$ for every set A. Informally, in a stable state there is no node that could pick a path better than its current path. It is easy to show that any stable state must contain a solution (a stable routing tree).

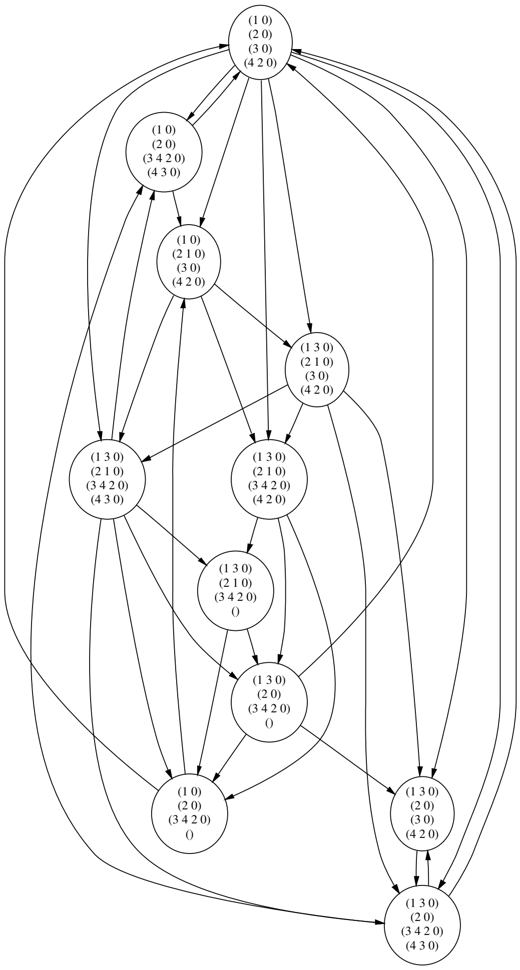

Figure 2. A strongly connected component from the evaluation digraph of NAUGHTY GADGET**.**

The *evaluation digraph* of a specification $S = (G, \mathcal{P}, \Lambda)$, denoted $\mathrm{Eval}(S)$, is a labeled directed graph having one node for each possible state. If $s \xrightarrow{A} s'$, then there is an arc from the node representing s to the node representing s' labeled A. A *cycle* C in $\mathrm{Eval}(S)$ is a sequence of states

$$s_1 \xrightarrow{A_1} s_2 \xrightarrow{A_2} s_3 \cdots \xrightarrow{A_m} s_{m+1}$$

where $s_1 = s_{m+1}$. This cycle is *non-trivial* if it contains no self loops. It is clear that if $\mathrm{Eval}(S)$ contains a non-trivial cycle, then the specification S can diverge.

An *update sequence* σ is a function such that $\sigma(t) \subseteq V$, for each $t \geq 0$. From any starting state s_0 the function σ defines an infinite path in $\mathrm{Eval}(S)$ composed of arcs $s_t \xrightarrow{\sigma(t)} s_{t+1}$. We write $\sigma(s_0, t)$ to denote s_t.

A specification S is said to *converge* with update sequence σ and initial state s_0, written $S(s_0, \sigma) \downarrow$, if there is some time t such that $\sigma(s_0, t)$ is a stable state. Otherwise, S is said to *diverge* with σ and initial state s_0, written $S(s_0, \sigma) \uparrow$. An update sequence σ is *fair* if each node u, $u \in \sigma(t)$ for infinitely many t's. A system S is said to be *safe* if $S(s_0, \sigma) \downarrow$ for every fair σ and every initial state s_0.

The specification GOOD GADGET is safe. On the other hand, NAUGHTY GADGET is solvable, but not safe. The evaluation digraph of NAUGHTY GADGET has 81 states. Figure 2 shows a strongly connected component from this digraph. For readability we have not labeled arcs, and we do not show parallel arcs nor self loops ($s \xrightarrow{A} s$). Any update sequence that remains within this strongly connected component will diverge. Finally, BAD GADGET will diverge for any update sequence.

3 Sufficient Conditions

In this section we develop a sufficient condition that will guarantee that an SPVP specification has a unique solution and is safe. The sufficient condition concerns the absence of a circular set of relationships between ranking functions that we call a *dispute cycle*. The structure of dispute cycles is further elucidated with the definition of an equivalent structure called a *dispute wheel*.

