A Safe Path Vector Protocol

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

Abstract—**An IP routing protocol is *safe* if it is guaranteed to converge in the absence of network topology changes. BGP, currently the only interdomain routing protocol employed on the Internet, is not safe in this sense. It may seem that the source of BGP's potential divergence is inherent in the requirements for any interdomain routing protocol — policy-based metrics must be allowed to override distance-based metrics, and each autonomous system must be allowed to independently define its routing policies with little or no global coordination. In this paper we present a *Simple Path Vector Protocol* (SPVP) that captures the underlying semantics of BGP by abstracting away all nonessential details. We then add a dynamically computed attribute to SPVP routing messages, called the *route history*. Protocol oscillations caused by policy conflicts produce routes whose histories contain cycles. These cycles identify the policy conflicts and the autonomous systems involved. SPVP is made safe by automatically suppressing routes whose histories contain cycles. We discuss how this safe SPVP can be used in the design of a safe BGP.**

I. INTRODUCTION

The Border Gateway Protocol, BGP, is currently the only interdomain routing protocol employed on the Internet [12], [7], [13]. 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 [8], [1], Varadhan *et al.* [14] have shown that the routing policies of autonomous systems can conflict in a manner that causes BGP to diverge, resulting in persistent route oscillations. Recent studies have highlighted the the adverse effects of interdomain routing instability [9], [10]. 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.

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.* [3]. 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 [5] have shown that checking for various global convergence conditions is either NP-complete or NP-hard.

A *dynamic* solution to the BGP divergence problem is some mechanism to suppress or completely prevent at *run time* those BGP oscillations that arise from policy conflicts. Using route flap dampening [15] as a dynamic mechanism to address this problem has two distinct drawbacks. First, route flap dampening cannot eliminate policy-based oscillations, it can only make these oscillations run in *slow motion*. Second, route flap dampening does not provide network administrators with enough information to identify the root cause of route flapping. Route flaps caused by policy conflicts cannot be distinguished from route flaps caused by other problems such as unstable routers or defective network interfaces. This suggests that a dynamic solution requires an *extension* to the BGP protocol to carry additional information that would allow policy-based oscillations to be detected and identified at run time. We present such an extension for an idealized version of BGP, called SPVP (Simple Path Vector Protocol).

The paper is organized as follows. In Section II we define the *Stable Paths Problem* (SPP) that is intended to capture the problem underlying the BGP protocol. Routing protocols such as RIP [8] and OSPF [11] are techniques for solving the *Shortest Paths Problem* in a distributed manner. We claim that the corresponding problem being solved by BGP is the Stable Paths Problem. Next, in Section III, we present SPVP_1, a protocol that implements a distributed algorithm for computing solutions to the Stable Paths Problem. This protocol represents an abstraction of the current BGP protocol. SPVP_1 does not always find a solution. Indeed, we show that if a Stable Paths Problem has no solution, then the protocol will never converge. We then extend this protocol to SPVP_2 by adding a dynamically computed attribute called the *path history*. Protocol oscillations caused by policy conflicts produce paths whose histories contain cycles. These cycles identify the policy conflicts and the autonomous systems involved. Finally, we extend SPVP_2 to a safe protocol, SPVP_3, that automatically suppresses those paths whose histories contain cycles.

In Section IV we prove *soundness* and *completeness* theorems for SPVP_2.

Soundness: The detection of a cycle in a path history means that a policy-based oscillation has been dynamically realized, and the cycle describes the policy conflicts causing the oscillation.
Completeness: If a policy-based protocol oscillation persists, then all oscillating nodes will eventually detect cycles in their path histories.

The notions of *policy conflicts* and *policy-based protocol oscillation* are formalized using the *dispute digraph* defined in [4]. We show that the dynamically computed path histories correspond exactly to paths in the dispute digraph. Thus history cycles correspond to cycles in the dispute digraph that describe a *circularity* in the routing policies of some collection of nodes. Finally, we show that SPVP_3 is safe.

Section V outlines how the SPVP framework can be used to design a protocol extension that makes BGP safe. Unlike route flap dampening, the path history supplies network administrators with enough information (1) to identify policy-induced oscillation, (2) to identify those autonomous systems involved in the dispute cycle, and (3) to identify the *policy conflicts* causing oscillation. Section VI concludes with open problems.

