The Stable Paths Problem and Interdomain Routing

Timothy G. Griffin — AT&T Labs

F. Bruce Shepherd — Bell Labs

Gordon Wilfong — Bell Labs

Abstract—

Dynamic routing protocols such as RIP and OSPF essentially implement distributed algorithms for solving the *Shortest Paths Problem*. The Border Gateway Protocol (BGP) is currently the only interdomain routing protocol deployed in the Internet. BGP is not solving a shortest paths problem since any interdomain protocol is required to allow policy-based metrics to override distance-based metrics and enable autonomous systems to independently define their routing policies with little or no global coordination. It is then natural to ask if BGP can be viewed as a distributed algorithm for solving some fundamental problem. We introduce the *Stable Paths Problem* and show that BGP can be viewed as a distributed algorithm for solving this problem. Unlike a shortest path tree, such a solution does not represent a *global* optimum, but rather an equilibrium point in which each node is assigned its *local* optimum.

We study the Stable Paths Problem using a derived structure called a *dispute wheel*, representing conflicting routing policies at various nodes. We show that if no dispute wheel can be constructed, then there exists a unique solution for the Stable Paths Problem. We define the *Simple Path Vector Protocol* (SPVP), a distributed algorithm for solving the Stable Paths Problem. SPVP is intended to capture the dynamic behavior of BGP at an abstract level. If SPVP converges, then the resulting state corresponds to a stable paths solution. If there is no solution, then SPVP always diverges. In fact, SPVP can even diverge when a solution exists. We show that SPVP will converge to the unique solution of an instance of the Stable Paths Problem if no dispute wheel exists.

I. Introduction

The Border Gateway Protocol, BGP, is currently the only interdomain routing protocol employed on the Internet [13], [18], [19]. BGP 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 [2], [14], BGP is not *safe* in the sense that routing policies can conflict in a manner that causes BGP to diverge, resulting in persistent route oscillations [21]. Moreover, the safety of BGP routing policies may not be *robust* with respect to network failures. Recent studies have highlighted the adverse effects of interdomain routing instability [16], [17]. Although it is not known if any of the observed BGP instability has been caused by policy conflicts, in the worst case such conflicts could introduce extreme oscillations into the global routing system.

The goal of this paper is to clarify the nature of BGP policy inconsistencies that give rise to protocol divergence. Our main contribution is to describe a general condition on routing policies that guarantees safety and robustness.

We introduce the *Stable Paths Problem* (SPP), which captures the underlying semantics of any path vector protocol such as BGP. Just as routing protocols such as RIP and OSPF implement distributed algorithms for solving the Shortest Paths Problem, we claim that BGP can be viewed as a distributed algorithm for solving the Stable Paths Problem. Informally, the Stable Paths Problem consists of an undirected graph with a distinguished node called the *origin*. All other nodes have a set of permitted paths to the origin. Each node also has a ranking function on its permitted paths that indicates an order of preference. A solution to the Stable Paths Problem is an assignment of permitted paths to nodes so that each node's assigned path is its highest ranked path extending any of the assigned paths at its neighbors. Such a solution does not represent a *global* maximum, but rather an equilibrium point in which each node is assigned its *local* maximum. In Section IV we introduce the *Simple Path Vector Protocol* as a distributed means of computing solutions to the Stable Paths Problem.

We then study the Stable Paths Problem using a derived structure called a *dispute wheel*, which represents a circular set of dependencies between routing policies that cannot be simultaneously satisfied. We show that if no dispute wheel can be constructed, then the corresponding Stable Paths Problem has a unique solution. We define the *Simple Path Vector Protocol* (SPVP) as a distributed means of computing solutions to the Stable Paths Problem. We show that if there is no dispute wheel, then SPVP is guaranteed to converge to the unique solution of the corresponding Stable Paths Problem.

The paper is organized as follows. Section II provides a simplified picture of how BGP operates and provides motivation for the definition of the Stable Paths Problem. In

Section III we actually define the Stable Paths Problem (SPP). This formalism provides a simple semantics for routing policies of path vector protocols such as BGP while remaining free of many nonessential details. There is a tradeoff between the complexity of the SPP formalism and the complexity of the translation from a set of BGP routing policies to an instance of SPP. We opted for SPP simplicity, since the theoretical results remain quite challenging even for this model. Hence numerous BGP-specific details, such as internal BGP, confederations, route servers, private AS numbers, and so on, are pushed into the translation.

The protocol SPVP is defined in IV. We analyze the Stable Paths Problem in Section V. First, we explore the computational complexity of the Stable Paths Problem and show that the problem of determining whether an instance of the Stable Paths Problem has a solution is NP-complete. We define the notion of a dispute wheel, and show that an instance of SPP with no dispute wheel always has a unique solution. We also show that the protocol SPVP can only diverge when there is a dispute wheel.

In Section VI we explore the relationship between the Stable Paths and Shortest Paths Problems. SPP is different from shortest paths problem for several reasons. First, the relative ranking of paths in SPP is not, in general, based on path lengths. Second, each node can reject paths arbitrarily, even shortest paths. Even so, it seems a natural question to ask which instances of the Stable Paths Problem are consistent with some edge cost function. Even in this case, one may find routing trees which are not shortest path trees with respect to the cost function. However, we show that any instance of the Stable Paths Problem that is consistent with a cost function without non-positive cycles will be safe. An immediate consequence of this is that if we ignore internal BGP (IBGP), then 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 VII discusses the implication of our results for the Stable Paths Problem for real-world BGP as well as open problems.

A. 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.* [21], the convergence properties of an abstraction of BGP is studied. They describe a system (similar to BAD GADGET of Figure2) as an 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 an arbitrary 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 ring topology, and a restricted set of allowable paths at each node, namely the counter-clockwise paths. A return graph is defined as follows. For a node v and two permitted paths P, Q from v to 0, they 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 considered again. Thus return graphs are defined by the *dynamic* behavior of the system for a particular activation sequence whereas the dispute wheels defined in this paper is based purely on the *static* nature of the local preference functions of the nodes in the system. In addition, we consider a more general evaluation model, more general topologies, and arbitrary ranking of permitted paths.

Gouda and Schneider [7], [8] have studied metrics which always have a *maximal tree*, that is, a tree in which every node has its *most preferred* path to the origin contained in the tree. This notion is different from the central notion of a *stable tree* introduced in Section III. The latter is based on reaching a local optimum as opposed to requiring each node having its globally preferred path. A *metric* in their work corresponds to a method for ranking paths based on a given assignment of values from a prescribed set to the edges of the graph. In particular, this implies a universal ranking of how desirable each path is. They characterize the "maximizable" metrics, i.e., those which admit a maximal tree for any graph and any valid assignment. They show, in particular, that any such metric must be monotonic in the sense that if P is a sub-path of Q, then P cannot be less desirable than Q (for the shortest path metric this means that edges can only be assigned nonnegative costs).

Griffin and Wilfong [11] have shown that statically detecting solvability for real-world BGP is NP-hard. The translation from the "high-level" specification language used in that paper into an instance of the Stable Paths Problem (see Section II) may take exponential time and space (in the number of nodes). Even so, in Section V-A we show that the basic question of solvability is still NP-complete for instances of the Stable Paths Problem.

II. BGP Route Selection

In order to motivate the SPP formalism, we briefly review the route selection process of BGP [13], [18], [19]. 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 router to implement its *routing policies*. 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.