3.1 The Dispute Digraph

For any specification $S = (G, \mathcal{P}, \Lambda)$ we construct a directed graph, $\mathcal{DD}(S)$, called the *dispute digraph*. For each permitted path P of S there is a node in $\mathcal{DD}(S)$ labeled P. There are two kinds of arcs in the dispute digraph, *transmission arcs* and *dispute arcs*.

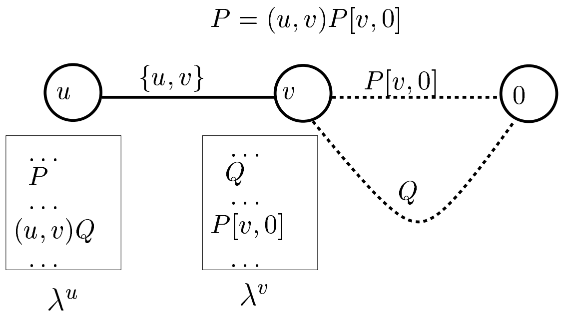

Figure 3. Conditions for dispute arc $Q \longrightarrow P$.

Suppose that nodes u and v are peers. If Q is a permitted path at v and P is a permitted path at u, then a *dispute arc* from path Q to path P, denoted $Q \longrightarrow P$, represents a local policy dispute between peers u and v concerning the relative ranking of paths P and Q. Informally, it represents the fact that node v could increase the rank of its best path by abandoning $P[v, 0]$ and adopting Q, while this action would force u to abandon path P and select as its best path one that could potentially have lower rank than P. More formally, $Q \longrightarrow P$ is a dispute arc if and only if the following conditions hold :

1. P is a permitted path from u to 0 with next-hop v,
2. Q is a path from v to 0, permitted at v,
3. path $(u, v)Q$ is rejected at u, or $\lambda^u((u, v)Q) < \lambda^u(P)$,
4. $\lambda^v(P[v, 0]) \leq \lambda^v(Q)$.

Figure 3 illustrates these conditions.

There is a *transmission arc* from vP to $(u, v)P$, denoted $vP \cdots> (u, v)P$, when nodes u and v are peers, vP is permitted at v, and $(u, v)P$ is permitted at u. Informally, $vP \cdots> (u, v)P$ might be read as "node v permits path vP, which allows u to permit path $(u, v)P$". Figure 4 shows the dispute digraphs for the specifications of Figure 1. Again, the dotted arcs are transmission arcs, while the solid arcs are dispute arcs.

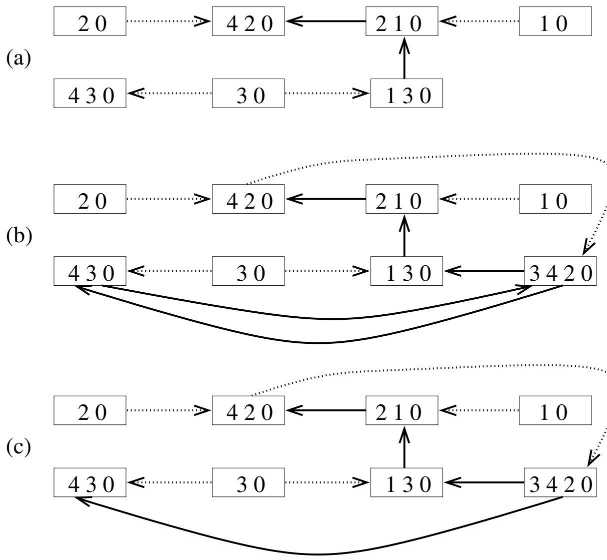

Figure 4. Dispute digraphs for (a) GOOD **, (b)** NAUGHTY**, and (c)** BAD GADGET**.**

A *directed path* in $\mathcal{DD}(S)$ is of the form

$$T = P_1\ a_1\ P_2\ a_2\ \cdots P_{n-1}\ a_{n-1}\ P_n,$$

where $P_i \in \mathcal{P}$, and each $P_i\ a_i\ P_{i+1}$ represents a dispute arc or a transmission arc. A directed path contains a cycle if $P_i = P_j$ for some $i \neq j$. We usually refer to cycles in the dispute digraph as *dispute cycles*.