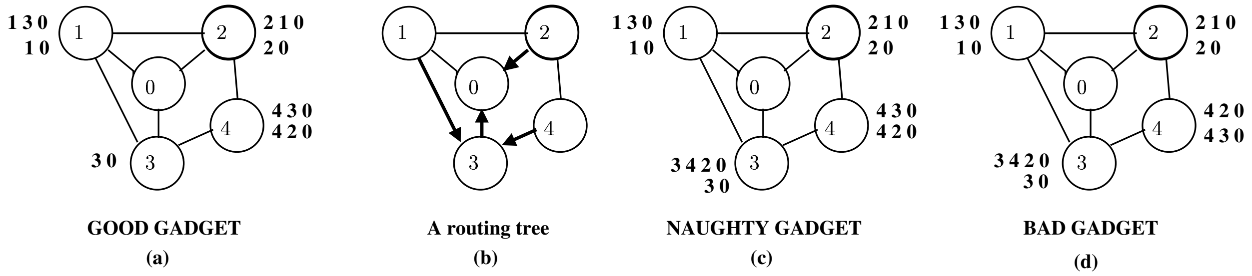

Fig. 1. Examples of Stable Paths Problems.

II. The Stable Paths Problem

The Stable Paths Problem (SPP) provides a simple semantics for routing policies of 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. Numerous BGP-specific details (internal BGP, confederations, route servers, private AS numbers, and so on) are pushed into the translation, which can become quite complex. Section V explores modeling BGP in our framework.

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.

We now formalize these concepts. 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 attempt to establish a path.

A *path* in G is either the empty path, denoted ϵ, or a sequence of nodes, $(v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$, such that for each i, $k \geq i > 0$, $\{v_i, v_{i-1}\}$ is an edge in E. We assume that all non-empty paths $P = (v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$ 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 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 $v \in V - \{0\}$, the set $\mathcal{P}^v$ denotes the *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* for path P. Let $\mathcal{P} = \{\mathcal{P}^v \mid v \in V - \{0\}\}$ be the set of all permitted paths.

For each $v \in V - \{0\}$, 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 non-zero node and the ranking functions for each non-zero node, where the following restrictions hold on Λ and $\mathcal{P}$.

(empty path is permitted) $\epsilon \in \mathcal{P}^v$,
(empty path is lowest ranked) $\lambda^v(\epsilon) = 0$,
(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),

Let $S = (G, \mathcal{P}, \Lambda)$ be an instance of the Stable Paths Problem. 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 *maximal path in* W, $\max(u, W)$, is defined to be

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

A *path assignment* is a function π that maps each node $u \in V$ to a path $\pi(u) \in \mathcal{P}^u$. Note that $\pi(u)$ may be the empty path. The set of paths $\text{choices}(u, \pi)$ is defined to be all $P \in \mathcal{P}^u$ such that either $P = (u\ 0)$ and $\{u, 0\} \in E$ or $P = (u\ v)\pi(v)$ for some $\{u, v\} \in E$. The path assignment π is *stable at node* u if

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

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 $\pi(u) = P_u$.

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

Figure 1 (a) presents a Stable Paths Problem 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, omitting the empty path. The path assignment

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

is illustrated in Figure 1 (b). This is the unique solution to this problem since no other path assignment is stable. For example, the path assignment

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

is not stable since it is not stable for nodes 1 and 2. To see this, note that

$$\begin{aligned} \mathrm{choices}(1, \pi) &= \{(1\,0),\ (1\,3\,0))\} \\ \mathrm{choices}(2, \pi) &= \{(2\,0),\ (2\,1\,0))\}, \end{aligned}$$

and so $\max(1, \mathrm{choices}(1, \pi)) = (1\ 3\ 0) \neq \pi(1)$ and $\max(1, \mathrm{choices}(2, \pi)) = (2\,1\,0) \neq \pi(2)$.

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 in Section III, the distributed evaluation of this specification (using protocols SPVP_1 and SPVP_2) 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 has no solution and its distributed evaluation (using protocols SPVP_1 and SPVP_2) will always diverge.

III. THREE SIMPLE PATH VECTOR PROTOCOLS

This section presents three *Simple Path Vector Protocols* (SPVPs) for solving the Stable Paths Problem in a distributed manner. The first, SPVP_1, is an abstract version of the existing BGP protocol and represents the current state of the art. This protocol will always diverge when a Stable Paths Problem has no solution. It can also diverge for Stable Path Problems that are solvable. We modify this protocol to arrive at SPVP_2. This protocol adds a dynamically computed attribute to messages, called the *path history*. As with SPVP_1, SPVP_2 is not safe. However, protocol oscillations caused by policy conflicts produce paths whose histories contain "event cycles". These cycles identify the policy conflicts and the network nodes involved. Finally, we produce SPVP_3, a safe extension to SPVP_2, which automatically suppresses paths whose histories contain cycles.

The protocol SPVP_1 defined below differs from the simpler model of evaluation presented in [5], [4]. Here we use a message processing framework which employs a reliable FIFO queue of messages for communication between peers. The use of histories described below will not function correctly without message queues that preserve message ordering and ensure no message loss. This is consistent with implementations of BGP which use TCP and message queues to implement this abstraction.