In BGP, route announcements are passed between routers. These 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 autonomous systems 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 [18] 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 deterministic since at any time there is at most one route record learned from $\mathbf{next\_hop}$ with a given $\mathbf{nlri}$.

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, sets $\mathbf{next\_hop}$, 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

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

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 the lexical rank of rank-tuple($r(P)$).

III. The Stable Paths Problem (SPP)

The SPP 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 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. The SPP framework is designed to capture the underlying semantics of any path vector protocol such as BGP. We seek to study the safety of routing policies in a manner independent of the details used to implement those policies.

Let $G = (V, E)$ be a simple, undirected graph where $V = \{0, 1, 2, \ldots, n\}$ is the set of nodes and E is the set of edges. For any node u, 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 attempt to establish a path.

A *path* in G is either the empty path, denoted by ϵ, or a sequence of nodes, $(v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$, $k \geq 0$, such that for each i, $k \geq i > 0$, $\{v_i, v_{i-1}\}$ is in E. Note that if $k = 0$, then (v_0) represents the trivial path consisting of the single node v_0. Each non-empty path $P = (v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$ has a direction from its *first node* v_k to its *last node* v_0. 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 example, $(4\ 3\ 2)\ (2\ 1\ 0)$ represents the path $(4\ 3\ 2\ 1\ 0)$, whereas $\epsilon\ (2\ 1\ 0)$ represents the path $(2\ 1\ 0)$. This notation is most commonly used when P is a path starting with node v and $\{u, v\}$ is an edge in E. In this case $(u\ v)P$ denotes the path that starts at node u, traverses the edge $\{u, v\}$, and then follows path P from node v.

For each $v \in V$, $\mathcal{P}^v$ denotes the set of *permitted paths* from v to the origin (node 0). If $P = (v\ v_k\ \ldots\ v_1\ 0)$ is in $\mathcal{P}^v$, then the node v_k is called the *next hop* of path P. Let $\mathcal{P}$ be the union of all sets $\mathcal{P}^v$.

For each $v \in V$, there is a non-negative, integer-valued *ranking function* λ^v, defined over $\mathcal{P}^v$, which represents 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 instance of the *Stable Paths Problem*, $S = (G,\ \mathcal{P},\ \Lambda)$, is a graph together with the permitted paths at each node and the ranking functions for each node. In addition, we assume that $\mathcal{P}^0 = \{(0)\}$, and for all $v \in V - \{0\}$:
(empty path is permitted) $\epsilon \in \mathcal{P}^v$,
(empty path is lowest ranked) $\lambda^v(\epsilon) = 0$, $\lambda^v(P) > 0$ for $P \neq \epsilon$,
(strictness) If $P_1 \neq P_2$ and $\lambda^v(P_1) = \lambda^v(P_2)$, then 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),

Let $S = (G, \mathcal{P}, \Lambda)$ be an instance of the Stable Paths Problem. A *path assignment* is a function π that maps each node $u \in V$ to a path $\pi(u) \in \mathcal{P}^u$. (Note, this means that $\pi(0) = (0)$.) We interpret $\pi(u) = \epsilon$ to mean that u is not assigned a path to the origin. The set of paths choices(π, u) is defined to be

$$\text{choices}(\pi, u) = \begin{cases} \{(u\ v)\pi(v) \mid \{u,\ v\} \in E\} \cap \mathcal{P}^u & (u \neq 0) \\ \{(0)\} & \text{o.w.} \end{cases}$$

This set represents all possible permitted paths at u that can be formed by extending the paths assigned to the peers of u. Given a node u, suppose that W is a subset of the permitted paths $\mathcal{P}^u$ such that each path in W has a distinct next hop. Then the *best path in* W is defined to be

$$\text{best}(W, u) = \begin{cases} P \in W \text{ with maximal } \lambda^u(P) & (W \neq \emptyset) \\ \epsilon & \text{o.w.} \end{cases}$$

The path assignment π is *stable at node* u if

$$\pi(u) = \text{best}(\text{choices}(\pi, u), u).$$

Note that if π is stable at node u and $\pi(u) = \epsilon$, then the set of choices at u must be empty. The path assignment π is *stable* if it is stable at each node u. We often write a path assignment as a vector, $(P_1,\ P_2,\ \cdots,\ P_n)$, where $\pi(u) = P_u$. (We omit P_0 since it is always (0).) It is easy to check that if π is stable, and $\pi(u) = (u\ w)P$, then $\pi(w) = P$. Therefore, any stable path assignment implicitly defines a tree rooted at the origin. Note, however, that this is not always a spanning tree.

The Stable Paths Problem $S = (G,\ \mathcal{P},\ \Lambda)$ is *solvable* if there is a stable path assignment for S. A stable path assignment is also called a *solution* for S. If no such assignment exists, then S is *unsolvable*.

Figure 1 (a) presents a Stable Paths Problem called SHORTEST 1. 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

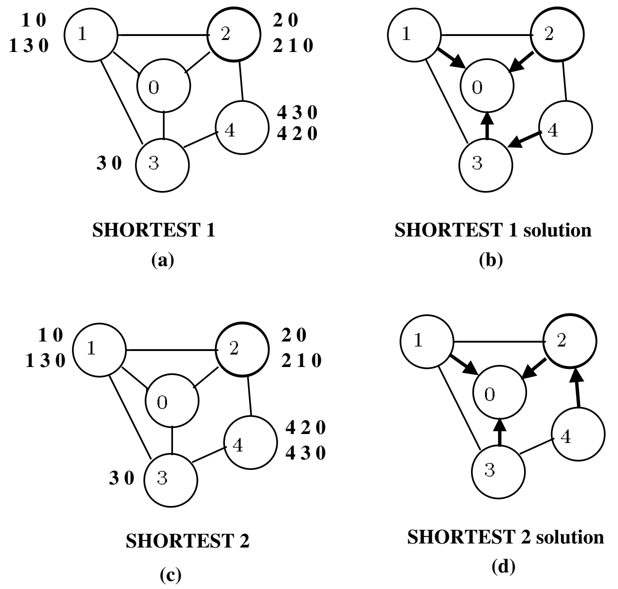


Fig. 1. Stable Paths Problems with shortest path solutions.

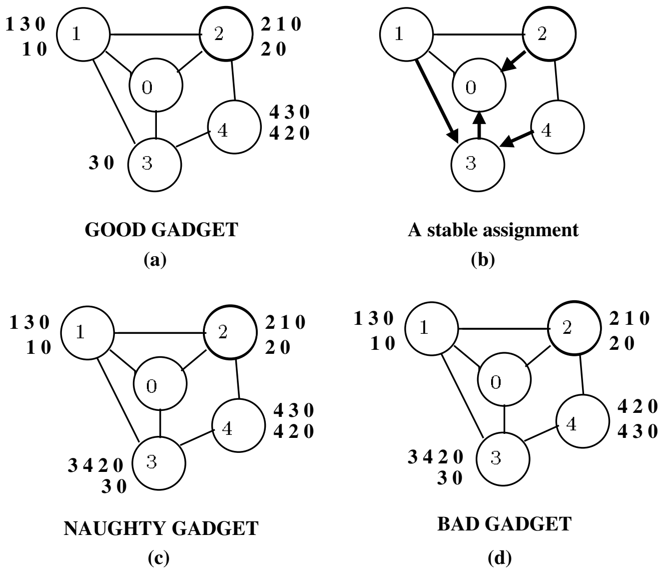


Fig. 2. Stable Paths Problems that are not shortest path problems.

non-empty path at the bottom. The stable path assignment

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

is illustrated in Figure 1 (b). If we reverse the ranking order of paths at node 4 we arrive at SHORTEST 2, depicted in Figure 1 (c). The stable path assignment

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

is illustrated in Figure 1 (d). In both cases, the ranking functions prefer shorter paths to longer paths and the solutions are shortest path trees. Note that the ranking at node 4 breaks ties between paths of equal length. This results in one shortest path tree as the solution for SHORTEST 1, while another shortest path tree as the solution for SHORTEST 2.