Lemma 3.1 *Any dispute cycle must contain at least two dispute arcs.*

From Figure 4 we see that the dispute digraph of GOOD GADGET has no cycles, while the dispute digraph of NAUGHTY GADGET contains the simple cycles

$$(3\ 4\ 2\ 0) \longrightarrow (4\ 3\ 0) \longrightarrow (3\ 4\ 2\ 0)$$

and

$$(1\ 3\ 0) \longrightarrow (2\ 1\ 0) \longrightarrow (4\ 2\ 0) \cdots> (3\ 4\ 2\ 0) \longrightarrow (1\ 3\ 0).$$

The second cycle is also contained in the dispute digraph of BAD GADGET.

3.2 Dispute Wheels

We now give an alternate representation of dispute cycles in terms of structures called *dispute wheels*. While dispute cycles are built from local relationships between the ranking functions of peers, dipute wheels are based on "long distance" relationships.

A *dispute wheel,* $\Pi = (U, \mathcal{Q}, \mathcal{R})$*, of size* k, is a set of nodes $U = \{u_0, u_1, \cdots u_{k-1}\}$, and sets of paths $\mathcal{Q} = \{Q_0, Q_1, \cdots Q_{k-1}\}$ and $\mathcal{R} = \{R_0, R_1, \cdots R_{k-1}\}$, such that for each $0 \leq i \leq k-1$ we have (1) R_i is a path from u_i to u_{i+1}, (2) $Q_i \in \mathcal{P}^{u_i}$, (3) $R_i Q_{i+1} \in \mathcal{P}^{u_i}$, and (4) $\lambda^{u_i}(Q_i) \leq \lambda^{u_i}(R_i Q_{i+1})$. When discussing dispute wheels, all subscripts are to be interpreted modulo k. See Figure 5 for an illustration of a dispute wheel. Since permitted paths are simple, it follows that the size of any dispute wheel is at least 2.

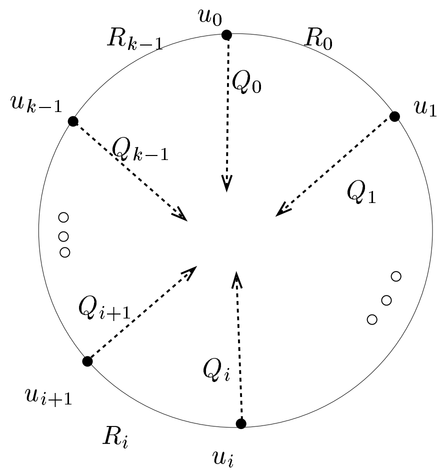

Figure 5. A dispute wheel of size k**.**

The *rim* of a dispute wheel Π is the (possibly non-simple) path $R_0 R_1 \cdots R_{k-1}$, which is a (possibly non-simple) cycle in the graph G. A *rim fragment* is any path of the form $R_i R_{i+1} \cdots R_{i+m}$, where $0 \leq i \leq k-1$ and $0 \leq m \leq k-2$.

A dispute wheel $\Pi' = (U', \mathcal{Q}', \mathcal{R}')$, is a *sub-wheel* of a dispute wheel Π if $U' \subseteq U$, $\mathcal{Q}' \subseteq \mathcal{Q}$, and each $R \in \mathcal{R}'$ is a rim fragment of Π.

A *minimal dispute wheel* is one in which for each $0 \leq i \leq k-1$, either $R_i R_{i+1} Q_{i+2}$ is not permitted at u_i, or $\lambda^{u_i}(R_i R_{i+1} Q_{i+2}) \leq \lambda^{u_i}(R_i Q_{i+1})$. Note that any dispute wheel of size 2 is minimal.