A. *The Status Quo : $SPVP_1$*

In SPVP_1, the messages exchanged between peers are simply paths. When a node u adopts a path $P \in \mathcal{P}^u$ it informs each $w \in \mathrm{peers}(u)$ by sending path P to w. There are two data structures at each node u. The path $\mathrm{rib}(u)$ is u's current path to the origin. For each $w \in \mathrm{peers}(u)$, $\text{rib-in}(u \Leftarrow w)$ stores the path sent from w most recently processed at u. The set of path choices available at node u is defined to be

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

and the best possible path at u is defined to be

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

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

process $\mathrm{spvp}_1[u]$
begin
    receive P **from** w $\longrightarrow$
        begin
            $\text{rib-in}(u \Leftarrow w) := P$
            if $\mathrm{rib}(u) \neq \mathrm{best}(u)$ **then**
            begin
                $\mathrm{rib}(u) := \mathrm{best}(u)$
                for each $v \in \mathrm{peers}(u)$ **do**
                begin
                    send $\mathrm{rib}(u)$ **to** v
                end
            end
        end
end

Fig. 2. The SPVP_1 process at node u.

Figure 2 presents the process $\mathrm{spvp}_1[u]$ that runs at each node u. The notation and semantics are from [2]. If there is an unprocessed message from any $w \in \mathrm{peers}(u)$, 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 $\text{rib-in}(u \Leftarrow w)$ always contains the most recently processed message from peer w and that $\mathrm{rib}(u)$ 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 $\mathrm{rib}(u)$, $\text{rib-in}(u \Leftarrow w)$, and the state of all communication links. It should be clear that any network state implicitly defines the path assignment $\pi(u) = \mathrm{rib}(u)$. A network state is *stable* if all communication links are empty. In Section IV 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 S. However, the converse of this theorem tells us that SPVP is not safe. That is, if S has no solution, then SPVP cannot converge to a stable state.

For example, consider BAD GADGET from Figure 1 (d). Using SPVP_1, it is easy to construct a sequence of network states that are associated with the path assignments of Figure 3. 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.

Note that even if a solution exists, SPVP_1 is not guaranteed to converge. Such a sequence of states can also be constructed for NAUGHTY GADGET, which has a solution. Whereas BAD GADGET is unable to exit this oscillation, NAUGHTY GADGET

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. 3. A sequence of path assignments for BAD GADGET.

can oscillate for an arbitrary amount of time before converging on a solution. In other words, NAUGHTY GADGET can produce both persistent and *transient* oscillations.

B. Dynamically computing histories with SPVP$_2$

In Figure 3, we see that node 2 has path $(2\ 0)$ at step 6. Can we *explain* the sequence of events that led 2 to adopt this path? Suppose that node u transitions from $\text{rib}(u) = P_{\text{old}}$ to $\text{rib}(u) = P_{\text{new}}$. We say that node u *went down from P_{old}* if u ranks path P_{new} lower than path P_{old}. Similarly, we say that node u *went up to path P_{new}* if u ranks path P_{old} lower than path P_{new}. We can now explain node 2's path $(2\ 0)$ at step 6 this way: *node* 2 *adopted path* $(2\ 0)$ *because it went down from* $(2\ 1\ 0)$ *because node* 1 *went up to* $(1\ 3\ 0)$ *because node* 3 *went down from* $(3\ 4\ 2\ 0)$ *because node* 4 *went down from* $(4\ 2\ 0)$ *because node* 2 *went up to* $(2\ 1\ 0)$. Thus, it is easy to see that there is a circularity involved — 2 went down from $(2\ 1\ 0)$ because $\cdots$ 2 went up to $(2\ 1\ 0)$.

We define a *path change event* to be a pair $e = (s,\ P)$ where $s \in \{+, -\}$ is the *sign* of e and P is a path. If e is a path change event, then $\text{sign}(e)$ denotes the sign of e. Suppose P_{old} and P_{new} are paths permitted at node u. Then the event $(+,\ P_{\text{new}})$ means that u went up to P_{new}, and the event $(-,\ P_{\text{old}})$ means that u went down from P_{old}. A *path history* h is either the empty history $\diamond$, or a sequence $h = e_k\ e_{k-1}\ \cdots\ e_1$, where each e_i is a path change event. Intuitively, event e_{i+1} occurred after event e_i, so the most recent event is e_k. Such a history *contains a cycle* if there exists i, j, with $k \geq j > i \geq 1$, such that $e_i = (s_1,\ P)$ and $e_j = (s_2,\ P)$. For ease of notation, we usually write a path change event such as $(-,\ (3\ 4\ 2\ 0))$ in the simpler form $(-\ 3\ 4\ 2\ 0)$.