The ranking of paths is not required to prefer shorter paths to longer paths. For example, Figure 2 (a) presents a Stable Paths Problem called GOOD GADGET. Note that both nodes 1 and 2 prefer longer paths to shorter paths. The stable path assignment

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

illustrated in Figure 2 (b), is not a shortest path tree. This is the unique solution to this problem.

A modification of GOOD GADGET, called NAUGHTY GADGET, is shown in Figure 2 (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 in Section IV, the protocol SPVP can diverge for this problem. Finally, by reordering the ranking of paths at node 4, we produce a specification called BAD GADGET, presented in Figure 2 (d). This specification has no solution and the SPVP protocol will always diverge.

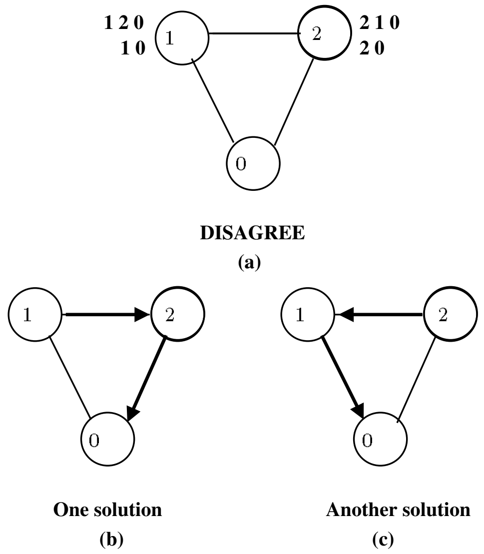


Fig. 3. DISAGREE and its two solutions.

So far, our examples each has had at most one solution. This is not always the case. The simplest instance, called DISAGREE, having more than one solution is illustrated in

Figure 3 (a). The stable path assignment

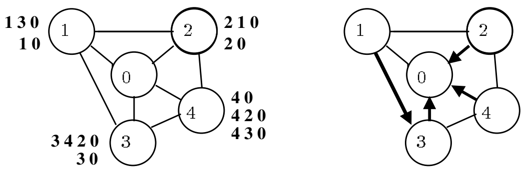
$$\pi_1 = ((1\ 2\ 0),\ (2\ 0)),$$

is depicted in Figure 3 (b). An alternative solution,

$$\pi_2 = ((1\ 0),\ (2\ 1\ 0)),$$

is shown in Figure 3 (c). No other path assignments are stable for this problem.

Figure 4 (a) describes a slight modification to BAD GADGET. The path $(4\ 0)$ is added and made the highest ranked path at node 4. The unique solution to this problem is illustrated in Figure 4 (b). Note that if the edge $\{0, 4\}$ is deleted, then this system becomes BAD GADGET. In terms of routing, this models the failure of link $\{0, 4\}$, and illustrates the fact that a network with a stable routing tree can be transformed into one with no solution with the failure of a single link.



(a) (b)

Fig. 4. BAD BACKUP

IV. A Simple Path Vector Protocol (SPVP)

This section presents a *Simple Path Vector Protocol* (SPVP) for solving the Stable Paths Problem in a distributed manner. SPVP represents an abstract version of the existing BGP protocol. This protocol always diverges when a Stable Paths Problem has no solution. It can also diverge for Stable Path Problems that are solvable. The protocol SPVP defined below differs from the simpler model of evaluation presented in [10], [11]. Here we use a message processing framework which employs a reliable FIFO queue of messages for communication between peers.

In SPVP, the messages exchanged between peers are simply paths. When a node u adopts a path $P \in \mathcal{P}^u$ it informs each $w \in$ peers(u) by sending path P to w. There are two data structures at each node u. The path rib(u) is u's current path to the origin. For each $w \in$ peers(u), rib-in$(u \Leftarrow w)$ stores the path sent from w most recently processed at u. The set of path choices available at node u is defined to be

$$\text{choices}(u) = \{(u\ w)P \in \mathcal{P}^u \mid P = \text{rib-in}(u \Leftarrow w)\},$$

and the best possible path at u is defined to be

$$\text{best}(u) = \text{best}(\text{choices}(u), u).$$

This path represents the highest ranked path possible for node u, given the messages received from its peers.

process spvp(u)
begin
  receive P **from** w $\longrightarrow$
    begin
      rib-in$(u \Leftarrow w) := P$
      if rib$(u) \neq \text{best}(u)$ **then**
      begin
        rib$(u) := \text{best}(u)$
        for each $v \in$ peers(u) **do**
        begin
          send rib(u) **to** v
        end
      end
    end
end

Fig. 5. The SPVP process at node u.

Figure 5 presents the process spvp(u) that runs at each node u. The notation and semantics are from [6]. If there is an unprocessed message from any $w \in$ peers(u), the guard **receive** P **from** w can be activated causing the message to be deleted from the incoming communication link and processed according to the program to the right of the arrow ($\longrightarrow$). We assume that this program is executed in one atomic step and that the communication channels are reliable and preserve message order. This protocol ensures that rib-in$(u \Leftarrow w)$ always contains the most recently processed message from peer w and that rib(u) is always the highest ranked path that u can adopt that is consistent with these paths.

The *network state* of the system is the collection of values rib(u), rib-in$(u \Leftarrow w)$, and the state of all communication links. It should be clear that any network state implicitly defines the path assignment $\pi(u) =$ rib(u). A network state is *stable* if all communication links are empty. In Section V-E it is shown that the path assignment associated with any stable state is always a stable path assignment, and thus a solution to the Stable Paths Problem. Therefore, if the Stable Paths Problem has no solution, then SPVP always diverges.

For example, consider BAD GADGET from Figure 2 (d). Using SPVP, it is easy to construct a sequence of network states that are associated with the path assignments of Figure 6. In this figure, an underlined path indicates that it has changed from the previous path assignment. Notice that this sequence begins and ends with the same path assignment and so represents one round of an oscillation.

step	π
0	(1 0) (2 0) (3 4 2 0) (4 2 0)
1	(1 0) $\underline{(2\ 1\ 0)}$ (3 4 2 0) (4 2 0)
2	(1 0) (2 1 0) (3 4 2 0) $\underline{\epsilon}$
3	(1 0) (2 1 0) $\underline{(3\ 0)}$ ϵ
4	(1 0) (2 1 0) (3 0) $\underline{(4\ 3\ 0)}$
5	$\underline{(1\ 3\ 0)}$ (2 1 0) (3 0) (4 3 0)
6	(1 3 0) $\underline{(2\ 0)}$ (3 0) (4 3 0)
7	(1 3 0) (2 0) (3 0) $\underline{(4\ 2\ 0)}$
8	(1 3 0) (2 0) $\underline{(3\ 4\ 2\ 0)}$ (4 2 0)
9	$\underline{(1\ 0)}$ (2 0) (3 4 2 0) (4 2 0)

Fig. 6. A sequence of path assignments for BAD GADGET.