Lemma 3.2 *Every dispute wheel contains a minimal sub-wheel.*

Proof: Suppose that Π is a dispute wheel that is not minimal. Then for some u_i in Π we have $\lambda^{u_i}(R_i Q_{i+1}) < \lambda^{u_i}(R_i R_{i+1} Q_{i+2})$. Create a sub-wheel of size $k-1$ by deleting u_{i+1} and Q_{i+1}, and replacing path R_i with rim fragment $R_i R_{i+1}$. Repeating this process must eventually arrive at a minimal sub-wheel. ■

We now show that dispute wheels are equivalent to dispute cycles. We can extend the notion of a dispute arc to "distant disputes" in the following way. Let $P = P_1 P_2$ be a path permitted at u, where P_2 is a path permitted at some node v. Suppose that Q is also permitted at v, and we have

1. path $P_1 Q$ is rejected at u, or $\lambda^u(P_1 Q) < \lambda^u(P_1 P_2)$,
2. $\lambda^v(P_2) \leq \lambda^v(Q)$.

We write $Q \rightsquigarrow P$ when these conditions hold.

Lemma 3.3 *If $Q \rightsquigarrow P$, then there is a path in the dispute digraph from Q to P of length $\mid P_1 \mid$.*

Corollary 3.4 *Suppose that $\Pi = (U, \mathcal{Q}, \mathcal{R})$ is a minimal dispute wheel of size k. Then*

$$R_0 Q_1 \rightsquigarrow R_{k-1} Q_0 \cdots R_i Q_{i+1} \rightsquigarrow R_{i-1} Q_i \cdots \rightsquigarrow R_0 Q_1$$

and so there is a directed cycle in the dispute digraph.

Lemma 3.5 *If the dispute digraph $\mathcal{DD}(S)$ contains a cycle, then S has a dispute wheel.*

Corollary 3.6 *A specification S has a dispute wheel if and only if the dispute digraph $\mathcal{DD}(S)$ contains a cycle.*

3.3 Two Trees Imply a Dispute Wheel

In general, an SPVP specification may have more than one solution. We show that in this case the dispute digraph has a cycle.

Although a solution is not always a spanning tree, it often simplifies proofs to assume that all solutions of S are spanning trees. For any specification S we can construct an essentially equivalent specification $\hat{S}$ all of whose solutions are spanning trees. Add a new node v^* adjacent to 0 for which v^*0 is its only permitted path. Also add the edge $\{u, v^*\}$ for each $u \in V$ and make $(u, v^*, 0)$ the unique path whose u-ranking is greater than $\lambda^u(\epsilon)$, and modify λ^u so that this is less than all other paths in $\mathcal{P}^u$. This allows us to use the following fact.

Fact 3.7 *For any specification S there is a 1-1 mapping between its solutions and those of $\hat{S}$. Namely, for for each solution T for S there is a unique solution $\hat{T}$ for $\hat{S}$ such that T is a subtree of $\hat{T}$.*

Theorem 3.8 *If a specification S has more than one solution, then it has a dispute wheel.*

Proof: Suppose S has two distinct solutions $\mathbf{P} = (P_1, \ldots, P_{n-1})$ and $\mathbf{Q} = (Q_1, \ldots, Q_{n-1})$. We represent the two stable configurations $\mathbf{P}, \mathbf{Q}$ as two trees T_1, T_2 rooted at the node 0. Using Fact 3.7, we assume that T_1, T_2 are spanning. Let H be the graph $(V, E(T_1) \cap E(T_1))$ which is induced by the intersection of these two trees. Now let T be the component of H containing the origin. Thus every edge of H entering $V(T)$ is either in $E(T_1) - E(T_2)$ or $E(T_2) - E(T_1)$.

We now construct a dispute wheel. Since $V - V(T)$ is nonempty (otherwise $T_1 = T_2$) we may choose an edge $\{u_0, v_0\} \in T_1$, where $u_0 \notin V(T)$ and $v_0 \in V(T)$. On the other hand, u_0 has a path to the origin in T_2. This path must be of the form $R_0(u_1, v_1)Q_1$ where (i) $u_1 \notin V(T), v_1 \in V(T)$ and Q_1 is the unique path in T from v_1 to the origin, (ii) R_0 is a path from u_0 to u_1 in T_2 but entirely contained in the node set $V - V(T)$ and (iii) R_0 has at least one edge (for otherwise one of T_1, T_2 would not be stable). We repeat this process at u_1, except we now examine a path from u_1 to the origin in the tree T_1. Continuing to alternate in this fashion we must eventually repeat some node, which without loss of generality is u_0.