Using this notation we can restate our explanation for the path $(2\ 0)$ at node 2 with the history

$$(-\ 2\ 1\ 0)\ \ (+\ 1\ 3\ 0)\ \ (-\ 3\ 4\ 2\ 0)\ \ (-\ 4\ 2\ 0)\ \ (+\ 2\ 1\ 0).$$

Note that this history contains a cycle.

We now extend SPVP$_1$ to SPVP$_2$, which dynamically computes such histories in a distributed manner. A message m is a pair $(P,\ h)$ where P is a path and h is a history. The intent is that h describes a sequence of path change events that allowed P to be adopted. For any message $m = (P,\ h)$, define $\text{path}(m) = P$ and $\text{history}(m) = h$. The set of choices at node u, $\text{choices}(u)$, is now redefined to be

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

Similarly, the path assignment associated with a network state is now $\pi(u) = \text{path}(\text{rib}(u))$.

process $\text{spvp}_2[u]$
begin
    receive m **from** w $\longrightarrow$
        begin
            $\text{rib-in}(u \Leftarrow w) := m$
            if $\text{path}(\text{rib}(u)) \neq \text{best}(u)$ **then**
            begin
                $P_{\text{old}} := \text{path}(\text{rib}(u))$
                $P_{\text{new}} := \text{best}(u)$
                $h_{\text{new}} := \text{hist}(u)$
                $\text{rib}(u) := (P_{\text{new}},\ h_{\text{new}})$
                for each $v \in \text{peers}(u)$ **do**
                begin
                    send $\text{rib}(u)$ **to** v
                end
            end
        end
end

Fig. 4. The SPVP$_2$ process at node u.

$\text{hist}(u)$	condition
$(+,\ P_{\text{new}})\ h$	if $\lambda^u(P_{\text{old}}) < \lambda^u(P_{\text{new}})$, $P_{\text{new}} = (u\ w)P_2$, and $\text{rib-in}(u \Leftarrow w) = (P_2,\ h)$,
$(-,\ P_{\text{old}})\ h$	if $\lambda^u(P_{\text{old}}) > \lambda^u(P_{\text{new}})$, $P_{\text{old}} = (u\ w)P_1$, $\text{rib-in}(u \Leftarrow w) = (P_2,\ h)$,
$(s,\ Q)\ h$	if $\lambda^u(P_{\text{old}}) = \lambda^u(P_{\text{new}})$, $P_{\text{old}} = (u\ w)P_1$, $P_{\text{new}} = (u\ w)P_2$, $\text{rib-in}(u \Leftarrow w) = (P_2,\ h)$, $h = e\ h'$, $s = \text{sign}(e)$, and $Q = \begin{cases} P_{\text{old}} & \text{if } s = - \\ P_{\text{new}} & \text{if } s = + \end{cases}$

Fig. 5. Definition of the function $\text{hist}(u)$.

Figure 4 presents the process $\text{spvp}_2[u]$ that implements SPVP$_2$ at node u. This process uses the auxiliary function $\text{hist}(u)$ to compute a new path history for P_{new} whenever the best path at node u changes from P_{old} to P_{new}. This function is defined in Figure 5. If u goes up to $P_{\text{new}} = (u\ w)P_2$, then the history of this path at u is $(+,\ P_{\text{new}})\ h$, where h is the history of P_2 received from w explaining why w adopted P_2. If u goes down from $P_{\text{old}} = (u\ w)P_1$, then the history of

step	u	best(u)	hist(u)
0	1	(1 0)	$\diamond$
	2	(2 0)	$\diamond$
	3	(3 4 2 0)	$\diamond$
	4	(4 2 0)	$\diamond$
1	1	(1 0)	$\diamond$
	2	(2 1 0)	(+ 2 1 0)
	3	(3 4 2 0)	$\diamond$
	4	(4 2 0)	$\diamond$
2	1	(1 0)	$\diamond$
	2	(2 1 0)	(+ 2 1 0)
	3	(3 4 2 0)	$\diamond$
	4	ϵ	(− 4 2 0) (+ 2 1 0)
3	1	(1 0)	$\diamond$
	2	(2 1 0)	(+ 2 1 0)
	3	(3 0)	(− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	ϵ	(− 4 2 0) (+ 2 1 0)
4	1	(1 0)	$\diamond$
	2	(2 1 0)	(+ 2 1 0)
	3	(3 0)	(− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	(4 3 0)	(+ 4 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
5	1	(1 3 0)	(+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	2	(2 1 0)	(+ 2 1 0)
	3	(3 0)	(− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	(4 3 0)	(+ 4 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
6	1	(1 3 0)	(+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	2	(2 0)	(− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	3	(3 0)	(− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	(4 3 0)	(+ 4 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
7	1	(1 3 0)	(+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	2	(2 0)	(− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	3	(3 0)	(− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	(4 2 0)	(+ 4 2 0) (− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
8	1	(1 3 0)	(+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	2	(2 0)	(− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	3	(3 4 2 0)	(+ 3 4 2 0) (+ 4 2 0) (− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	(4 2 0)	(+ 4 2 0) (− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
9	1	(1 0)	(− 1 3 0) (+ 3 4 2 0) (+ 4 2 0) (− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	2	(2 0)	(− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	3	(3 4 2 0)	(+ 3 4 2 0) (+ 4 2 0) (− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)
	4	(4 2 0)	(+ 4 2 0) (− 2 1 0) (+ 1 3 0) (− 3 4 2 0) (− 4 2 0) (+ 2 1 0)