A Stable Paths Problem is called *safe* if the protocol SPVP always converges. Note that SPP solvability does not imply safety. For example, NAUGHTY GADGET has a solution, but SPVP evaluation for this system can diverge. Whereas BAD GADGET is unable to converge, NAUGHTY GADGET can oscillate for an arbitrary amount of time before converging to a solution. In other words, NAUGHTY GADGET can produce both persistent and *transient* oscillations.

V. A SUFFICENT CONDITION FOR SPP SOLVABILITY, SAFETY, AND ROBUSTNESS

In this section we analyze the Stable Paths Problem. First, we show that determining if a solution exists is an NP-complete problem. We then define dispute wheels and show that the lack of dispute wheels is a sufficient condition which guarantees that a Stable Paths Problem has a unique solution. With respect to the protocol SPVP, we show that this sufficient condition also implies safety and robustness.

A. Complexity of SPP solvability

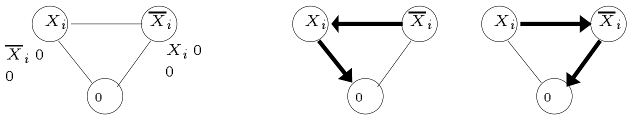
We now investigate the computational complexity of determining if a solution exists for an instance of the Stable Paths Problem. For a review of complexity theory, see [5].

Theorem V.1: The problem of determining whether an instance of the Stable Paths Problem is solvable is NP-complete.

Proof: We begin by noting that this problem is in NP, since we only need to *guess* a path assignment and check that it is indeed stable. This can clearly be done in time polynomial in the size of the instance of SPP.

The rest of the proof relies on a reduction from 3-SAT, a well-known NP-complete problem. An instance of 3-SAT consists of a set of boolean variables and a formula based on these variables and their negations where the formula has the form of a conjunction of terms each of which is a disjunction of three literals (a literal l is either a variable X or its negation $\overline{X}$). The 3-SAT problem asks if there exists a satisfying assignment for a given instance.

Suppose we are given an instance I of 3-SAT with variables $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$. We now construct an instance of the Stable Paths Problem $S(I)$ that is solvable if and only if I has a satisfying assignment.



(a) (b) X_i is **true** (c) X_i is **false**

Fig. 8. Variable assignment gadget for X_i.

For each variable X_i we use the structure of DISAGREE (Figure 3) to construct a "variable assignment gadget" shown in Figure 8 (a). The two distinct solutions of this gadget, depicted in Figure 8 (b) and (c), represent the assignment of X_i to **true** and **false**, respectively.

Given an arbitrary clause $C = l_1 \vee l_2 \vee l_3$ of I, the instance $S(I)$ contains a node labeled C. For each literal in C there is an edge from C to the corresponding node of the variable assignment gadget for the variable of that lit-

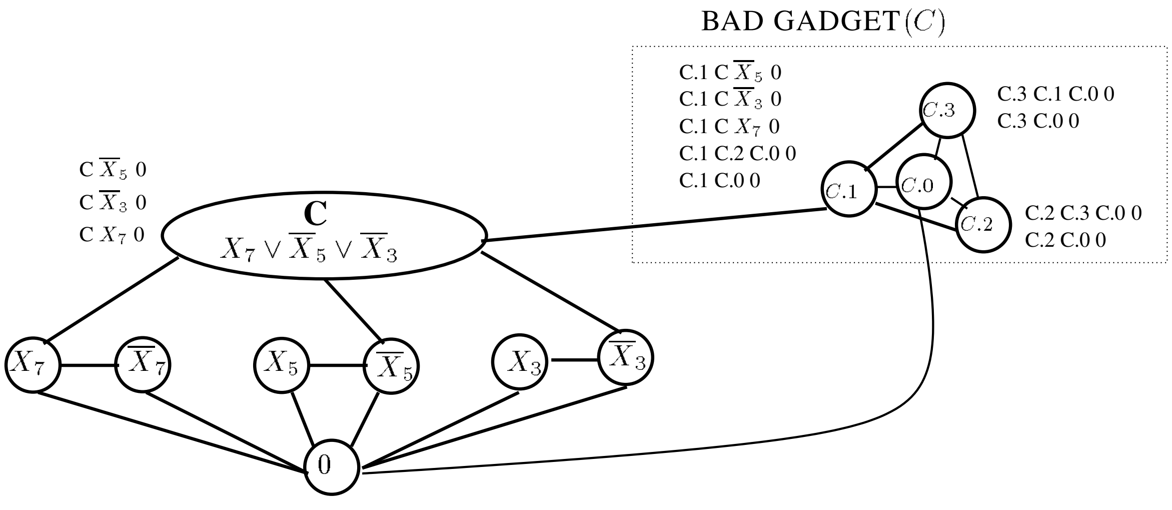


Fig. 7. Example of construction for clause $C = X_7 \vee \overline{X_5} \vee \overline{X_3}$

eral. The node C has only three permitted paths, each of length two, corresponding to the variable assignment that makes the literals **true**. (Note that the ranking is not important.) See Figure 7 for an illustration of this construction for three variables X_3, X_5, and X_7, and for one clause $C = X_7 \vee \overline{X_5} \vee \overline{X_3}$. For each clause C, a copy of a simplified BAD GADGET, called BAD GADGET(C), is attached as shown in Figure 7. It is clear that $S(I)$ is polynomial in the size of I.

We now show that I is satisfiable if and only if $S(I)$ has a solution. Suppose that the variable assignment function $A : \mathcal{X} \rightarrow \{\textbf{true}, \textbf{false}\}$ satisfies every clause of I. We now define a stable path assignment π_A for $S(I)$. First, we define π_A on variable assignment gadgets as

$$\pi_A(X_i) = \begin{cases} (X_i\ 0) & \text{if } A(X_i) = \textbf{true} \\ (X_i\ \overline{X_i}\ 0) & \text{if } A(X_i) = \textbf{false} \end{cases}$$

and

$$\pi_A(\overline{X_i}) = \begin{cases} (\overline{X_i}\ 0) & \text{if } A(X_i) = \textbf{false} \\ (\overline{X_i}\ X_i\ 0) & \text{if } A(X_i) = \textbf{true} \end{cases}$$

Suppose $C = l_1 \vee l_2 \vee l_3$ is a clause. Since A is a satisfying assignment, we know that at least one literal of C is true. Let l_j be the true literal such that the path $(C\ l_j\ 0)$ has the highest rank at C. Then let $\pi_A(C) = (C\ l_j\ 0)$. Finally, for this same C consider BAD GADGET(C). Let $\pi_A(C.1) = (C.1\ C\ l_j\ 0)$, $\pi_A(C.2) = (C.2\ C.3\ C.0\ 0)$, $\pi_A(C.3) = (C.3\ C.0\ 0)$, and $\pi_A(C.0) = (C.0\ 0)$. It is easy to check that π_A is a stable path assignment.