To see that this is a dispute wheel, we need only show that for each i,

$$\lambda^{u_i}((u_i, v_i)Q_i) \leq \lambda^{u_i}(R_i(u_{i+1}, v_{i+1})Q_{i+1}).$$

Without loss of generality, assume that $(u_i, v_i)Q_i$ is in T_1. If the inequality did not hold, then we would have

$$\lambda^{u_i}(R_i(u_{i+1}, v_{i+1})Q_{i+1}) < \lambda^{u_i}((u_i, v_i)Q_i),$$

which would mean that T_2 is not stable. ■

Note that NAUGHTY GADGET illustrates the fact that the converse of this result does not hold. NAUGHTY GADGET has a unique solution but is not safe, and so has a cycle in its dispute digraph.

3.4 No Dispute Wheel Implies a Solution

Theorem 3.9 *Let S be an SPVP specification. If S has no dispute wheel, then S is solvable.*

Proof: Using Fact 3.7, we can assume that any solution for S will be a spanning tree. Suppose that T is a tree (not necessarily spanning) in G, rooted at 0, such that each node of T is stable with respect to T. If $u \in V - V(T)$ and $P \in \mathcal{P}^u$, then P is said to be *consistent with* T if it can be written as $P = P_1(u_1, v_1)Q_1$, where P_1 is a path in $V - V(T)$, $v_1 \in V(T)$, Q is the unique path from v_1 to the origin in T, and $\{u_1, v_1\} \in E$. Such a P is called a *direct path to* T if P_1 is empty and $u = u_1$. Let $D(T)$ be the set of nodes $u \in V - V(T)$ that have a direct path to T. Note that if $V - V(T)$ is not empty, then $D(T)$ is not empty (G is connected). Let $H(T)$ be the set of nodes $u \in D(T)$ whose highest ranked path consistent with T is a direct path. If $V - V(T)$ is not empty, let $F(T)$ be the tree formed by adding the nodes of $H(T)$ to T together with their highest ranked direct paths.

Let $T_0 = \{0\}$ be the trivial tree rooted at the origin. If T_i is such that $V - V(T_i)$ is not empty, then define

$$T_{i+1} = \begin{cases} F(T_i) & \text{if } H(T_i) \text{ is not empty} \\ \textbf{error} & \text{otherwise} \end{cases}$$

Note that if all nodes of T_i are stable with respect to T_i, then all nodes of T_{i+1} are stable with respect to T_{i+1}. Thus, if there exists an i with $V(T_i) = V$, then T_i is a solution for S, since it is stable.

On the other hand, suppose there exists an i such that $T_{i+1} =$ **error**. Let u_0 be any node in $D(T_i)$ and let $Q_0 \in \mathcal{P}^{u_0}$ be a direct path. Note that there must be a path P_0, permitted at u_0 and consistent with T_i, which has higher rank than Q_0. Since P_0 is consistent with T_i it has the form $P_0 = R_0(u_1, v_1)Q_1$ where R_0 is a path from u_0 to u_1 in $V - V(T_i)$, $v_1 \in V(T_i)$, Q_1 is the unique path from v_1 to 0 in T_i, and $\{u_1, v_1\} \in E$. Note that $v_1 \in D(T_i)$, and since $H(T_i)$ is empty we can repeat this process with u_1. If we continue in this manner it is clear that we will eventually form a dispute wheel. ■

3.5 Divergence Implies a Dispute Wheel

Suppose that C is a non-trivial cycle in the evaluation digraph. A node u *is changing in a cycle* C if there are at least two distinct states of C in which u has different paths. Since C is non-trivial, there is at least one node changing in C. Let $\mathrm{values}(C, u)$ be the set of paths that u adopts in C. Let $F(C)$ be the set of nodes that store a fixed path throughout C. Note that $0 \in F(C)$, so this set is never empty.

Lemma 3.10 *Suppose* $P \in P^u$ *is adopted by* u *in* C, *and let* v *be the first fixed node of* P. *Then each node* $w \in P[u, v]$, *stores the path* $P[w, 0]$ *in some state of* C. *In particular,* v *stores* $P[v, 0]$ *throughout* C.