Fig. 6. Trace for system BAD GADGET.

the new path at u is $(-,\ P_{\text{old}})\ h$, where h is the history of the most recent path from w explaining why w abandoned P_1. Finally, if u changes paths to one of equal rank, then both paths must be from the same peer w. (This follows from the *strictness* condition imposed on ranking functions in Section II.) Node u generates a path change event (s, Q), where the sign s is taken from the most recent message received from w. The path Q is chosen to be consistent with this sign in the following sense. If $s = -$, then w went down, and u will agree with w and generate the event $(-,\ P_{\text{old}})$. If $s = +$, then w went up, and again u will agree with w and generate the event $(+,\ P_{\text{new}})$.

Figure 6 presents an dynamic trace for SPVP_2 that replays the sequence presented in Figure 3. At step 6, node 2 has exactly the history informally described above. For another example, at step 4, we have $\text{rib}(4) = ((4\ 3\ 0),\ h)$, where $h = (+\ 4\ 3\ 0)\ (-\ 3\ 4\ 2\ 0)\ (-\ 4\ 2\ 0)\ (+\ 2\ 1\ 0)$. We can the read this history as *node* 4 *went up to* (4 3 0) *because node* 3 *went down from* (3 4 2 0) *because node* 4 *went down from* (4 2 0) *because node* 2 *went up to* (2 1 0).

At step 9 every node contains a cycle in its history. Since BAD GADGET has no solution, these histories would grow in an unbounded manner if the evaluation continued. Note that without the histories, each node would see an oscillation of its path to the origin, but it would not be able to identify this as a policy-based oscillation.

C. SPVP_3 : a safe path vector protocol

Using SPVP_2, the presence of cycles in path histories alerts nodes to the fact that a policy-based oscillation has been dynamically realized. SPVP_3 extends SPVP_2 to a safe protocol by using this information to suppress selected paths, effectively eliminating them from the set of permitted paths. This process of elimination can continue until no protocol oscillation is possible. The resulting solution is then not a solution to the original Stable Paths Problem, but rather to a subproblem.

There are several approaches possible, and we present only one here. We add to each node an additional data structure, $\mathrm{B}(u)$, that contains a set of *bad paths*. We modify the definition of the auxiliary function $\text{best}(u)$ so that it never considers paths in the set $\mathrm{B}(u)$. Define $\text{choices}_{\mathrm{B}}(u)$ to be

$$\{(u\ w)P \in (\mathcal{P}^u - \mathrm{B}(u))\ \mid P \in \text{rib-in}(u \Leftarrow w)\}$$

and

$$\text{best}_{\mathrm{B}}(u) = \max(u, \text{choices}_{\mathrm{B}}(u)).$$

```
process spvp_3[u]
begin
    receive m from w  -->
        begin
            rib-in(u <= w) := m
            if path(rib(u)) ≠ best_B(u) then
            begin
                P_old := path(rib(u))
                P_new := best_B(u)
                h_new := hist(u)
                if h_new contains a cycle then
                begin
                    B(u) := B(u) ∪ {P_new}
                    P_new := best_B(u)
                    if P_new ≠ P_old then
                    begin
                        h_new := (−, P_old)
                    end
                end
                if P_new ≠ P_old then
                begin
                    rib(u) := (P_new, h_new)
                    for each v ∈ peers(u) do
                    begin
                        send rib(u) to v
                    end
                end
            end
        end
end
```

Fig. 7. The SPVP_3 process at node u.

We then modify SPVP_2 so that if u would construct a history containing a cycle for path $P_{\mathrm{new}} = \mathrm{best}_{\mathrm{B}}(u)$, then it adds path P_{new} to the set $\mathrm{B}(u)$ and recomputes $P_{\mathrm{new}} = \mathrm{best}_{\mathrm{B}}(u)$ with this modified $\mathrm{B}(u)$. If $P_{\mathrm{new}} \neq P_{\mathrm{old}}$, then u must have gone down from P_{old}. In this case the history of the new path is *truncated* to $(-,\ P_{\mathrm{old}})$ to ensure that the resulting history does not contain a cycle. Figure 7 presents the process SPVP_3.