For the other direction, suppose that π is a stable path assignment for $S(I)$. We now construct a variable assignment $A_\pi : \mathcal{X} \rightarrow \{\textbf{true}, \textbf{false}\}$ that satisfies I. For each clause C, it must be the case that $\pi(C.1) = (C.1\ C\ l_j\ 0)$, for some literal l_j of clause C. If this were not the case, then π cannot be stable for at least one node in BAD GADGET(C). Since π is stable, we know that $\pi(C) = (C\ l_j\ 0)$ and that $\pi(l_j) = (l_j\ 0)$. Suppose that $l_j = X_i$ for some i. Then define $A_\pi(X_i) = \textbf{true}$. Otherwise, $l_j = \overline{X_i}$ for some i, and we define $A_\pi(X_i) = \textbf{false}$. If after considering each clause C there remains some unassigned variables, simply assign them the value **true**. The assignment A_π is well defined because we cannot have two clauses C and C' such that $\pi(C) = (C\ l_j\ 0)$ and $\pi(C') = (C'\ \overline{l_j}\ 0)$. Such a π could not be stable at l_j and $\overline{l_j}$. Since A_π assigns at least one literal for each clause the value **true**, we conclude that this is a satisfying assignment. ■

B. *Dispute Wheels*

Given the NP-completeness of the solvability problem for stable paths, we turn to developing a heuristic procedure. The procedure attempts to grow a stable path assignment (a routing tree) in a *greedy* manner.

Suppose $V' \subseteq V$, such that $0 \in V'$. A *partial path assignment* π for V' is a path assignment such that for every $u \in V'$, every node in $\pi(u)$ is in V'. The heuristic procedure constructs a sequence of subsets of V, $\{0\} = V_0 \subset V_1 \subset V_2 \ldots$, together with a sequence of partial path assignments $\pi_0,\ \pi_1,\ \pi_2, \ldots$, where each π_i is a partial path assignment for V_i. For each π_i, define $\hat{\pi}_i$ to be the path assignment for V, where $\hat{\pi}_i(u) = \pi(u)$ for $u \in V_i$, and $\hat{\pi}_i(u) = \epsilon$ for $u \notin V_i$. The partial path assignment π_i is *stable for* V_i if $\hat{\pi}_i$ is stable for each $u \in V_i$.

If $u \in V - V_i$ and $P \in \mathcal{P}^u$, then P is said to be *consistent with* π_i if it can be written as $P = P_1(u_1\ u_2)P_2$, where P_1 is a path in the digraph induced by $V - V_i$, $u_2 \in V_i$, and $P_2 = \pi(u_2)$, and $\{u_1,\ u_2\} \in E$. Such a P is called a *direct path to* V_i if P_1 is empty. Let D_i be the set of nodes $u \in V - V_i$ that have a direct path to V_i. Without loss of generality, each node has a non-empty permitted path to the origin, and hence if $V - V_i$ is not empty, then D_i is not empty. Let H_i be the set of nodes $u \in D_i$ whose highest ranked path consistent with V_i is a direct path. Denote this path as B_i^u. If H_i is not empty, let $V_{i+1} = V_i \cup H_i$. Define the partial path assignment π_{i+1} on V_{i+1} as

$$\pi_{i+1}(u) = \begin{cases} B_i^u & u \in H_i \\ \pi_i(u) & u \in V_i \end{cases}$$

This process continues until for some k either (1) $V_k = V$, or (2) $V_k \neq V$ and $H_k = \emptyset$. In the first case, π_k is clearly a stable path assignment. In the second case, we are *stuck*, and the procedure fails to find a solution.

If we perform this sequence of operations on GOOD GADGET (Figure 2 (a)), then it will arrive at the solution depicted in Figure 2 (b). However, for both NAUGHTY GADGET and BAD GADGET, this procedure will get stuck attempting to construct V_1 (that is H_0 is empty). This is because each node that has a direct path to $V_0 = \{0\}$, (nodes 1, 2, and 3), prefers a path that is not direct. We now show that getting stuck implies the existence of a circular set of conflicting rankings between nodes, which we call a *dispute wheel*.

Formally, a *dispute wheel,* $\Pi = (\vec{U}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$*, of size* k, is a sequence of nodes $\vec{U} = u_0, u_1, \cdots u_{k-1}$, and sequences of non-empty paths $\vec{\mathcal{Q}} = Q_0, Q_1, \cdots Q_{k-1}$ and $\vec{\mathcal{R}} = R_0, R_1, \cdots R_{k-1}$, 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) $R_iQ_{i+1} \in \mathcal{P}^{u_i}$, and (4) $\lambda^{u_i}(Q_i) \leq \lambda^{u_i}(R_iQ_{i+1})$. (All subscripts are to be interpreted modulo k.) See Figure 9 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.

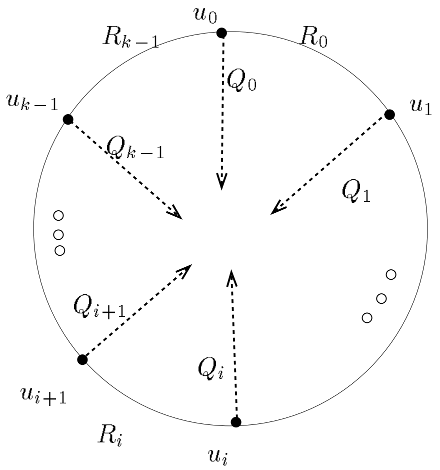
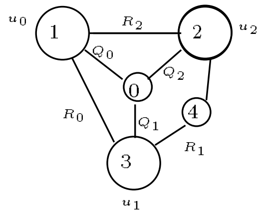
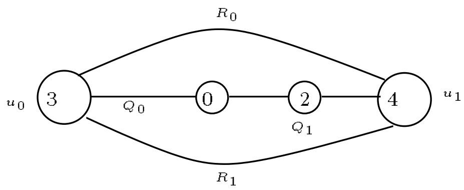


Fig. 9. A dispute wheel of size k.

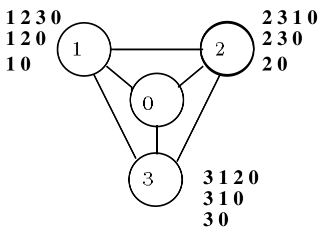
Both NAUGHTY GADGET and BAD GADGET of Figure 2 have this dispute wheel



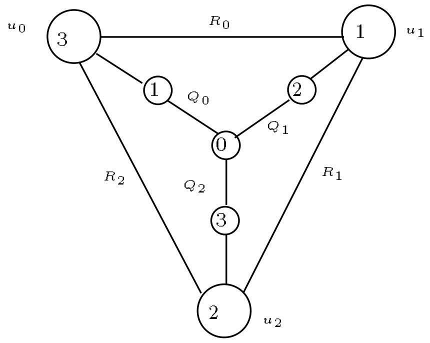
In addition, NAUGHTY GADGET has the dispute wheel



It may be the case that nodes of G appear multiple times in $\vec{U}$ and multiple times in any of the paths of $\vec{\mathcal{Q}}$ and $\vec{\mathcal{R}}$. For example, consider the SPP



This system has the following dispute wheel.



Note that nodes 1, 2, and 3 must be duplicated in order to present this dispute wheel in an "untangled" form.

C. No dispute wheel implies solvability

If Π is a dispute wheel, the triple resulting from *suppressing index* i is defined to be $\Pi' = (\vec{U}', \vec{Q}', \vec{\mathcal{R}}')$ where $\vec{U}'$ and $\vec{Q}'$ result from removing u_i from $\vec{U}$ and Q_i from $\vec{Q}$ and $\vec{R}' = R_0, \cdots R_{i-2}, R', R_{i+1}, \cdots, R_{k-1}$, where $R' = R_{i-1}R_i$. A *sub-wheel* of Π is any dispute wheel obtained by a sequence of such operations. 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 V.2: Every dispute wheel contains a minimal sub-wheel.