Proof: Let $P[u, v] = (u = x_0, x_1, \ldots x_{t-1}, x_t = v)$. The result holds by assumption for x_0, so suppose that for some $i \geq 0$, and for each $j \leq i$, x_j adopts the path $P[x_j, 0]$ in some state of C. If $i = t$, the result is proved. Otherwise x_i is changing in C and adopts the path $P[x_i, 0]$; thus at this point in the cycle, x_{i+1} must have $P[x_{i+1}, 0]$ stored. The result follows by induction. ■

Theorem 3.11 *If there is a non-trivial cycle in the evaluation digraph, then* S *contains a dispute wheel.*

Proof: Let C be a non-trivial cycle in $\mathrm{Eval}(S)$. Let U be the subset of nodes u changing in C such that there is a path $(u, w)Q \in \mathrm{values}(C, u)$ where $w \in F(C)$. That is, u adopts a path in C that leads directly to a fixed node. By Lemma 3.10, U cannot be empty.

We now construct a dispute wheel. Let u_0 be a node in U. Let Q_0 be u_0's direct path to $F(C)$, $(u_0, w_0)Q_0'$. It is easy to check that Q_0 is unique, and that of all paths in $\mathrm{values}(C, u_0)$ the path Q_0 is of lowest rank. Let $H_0 \in \mathrm{values}(C, u_0)$ be the adopted path of highest rank at u_0. Lemma 3.10 tells us that we can write this path as $H_0 = R_0Q_1$, where R_0 is a path from u_0 to u_1 of changing nodes, $u_1 \in U$, and $Q_1 = (u_1, w_1)Q_1'$ for some $w_1 \in F(C)$. We can now perform the same construction for u_1. Repeating this process in the obvious way results in a dispute wheel. ■

Corollary 3.12 *If* S *has no dispute wheel, then the evaluation graph* $\mathrm{Eval}(S)$ *has no non-trivial cycles, and so* S *is safe.*

The converse of this result does not hold. For example, BAD BACKUP presented in Figure 6 is the result of a slight modification to BAD GADGET (the path (40) is added and made the highest ranked path at node 4). This specification has a dispute wheel but the evaluation graph has no cycles. In other words, the dispute wheel of BAD BACKUP is not dynamically realizable in our simple model of evaluation. Note that if the edge $\{0, 4\}$ is deleted (modeling link failure), then this system becomes BAD GADGET.

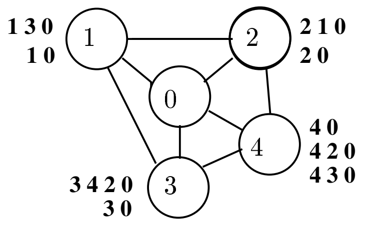

Figure 6. The specification BAD BACKUP**.**

4 SPVP and Shortest Paths

Varadhan *et al.* [11] first observed that BGP policies could interact in a way that results in protocol divergence. Their examples always include autonomous systems that choose longer paths (in terms of "hop count") over shorter ones. They stated "*We believe that only shortest path route selection is provably safe*." The results of the previous sections will be used to explore this statement. We interpret it to mean that any class of policies not based on shortest path route selection will not be provably safe. Notice that implicitly, the conjecture is suggesting that systems whose policies are based on shortest path route selection will, in fact, be safe.

We begin by formalizing a fairly liberal notion of "shortest path route selection" that seems appropriate for protocols such as BGP. We then show that any SPVP specification that is consistent with shortest path route selection will indeed be safe. However, we show that the converse is not true. Hence, BGP-like systems can actually violate "distance metrics" and remain safe.

As is standard for undirected graphs, we work with an *associated orientation* of it; we think of an undirected edge $e = \{a, b\}$ as being replaced by two arcs, $e^- = (a, b)$ and $e^+ = (b, a)$. We are also given costs $c(e^+)$ and $c(e^-)$ associated with traversing the edge e in the two directions. Thus c induces a cost function on any directed path P in the resulting oriented graph: $c(P) = \sum_{a \in A(P)} c(a)$. The cost function c is positive if for each arc a, $c(a) > 0$.