For the evaluation of BAD GADGET presented in Figure 6, this results in convergence to the path assignment

$$((1\ 3\ 0),\ \epsilon,\ (3\ 0),\ (4\ 3\ 0)).$$

In the transition from step 5 to step 6 node 2 first constructs a cycle in the history of path $(2\ 0)$,

$$(-\ 2\ 1\ 0)\ \ (+\ 1\ 3\ 0)\ \ (-\ 3\ 4\ 2\ 0)\ \ (-\ 4\ 2\ 0)\ \ (+\ 2\ 1\ 0).$$

Because of this, it adds path $(2\ 0)$ to its set of bad paths $\mathrm{B}(2)$. It then adopts the empty path, which results in a stable state. This is exactly the solution to the Stable Paths Problem obtained from BAD GADGET by deleting path $(2\ 0)$ from the set of permitted paths.

IV. Analysis

In this section we prove the Soundness and Completeness theorems. We first review the notion of a *dispute digraph* first introduced in [4]. Given an instance of the Stable Paths Problem, $S = (G,\ \mathcal{P},\ \Lambda)$, the dispute digraph $\mathcal{DD}(S)$ is a graph whose nodes are labeled with the permitted paths of S and whose arcs represent certain relationships between the policies of peers. Cycles in this digraph represent circular dependencies that cannot be simultaneously satisfied. In [4] is is shown that if $\mathcal{DD}(S)$ is acyclic, then S has a unique solution. We show that path histories correspond exactly to trails in the dispute digraph and that history cycles correspond to cycles in the dispute digraph. Thus, the dispute digraph allows us to formalize the notions of "policy conflicts" and "policy-based protocol oscillation" and link them to dynamically computed histories. See [6] for full proofs.

A. Dispute Digraph

For any instance of the Stable Paths Problem, $S = (G,\ \mathcal{P},\ \Lambda)$, 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*. There is a transmission arc $P \ \cdots\!> \ (u\ v)P$ when nodes u and v are peers, P is permitted at v, and $(u\ v)P$ is permitted at u. Informally, $P\ \cdots\!> (u\ v)P$ might be read as "node v permits path P, which allows u to permit path $(u\ v)P$".

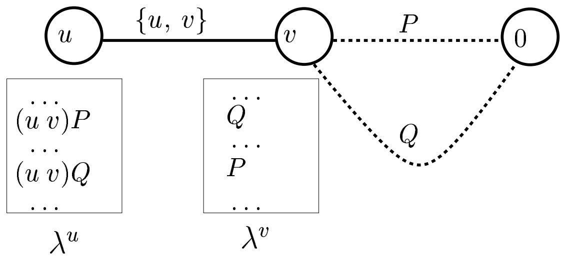

Fig. 8. Conditions for dispute arc $Q\ \ {-\!\!-\!\!>}\ (u\ v)P$.

Suppose that nodes u and v are peers. Let Q and P be two permitted paths at v such that $(u\ v)P$ is a permitted path at u. A *dispute arc* from path Q to path $(u\ v)P$, denoted $Q\ \ {-\!\!-\!\!>}$ $(u\ v)P$, 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 and adopting Q, while this action would force u to abandon path $(u\ v)P$ and select as its best path one that could potentially have lower rank. More formally, $Q\ \ {-\!\!-\!\!>}\ (u\ v)P$ is a dispute arc if and only if the following conditions hold :

1. $(u\ v)P$ is a permitted path at node u,
2. Q and P are permitted paths at node v,
3. path $(u\ v)Q$ is not permitted at node u, or $\lambda^u((u\ v)Q) < \lambda^u((u\ v)P)$,
4. $\lambda^v(P) \leq \lambda^v(Q)$.

These conditions are illustrated in Figure 8. Figure 9 shows the dispute digraphs for the specifications of Figure 1. Again, the dotted arcs are transmission arcs, while the solid arcs are dispute arcs.

A *trail* in $\mathcal{DD}(S)$ is of the form $P_1\ a_1\ P_2\ a_2\ \cdots P_{n-1}\ a_{n-1}\ P_n$, where $n \geq 1$, $P_i \in \mathcal{P}$, and each $P_i\ a_i\ P_{i+1}$ represents a dispute arc or a transmission arc. Such a trail contains a cycle if $P_i = P_j$