Proof: Suppose that dispute wheel Π 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 by suppressing index $i+1$. Repeating this process must eventually arrive at a minimal sub-wheel. ■

Theorem V.3: Let S be an instance of the Stable Paths Problem. If S has no dispute wheel, then S is solvable.

Proof: Suppose that our heuristic procedure gets stuck at step i. Let u_0 be any node in D_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 V_i, which has higher rank than Q_0. Since P_0 is consistent with V_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 $V - V_i$, $v_1 \in V_i$, Q_1 is $\pi_i(v_1)$. and $\{u_1,\ v_1\} \in E$. Note that $v_1 \in D_i$, and since H_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. ■

Note that BAD BACKUP is solvable and yet has a dispute wheel.

D. No dispute wheel implies a unique solution

In general, an instance of the Stable Paths Problem may have more than one solution. We show that in this case the problem has a dispute wheel.

Theorem V.4: If the Stable Paths Problem S has no dispute wheel, then it has a unique solution.

Proof: Suppose that S has no dispute wheel, and has two distinct solutions, $\pi_1 = (P_1, \ldots, P_{n-1})$ and $\pi_2 = (Q_1, \ldots, Q_{n-1})$. Let T_1 and T_2 be the trees, rooted at node 0, that are defined by the non-empty paths of π_1 and π_2 respectively. 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 $T_1 \cup T_2$ entering $V(T)$ is either in $E(T_1) - E(T_2)$ or $E(T_2) - E(T_1)$. See Figure 10 for an illustration.

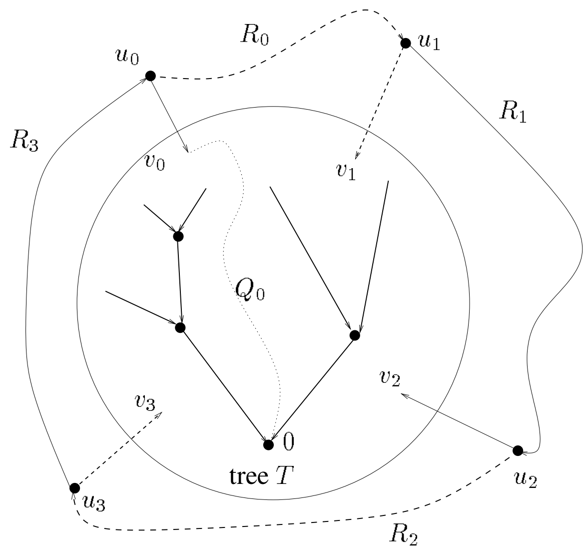


Fig. 10. Illustration for Theorem V.4.

We now construct a dispute wheel. Note that $T_1 \neq T_2$ implies that $V - V(T)$ is nonempty, and that at least one of the trees has an edge entering $V(T)$. Without loss of generality, consider any $\{u,\ v\}$ in T_1 where v is in T, and u is not. Note that u must be in T_2, otherwise it would have the empty path in π_2, which it cannot prefer to the path $(u,\ v)Q_v$. 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 has a unique solution and has a dispute wheel.

E. No dispute wheel implies safety

We now show that the protocol SPVP can never diverge for an instance of the Stable Paths Problem that has no dispute wheel.

We model (logical) time t with discrete values $0, 1, 2, \ldots$. For each node u and each $w \in$ peers(u), mq$(u \Leftarrow w, t)$ denotes the state of the communication link from node w to node u at time t. This is a FIFO message queue, and the notation mq$(u \Leftarrow w, t)[i]$ refers to the ith element in the queue. In particular, mq$(u \Leftarrow w, t)[1]$ is the first element, or the oldest unprocessed message in the communication link. If k is the number of messages in mq$(u \Leftarrow w, t)$, then mq$(u \Leftarrow w, t)[k]$ denotes the last element, or the message most recently sent from w to u. For each u, rib(u, t) denotes the value of rib(u) at time t. For each u and each $w \in$ peers(u), rib-in$(u \Leftarrow w, t)$ denotes the value of rib-in$(u \Leftarrow w)$ at time t. For ease of presentation, we define the *pipe* from node w to u at time t, pipe$(u \Leftarrow w, t)$, to be the message queue obtained by inserting rib-in$(u \Leftarrow w, t)$ into mq$(u \Leftarrow w, t)$ before the first position. In other words, pipe$(u \Leftarrow w, t)[1]$ = rib-in$(u \Leftarrow w, t)$ and pipe$(u \Leftarrow w, t)[i + 1]$ = mq$(u \Leftarrow w, t)[i]$, for $1 \leq i \leq k$, where k is the number of messages in mq$(u \Leftarrow w, t)$.

The *network state at time* t, denoted by $s(t)$, is comprised of all values rib(u, t), rib-in$(u \Leftarrow w, t)$, and mq$(u \Leftarrow w, t)$. Suppose $\pi = (P_1,\ P_2,\ \cdots,\ P_n)$ is a path assignment. An initial state *induced* by π is the state where each queue mq$(u \Leftarrow w)$ contains the single message P_w, each rib-in$(u \Leftarrow w) = \epsilon$, and each rib$(u) = P_u$.

At each state transition from $s(t-1)$ to $s(t)$, either (1) the network state remains unchanged, or (2) some node u processes a message from some $w \in$ peers(u). If node u changes its path in this transition from P_{old} to P_{new}, we say that u *adopted* path P_{new} at time t. We will encode an arbitrary run in an *activation sequence* σ, where $\sigma(t)$ = no-op, or $\sigma(t)$ = recompute(u, w). If $\sigma(t)$ = no-op, then the state remained unchanged in the transition from state $s(t-1)$ to $s(t)$. If $\sigma(t)$ = recompute(u, w), then node u processed one message from $w \in$ peers(u). We write $s(t-1) \xrightarrow{\sigma(t)} s(t)$ to denote this transformation. If $t_1 < t_2$, the notation $s(t_1) \xrightarrow{\sigma} s(t_2)$ denotes the composition of one-step transitions $s(t_1) \xrightarrow{\sigma(t_1+1)} s(t_1+1) \xrightarrow{\sigma(t_1+2)} s(t_1+2) \xrightarrow{\sigma(t_1+3)} \cdots \xrightarrow{\sigma(t_2)} s(t_2)$.

Let $s_0 = s(0)$ be some initial state. An activation sequence σ is *fair* with respect to s_0 if any message sent from w to u will eventually be received and processed by u, assuming the system started in state s_0. In other words, if $s_0 \xrightarrow{\sigma} s(t_1)$ and mq$(u \Leftarrow w, t_1)$ is not empty, then there is a time $t_2 > t_1$ such that $s(t_1) \xrightarrow{\sigma} s(t_2)$ and $\sigma(t_2)$ = recompute(u, w).

Let $S = (G,\ \mathcal{P},\ \Lambda)$ be an instance of the Stable Paths Problem. If at time t the network state $s(t)$ is such that all message queues mq$(u \Leftarrow v, t)$ are empty, then we say that the system has *converged* at time t, and write $S(\sigma, s_0, t) \downarrow$, where s_0 is the initial state ($s_0 = s(0)$). If the system does not converge for any time t we say the system diverges, and write $S(\sigma, s_0) \uparrow$.