There are several possible ways to formalize the notion of "shortest path route selection" for a cost function c. Since a node u in an SPVP specification is not required to treat all possible paths to the origin as permitted paths, we cannot insist that u take the shortest path. However, it seems reasonable to insist that if u has a choice between two permitted paths and these paths have different costs, then u cannot prefer the higher cost path over the lower cost path. Formally, we say that the $S = (G, \mathcal{P}, \Lambda)$ is *consistent with the cost function* c if for each w and $P_1, P_2 \in \mathcal{P}^w$, (1) if $\lambda^w(P_1) < \lambda^w(P_2)$, then $c(P_2) \leq c(P_1)$, and (2) if $\lambda^w(P_1) = \lambda^w(P_2)$, then $c(P_2) = c(P_1)$.

If S is consistent with a cost function c, then there are only two sources for policy disputes. First, not all paths have to be permitted at any given node. Second, ranking functions force ties to be broken, and this may be done differently at different nodes. Both reasons are captured in the following lemma.

Lemma 4.1 *If specification S is consistent with cost function c, and $Q \longrightarrow P$ is a dispute arc, where $P \in \mathcal{P}^u$ and $Q \in \mathcal{P}^w$. Then either (a) $(u, w)Q \notin \mathcal{P}^u$ or (b) $c(Q) = c(P[w, 0])$.*

If a cost function c has negative directed cycles, then S can be consistent with c and yet not safe. For example, consider the costs attached to the edges of NAUGHTY GADGET in Figure 7, where the cost of traversing an edge is the same in each direction. NAUGHTY GADGET is consistent with this cost function, but we know from Section 2 that this system is not safe. Note that this graph contains a cycle of cost -16. Also, notice that any S will be consistent with the cost function c that has cost 0 for every arc and so, in particular, NAUGHTY GADGET will be consistent with such a cost function. Thus we restrict ourselves to SPVP specifications consistent with cost functions that do not realize any directed cycles of cost at most 0.

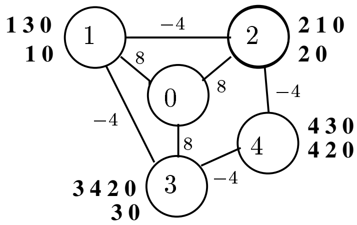

Figure 7. NAUGHTY GADGET **with negative link costs**

Define a cost function c to be *coherent* if it does not result in any non-positive directed cycles. Note that any positive cost function is coherent.

Theorem 4.2 *If S is consistent with a coherent cost function, then S has no dispute wheel.*

Proof: Suppose that c is a coherent cost function, S is consistent with c, and S contains a dispute wheel of size k. For any $0 \leq i \leq k-1$ we have $\lambda^{u_i}(Q_i) \leq \lambda^{u_i}(R_i Q_{i+1})$, and so $c(R_i Q_{i+1}) = c(R_i) + c(Q_{i+1}) \leq c(Q_i)$. Summing these inequalities we obtain

$$\sum_{i=0}^{k-1} c(R_i) + c(Q_{i+1}) \leq \sum_{i=0}^{k-1} c(Q_i).$$

After cancellation this implies $\sum_{i=0}^{k-1} c(R_i) \leq 0$. Thus the rim of the dispute wheel is a cycle of cost at most zero, which is a contradiction. ■

From Corollary 3.12, we can conclude that any S consistent with a positive cost function is safe. In particular, routing policies based on hop-count (even with AS-padding) are always safe. In addition, it can be shown that if all paths are permitted, then this results in a shortest-path routing tree.

We now show that the converse of Theorem 4.2 is not true. The specification INCOHERENT of Figure 8 has an acyclic dispute digraph and hence is safe. However, it is not consistent with any coherent cost function. To see this, suppose that we are given arc costs $c(1, 2) = A$,

$c(2,3) = B$, $c(3,1) = C$, $c(1,0) = D$, $c(3,0) = E$ and $c(4,3) = F$. The cost for any other arc is arbitrary. Suppose INCOHERENT is consistent with these costs, then the fact that node 1 prefers path $[1\ 2\ 3\ 0]$ over path $[1\ 0]$ means that $A + B + E \leq D$. Also the fact that node 4 prefers path $[4\ 3\ 1\ 0]$ over path $[4\ 3\ 0]$ means that $F + C + D \leq F + E$. Adding these inequalities together we obtain $A + B + C + D + E + F \leq D + E + F$. By cancellation, we arrive at $A + B + C \leq 0$, so there is a nonpositive cycle (123). That is, INCOHERENT is not consistent with any coherent cost function. Notice that the dispute digraph of INCOHERENT, as shown in Figure 8, is acyclic and hence INCOHERENT is safe.