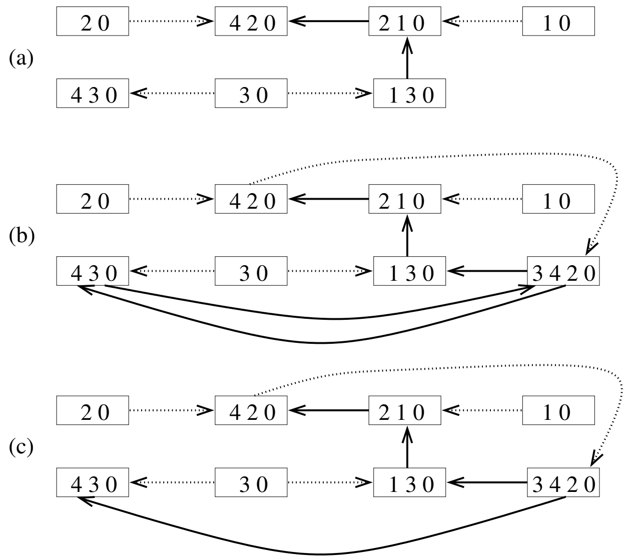

Fig. 9. Dispute digraphs for GOOD (a), NAUGHTY (b), and BAD (c) GADGETs.

for some $i \neq j$. We usually refer to cycles in the dispute digraph as *dispute cycles*. The following fact is proved in [4].

Fact 1: Let S be an instance of the Stable Paths Problem. If $\mathcal{DD}(S)$ is acyclic, then S has a unique solution.

From Figure 9 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

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

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

We define the function $T(h)$ that translates a non-empty history h into a trail in the dispute digraph.

$$\begin{aligned} T((s,\ P)) &= P \\ T((s_1,\ P_1)\,(s_2,\ P_2)\,h) &= T((s_2,\ P_2)\,h)\ \text{arc}(s_1, s_2)\ P_1, \end{aligned}$$

where

$$\text{arc}(s_1, s_2) \;=\; \begin{cases} \cdots\!> & (\text{if } s_1 = s_2) \\ \longrightarrow & (\text{otherwise}). \end{cases}$$

Whereas a history h records a dynamic sequence of path change events, the trail $T(h)$ can be interpreted as a static chain of policy relationships that allowed this event sequence to be realized.

It is easy to check that if h is the history

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

then $T(h)$ is the trail

$$(2\,1\,0) \;\longrightarrow (4\,2\,0) \;\cdots\!> (3\,4\,2\,0) \;\longrightarrow (1\,3\,0) \;\longrightarrow (2\,1\,0)$$

in the dispute digraphs of both NAUGHTY GADGET and BAD GADGET. Note that not all histories translate to valid trails in the dispute digraph. However, the Soundness Theorem proved below shows that this is the case for histories constructed by SPVP_2.

B. Network States

We model (logical) time t with discrete values $0, 1, 2, \ldots$. For each 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 $(P_w,\ \diamond)$, each rib-in$(u \Leftarrow w)$ = $(\epsilon,\ \diamond)$, and each rib$(u) = (P_u,\ \diamond)$.

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. Suppose that we are using protocol SPVP_k, $k \in \{1, 2, 3\}$. 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 $\text{SPVP}_k(S, \sigma, s_0, t)\!\downarrow$, 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 $\text{SPVP}_k(S, \sigma, s_0)\!\Uparrow$.

We now define the notion of a consistent network state. The Soundness Theorem will show that consistency is preserved under state transitions.

The state at time t is *rib consistent* if for all u, $\text{path}(\text{rib}(u, t))$ is the best path possible, given the values of rib-in$(u \Leftarrow w, t)$, for $w \in$ peers(u).

If $m = (P, h)$ is a message, with $P \in \mathcal{P}^u$, then m is *consistent* if and only if the following conditions hold :
1. if $h \neq \diamond$, then $T(h)$ is a trail in $\mathcal{DD}(S)$,
2. if $m = (P,\ (+,\ P')\ h')$, then $P = P'$,
3. if $m = (P,\ (-,\ P')\ h')$, then $P \neq P'$ and $\lambda^u(P) \leq \lambda^u(P')$.

A state $s(t)$ is message consistent if all messages in it are consistent.

We say that pipe$(u \Leftarrow w, t)$ is *pipe consistent* if all messages in it are consistent and if the following conditions hold :
1. if $i > 1$ and pipe$(u \Leftarrow w, t)[i] = (P_1,\ (-,\ P_2)\ h)$, then pipe$(u \Leftarrow w, t)[i-1] = (P_2,\ h')$ or pipe$(u \Leftarrow w, t)[i-1] = (\epsilon,\ \diamond)$,
2. if $i > 1$, pipe$(u \Leftarrow w, t)[i-1] = (P_1,\ h')$, and pipe$(u \Leftarrow w, t)[i] = (P_2,\ (+,\ P_2)\ h)$, then $\lambda^w(P_1) \leq \lambda^w(P_2)$,
3. if $i > 1$ and pipe$(u \Leftarrow w, t)[i] = (P,\ h)$, then $h \neq \diamond$,
4. pipe$(u \Leftarrow w, t)[k] =$ rib(w, t), where k is the number of messages in pipe$(u \Leftarrow w, t)$.