We now define the notion of a consistent network state. The state at time t is *rib consistent* if for all u, rib(u, t) is the best path possible, given the values of rib-in$(u \Leftarrow w, t)$, for $w \in$ peers(u). We say that pipe$(u \Leftarrow w, t)$ is *pipe consistent* if pipe$(u \Leftarrow w, t)[k]$ = rib(w, t), where k is the number of messages in pipe$(u \Leftarrow w, t)$. Note that this implies that if pipe$(u \Leftarrow w, t)$ contains only one message, then it is identical to rib(w, t). In particular, if the communication links mq$(u \Leftarrow w, t)$ are empty, then rib-in$(u \Leftarrow w, t)$ = rib(w, t). A state $s(t)$ is *consistent* if it is rib consistent and all pipes are pipe-consistent.

We now show that consistency is preserved under state transitions.

Lemma V.5: Let σ be an activation sequence. Suppose that $s(t)$ is a consistent state and $s(t) \xrightarrow{\sigma(t)} s(t+1)$. Then $s(t+1)$ is a consistent state.

Proof: Obvious. ■

Theorem V.6 (Correctness) Let s_0 be a consistent state and σ an activation sequence that is fair with respect to s_0. Suppose that for some time t we have $S(\sigma, s_0, t) \downarrow$. Let $\vec{P} = (P_1, \cdots, P_n)$ where rib$(i,t) = P_i$. Then $\vec{P}$ is a solution for the specification S.

Proof: By repeated application of Lemma V.5 we know that the state at time t is consistent, and since the system has converged we know that all communication links are empty. By pipe-consistency, we know that if i and j are peers, then rib-in$(i \Leftarrow j, t) =$ rib$(j,t) = P_j$. Therefore, if $\vec{P}$ is not a solution for S, then there is some node i that is not rib-consistent, which is a contradiction. ■

Suppose s_0 is a consistent state, σ is a fair activation sequence with respect to s_0, and that $S(\sigma, s_0) \uparrow$. The *set of converging nodes*, $\mathcal{C} \subset V$, are those nodes u such that for some time t and for all $t' \geq t$, we have rib$(u, t') =$ rib(u, t). The *oscillating nodes*, denoted $\mathcal{O}$, is the set of nodes in V not in $\mathcal{C}$.

By the definition of $\mathcal{C}$ we can define a time t_c such that for all $t \geq t_c$ and for all $u \in \mathcal{C}$, rib$(u,t) =$ rib(u, t_c). If $u \in \mathcal{C}$ and w is a peer of u, then after time t_c no new messages are placed into pipe$(w \Leftarrow u)$ and so by the fairness of σ there is a time $t_f > t_c$ such that for all times $t > t_f$ all such messages from nodes in $\mathcal{C}$ have been flushed from all communication links. In particular, for all $t > t_f$ and all $u \in \mathcal{C}$, pipe$(w \Leftarrow u, t) =$ rib-in$(w \Leftarrow u, t) =$ rib(u,t) for all peers w of u. For $u \in \mathcal{C}$, let m^u be the fixed message in rib-in$(w \Leftarrow u, t)$ for all peers w of u and hence the message in rib(u,t) for all $t > t_f$.

For every $u \in \mathcal{O}$ define $\mathrm{values}(\sigma, s_0, u)$ to be the set of paths that u adopts infinitely often. For every $w \in \mathcal{C}$ define $\mathrm{values}(\sigma, s_0, w)$ to be the singleton set $\{\text{rib}(w, t_c)\}$. Let t_F be the time after which each $u \in \mathcal{O}$ adopts only paths in $\mathrm{values}(\sigma, s_0, u)$. 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 $(v_i\ v_{i-1}\ \ldots\ v_j)$.

Lemma V.7: For $w \in V$, suppose that $P \notin \mathrm{values}(\sigma, s_0, w)$. Then there is a time t after which there is no path of the form QP in the network state.

Proof: By definition, there must be a time t after which all nodes $w \in V$ adopt only paths $P \in \mathrm{values}(\sigma, s_0, w)$. Since σ a fair activation sequence, we know that there is some time $t' > t$ after which all communication links have been renewed. ■

Lemma V.8: Suppose $P \in \mathrm{values}(\sigma, s_0, u)$ for some $u \in \mathcal{O}$. If $w \neq u$ is a node in P and $w \in \mathcal{O}$, then $P[w, 0] \in \mathrm{values}(\sigma, s_0, w)$. In addition, if v is a node in P and $v \in \mathcal{C}$, then $P[v, 0] =$ rib(v, t_c).

Proof: Let $w \neq u$ be a node in P such that $w \in \mathcal{O}$. Suppose that $P[w, 0] \notin \mathrm{values}(\sigma, s_0, w)$. By Lemma V.7, there is a time t after which there is no path of the form QP in the network state. Therefore, u cannot adopt this path infinitely often, which is a contradiction. A similar argument holds for the case where v is a node in P and $v \in \mathcal{C}$. ■

Theorem V.9: If S has no dispute wheel, then S is safe.

Proof: Suppose that S diverges, $S(\sigma, s_0) \uparrow$. We show that S contains a dispute wheel. Let $\mathcal{O}$, $\mathcal{C}$, and t_f be defined as above. Let t be any time $t > t_f$. Let U be the subset of nodes $u \in \mathcal{O}$ such that there is a path $(u\ w)Q \in \mathrm{values}(\sigma, s_0, u)t$ where $w \in \mathcal{C}$. That is, each u in U adopts a path that leads directly to a fixed node. By Lemma V.8, 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 $\mathcal{C}$, $(u_0\ w_0)Q'_0$. It is easy to check that Q_0 is unique, and that of all paths in $\mathrm{values}(\sigma, s_0, u_0)t$ the path Q_0 is of lowest rank. Let $H_0 \in \mathrm{values}(\sigma, s_0, u_0)t$ be the adopted path of highest rank at u_0. Lemma V.8 tells us that we can write this path as $H_0 = R_0 Q_1$, 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 $w_1 \in \mathcal{C}$. We can now perform the same construction for u_1. Repeating this process in the obvious way results in a dispute wheel. ■

F. No dispute wheel implies robustness

We model the failure of an arbitrary number of links as follows. Let $S = (G,\ \mathcal{P},\ \Lambda)$ be an instance of the Stable Paths Problem where $G = (E,\ V)$. Suppose $E' \subset E$. We define S/E' to be the stable paths problem obtained by (1) deleting the edges E' from the graph G, (2) removing all permitted paths that traverse an edge in E', and (3) amending the ranking functions accordingly. The problem S is *fragile* if S is solvable but there exists some $E' \subset E$ such that S/E' is not solvable. The problem S is *robust* if S is safe and for each $E' \subset E$ the problem S/E' is also safe. The system GOOD GADGET of Figure 2 (a) is robust, while BAD BACKUP of Figure 4 is fragile.

Theorem V.10: Let S be an instance of the Stable Paths Problem. If S has no dispute wheel, then S is robust.

Proof: Suppose that S has no dispute wheel. From TheoremV.9, we know that S is safe. Suppose that $E' \subset E$. If S/E' is not safe, then by TheoremV.9 there must be a dispute wheel for S/E'. But any dispute wheel for S/E' is also a dispute wheel for S, which is a contradiction. Hence, S is robust. ■

VI. Stable Paths and Shortest Paths

Varadhan *et al.* [21] 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 a protocol such as BGP. We then show that any instance of the Stable Paths Problem that is consistent with shortest path route selection will indeed be safe. However, we show BGP-like systems can actually violate "distance metrics" and remain still safe.