In summary, the class of specifications with acyclic dispute digraphs is provably safe, yet it is strictly larger than those based on shortest paths.

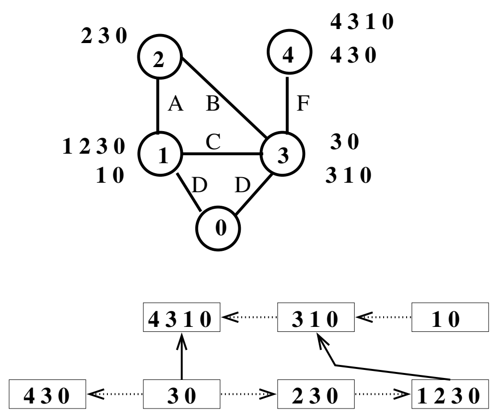

Figure 8. INCOHERENT **and its dispute digraph.**

5 Discussion and Future Work

Is it possible to guarantee that BGP will not diverge? Broadly speaking, this problem can be addressed either *statically* or *dynamically*. A static solution would rely on programs to analyze routing policies to verify that they do not contain policy conflicts that could lead to protocol divergence. This is essentially the approach advocated in Govindan *et al.* [4]. However, there are two *practical* challenges facing this approach. First, autonomous systems currently do not widely share their routing policies, or only publish incomplete specifications. Second, even if there were complete knowledge of routing policies, Griffin and Wilfong [6] have recently shown that checking for various global convergence conditions is either NP-complete or NP-hard. Therefore, a static approach would most likely require the development of new heuristic algorithms for detecting this class of policy conflict.

A *dynamic* solution to the BGP divergence problem would be some mechanism to suppress or completely prevent at "run time" those BGP oscillations that arise from policy conflicts. Using route flap dampening [12] as a dynamic mechanism to address this problem has two distinct drawbacks. First, route flap dampening cannot eliminate BGP protocol oscillations, it will only make these oscillations run in "slow motion". Second, route flap dampening events do not provide network administrators with enough information to identify the source of the route flapping. In other words, route flapping caused by policy conflicts will look the same as route flapping caused by unstable routers or defective network interfaces. So it seems that any dynamic solution would require an *extension* to the BGP protocol to carry additional information that would allow policy disputes to be detected and identified at *run time*.

The proof of Theorem 3.11 contains an algorithm for extracting dispute wheels from dynamic cycles in the evaluation graph. This may provide a key to the design of a dynamic solution. It might be possible to extend the BGP protocol in such a way that this "extraction" can be done in a distributed manner. This could allow for the suppression of routes involved in policy-based oscillations and for the identification of the autonomous systems involved.

References

[1] D. Bertsekas and R. Gallagher. *Data Networks*. Prentice Hall, 1992.

[2] M. Gouda and M. Schneider. Maximizable routing metrics. *Proc. Sixth International Conference on Network Protocols*, pages 71–78, 1998.

[3] M. Gouda and M. Schneider. Stabilization of maximal metric trees. *Workshop on Self-Stabilizing Systems '99*, 1999.

[4] 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.

[5] T. Griffin, F. Shepherd, and G. Wilfong. Policy disputes in path-vector protocols. Bell Labs Technical Memorandum, 1999.

[6] T. Griffin and G. Wilfong. An analysis of BGP convergence properties. In *SIGCOMM'99*, 1999.

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

[8] C. Hendrick. Routing information protocol. RFC 1058, 1988.

[9] Y. Rekhter and T. Li. A border gateway protocol. RFC 1771 (BGP version 4), 1995.

[10] J. W. Stewart. *BGP4, Inter-Domain Routing in The Internet*. Addison-Wesley, 1998.

[11] K. Varadhan, R. Govindan, and D. Estrin. Persistent route oscillations in inter-domain routing. ISI technical report 96-631, USC/Information Sciences Institute, 1996.

[12] C. Villamizar, R. Chandra, and R. Govindan. BGP route flap damping. RFC 2439, 1998.