As is standard for undirected graphs, we work with an *associated digraph*, where each undirected edge $e = \{a, b\}$ is 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 digraph: $c(P) = \sum_{a \in A(P)} c(a)$. 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 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 an instance of the Stable Paths Problem, $S = (G, \mathcal{P}, \Lambda)$, 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 a cost function c has negative directed cycles, then S can be consistent with c and yet not be safe. For example, consider the costs attached to the edges of NAUGHTY GADGET in Figure 11, where the cost of traversing an edge is the same in each direction. NAUGHTY GADGET is consistent with this cost function, but it 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.

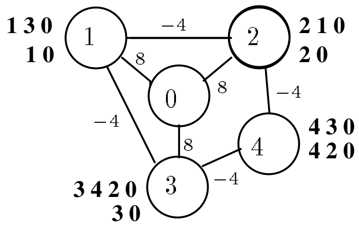


Fig. 11. NAUGHTY GADGET with negative link costs

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 VI.1: 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 Theorem V.9 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.

Note that the system INCOHERENT of Figure 12 has no dispute wheel, and hence is safe, yet 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 $(1\ 2\ 3\ 1)$. That is, INCOHERENT is not consistent with any coherent cost function.

In summary, the class of Stable Path Problems no dispute wheels is provably safe, yet it is strictly larger than those based on shortest paths.

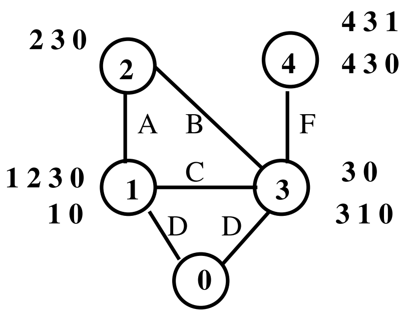


Fig. 12. The system INCOHERENT

VII. Discussion and open problems

Is it possible to guarantee that BGP will not diverge? Broadly speaking, there are three complementary approaches to addressing this problem : (1) *operational guidelines*, (2) *static analysis of routing policies*, and (3) *dynamic detection*. We briefly discuss each of these techniques.

A set of *operational guidelines* is a collection of rules that should be followed by every autonomous system. One use of the framework presented in the current paper is to prove that a given collection of rules will indeed guarantee safe BGP policies. For example, using the results of Section VI it is easy to see that any set of BGP policies that can be implemented using route filtering alone will be safe. This includes standard policies that determine which routes should be imported from and exported to customers, peers, and upstream providers [15]. A more elaborate set of guidelines, together with correctness proofs, can be found in [4]. One difficulty with this approach is that many Internet Service Providers (ISPs) are in fact composed of multiple autonomous systems. Restrictions that make economic sense when we think of autonomous systems as independent ISPs may no longer hold when they are all owned by the same company. The *member autonomous systems* of BGP confederations [20] can be considered as a special case of this kind of multi-AS service provider.

A solution based on *static analysis* 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.* [9]. 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 [11] have 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 [22] 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*.

Such an extension is presented in [12]. This is done by adding a dynamically computed attribute to SPVP called the *path history*. Protocol oscillations caused by policy conflicts produce paths whose histories contain cycles. These cycles correspond to dispute wheels, and identify the policy conflicts and the nodes systems involved. This protocol can be further extended to automatically suppress those paths whose histories contain cycles. This guarantees that the resulting protocol can never diverge.

There are several open problems that need to be addressed. The computational complexity of deciding safety or robustness for an SPP specification remains open. Our treatment has ignored the complexities of interior BGP (IBGP), such as route reflectors and confederations. We have also ignored address aggregation. These issues need to be addressed in a more complete model of BGP.

In this paper we have studied the stable paths as a computational problem. However, the stable paths problem could be studied in the context of a multi-person *repeated game* where each node corresponds to a player and each subgame requires every node i to choose a path from the set of permitted paths at i, $\mathcal{P}^i$. We do not define this game in its most formal terms (see [3] for an introduction to game theory), but rather give a slight simplification of the strategy sets for the players. A *pure strategy* for node i, is a function $\Psi^i : \mathbf{N} \times \mathcal{P}^1 \times \mathcal{P}^2 \times \ldots \times \mathcal{P}^n \rightarrow \mathcal{P}^i$ where $\Psi^i(t, p_1, p_2, \ldots, p_n) = (i, j, p)$, then we must have $p_j = p$. The interpretation is that if at time t, each node j has chosen the path p_j, then $\Psi^i(t, p_1, p_2, \ldots, p_n)$ determines the path which node i will adopt at time $t + 1$. A *play* of the game corresponds to each node i fixing some pure strategy and then playing each subgame $t = 1, 2, \ldots$ (we may assume that each path stores the empty path at time 0) and updating the paths stored at each node accordingly. The *payoff* for node i after game t is simply the rank of the path it stores at that time. A *(pure) Nash equilibrium* for the game corresponds to a play of the game where for some t_0, we have that $\Psi^i(t, p_1, p_2, \ldots, p_n) = p_i$ for each node i and $t \geq t_0$. We note that a mixed strategy for a player corresponds to some collection $\mathcal{S}$ of pure strategies for that player and an assignment $\lambda : \mathcal{S} \rightarrow \mathbf{R}_+$ such that $\sum_S \lambda(S) = 1$; thus the player will use the strategy S with probability $\lambda(S)$. Finally, we remark that BGP defines a unique pure strategy for each player which it must then use always. Namely, a node must always choose its best path amongst those available. Thus a player's strategy is time independent, and so it can only alter its strategy (and hence any equilibrium adopted) by changing the ranking of its paths.

References

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

[2] K. Bhargavan, D. Obradovic, and C. Gunter. Formal verification of standards for distance vector routing protocols. UPenn Tech Report, 1999.

[3] K. Binmore. *Fun and Games*. D.C. Heath and Company, Lexington, Mass., 1992.

[4] L. Gao and J. Rexford. Stable internet routing without global coordination. In *SIGMETRICS 2000*, 2000.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, CA, 1979.

[6] M. G. Gouda. *Elements of Network Protocol Design*. John Wiley & Sons, Inc., 1998.

[7] M.G. Gouda and M. Schneider. Maximizable routing metrics. In *Proc. Sixth International Conference on Network Protocols (ICNP'98*, pages 71–78, 1998.

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

[9] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W.S. Lee. An architecture for stable, analyzable internet routing. *IEEE Network*, 13(1):29–35, 1999.

[10] T. Griffin, F.B. Shepherd, and G. Wilfong. Policy disputes in path-vector protocols. In *Proc. Seventh International Conference on Network Protocols (ICNP'99)*, pages 21–30, 1999.

[11] T. Griffin and G. Wilfong. An analysis of BGP convergence properties. In *SIGCOMM'99*, pages 277 – 288, 1999.

[12] T. Griffin and G. Wilfong. A safe path vector protocol. In *INFOCOM2000*, 2000.

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

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

[15] G. Huston. *ISP Survival Guide*. John Wiley & Sons, Inc., 1999.

[16] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. In *SIGCOMM'97*, 1997.

[17] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *INFOCOM'99*, 1999.

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

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

[20] P. Traina. Autonomous systems confederations for BGP. RFC 1965, 1996.

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

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