Note that clause (4) 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 message consistent, rib consistent, and all pipes are pipe-consistent

C. Soundness of SPVP$_2$

Theorem 2 (Soundness Theorem) Let σ be an activation sequence. Suppose that $s(t)$ is a consistent state and $s(t) \xrightarrow{\sigma(t)} s(t+1)$, using SPVP$_2$. Then $s(t+1)$ is a consistent state.

Proof: Note that $s(t)$ is message consistent, rib consistent, and all pipes are pipe consistent. It is not difficult to see that the only way that $s(t+1)$ could violate these conditions is if some node u receives a message $m_w = (P^w_{\text{new}},\ h^w)$ from some peer w and then constructs a message that is not message consistent, is not rib-consistent, or violates pipe-consistency. Suppose $m_{\text{old}} = \text{rib}(u, t)$ and $m_{\text{new}} = \text{rib}(u, t+1)$. Let $P_{\text{old}} = \text{path}(m_{\text{old}})$, $P_{\text{new}} = \text{path}(m_{\text{new}})$, and $h_{\text{new}} = \text{history}(m_{\text{new}})$

By inspection of spvp$_2[u]$ in Figure 4, we see that if $\text{path}(\text{rib}(u)) = \text{best}(u)$, then there is no change to the network state other than the fact that the pipe pipe$(w \Leftarrow w, t+1)$ is now shorter by one message. In this case it is easy to see that consistency of state $s(t)$ implies consistency of state $s(t+1)$.

Otherwise, u will change its path. For SPVP$_2$, this means that $\text{path}(\text{rib}(u)) \neq \text{best}(u)$ and u recomputes its path as

$$P_{\text{new}} := \text{best}(u).$$

This means that rib-consistency is guaranteed. The protocol then creates the message $m_{\text{new}} = \text{rib}(u, t+1)$. This is defined with the assignment

$$\text{rib}(u) := (P_{\text{new}},\ \text{hist}(u)),$$

and then m_{new} is sent to all peers of u. Therefore, we need to show that m_{new} is message consistent, and that for each $v \in$ peers(u), pipe$(v \Leftarrow u, t+1)$ is pipe-consistent. There are three cases to consider.

Case 1: $\lambda^u(P_{\text{old}}) < \lambda^u(P_{\text{new}})$. Since $s(t)$ is rib-consistent and u went up to P_{new} we know that $P_{\text{new}} = (u\ w)P^w_{\text{new}}$ and that $h_{\text{new}} = (+,\ P_{\text{new}})\ h^w$. Given that $s(t)$ is pipe-consistent it is easy to see that adding m_{new} to any pipe pipe$(v \Leftarrow u, t)$ will satisfy clauses (2), (3), and (4) in the definition of pipe consistency.

To show that m_{new} is message consist, it is enough to show that $T(h_{\text{new}})$ is a is a trail in $\mathcal{DD}(S)$. There are three cases to consider.

Case 1.1 : $h^w = \diamond$. Then $T(h_{\text{new}}) = P_{\text{new}}$, which is a label for a single node in $\mathcal{DD}(S)$, and so is a (trivial) trail.

Case 1.2 : $h^w = (+,\ P^w_{\text{new}})\ h'$. Then

$$\begin{aligned} T(h_{\text{new}}) &= T(h^w)\ \text{arc}(+,+)\ P_{\text{new}} \\ &= T(h^w)\ \cdots\!>\ P_{\text{new}}. \end{aligned}$$

By consistency of $s(t)$ we know that $T(h^w)$ is a trail in the dispute digraph ending in path P^w_{new}, and since $P^w_{\text{new}} \cdots\!> (u\ w)P^w_{\text{new}}$ is a transmission arc, we conclude that $T(h_{\text{new}})$ is a trail in the dispute digraph.

Case 1.3 : $h_3 = (-,\ P^w_{\text{old}})\ h'$. Then

$$\begin{aligned} T(h_{\text{new}}) &= T(h^w)\ \text{arc}(+,-)\ P_{\text{new}} \\ &= T(h^w)\ \longrightarrow\ P_{\text{new}}. \end{aligned}$$

By pipe consistency of $s(t)$, we know that $P^w_{\text{old}} = \text{path}(\text{rib-in}(u \Leftarrow w, t))$. Since u went up to P_{new} we know that either $(u\ w)P^w_{\text{old}}$ is not permitted at u or $\lambda^u((u\ w)P^w_{\text{old}}) < \lambda^u((u\ w)P^w_{\text{new}})$. By the consistency of message m_w, we know that $\lambda^w(P^w_{\text{new}}) \leq \lambda(P^w_{\text{old}})$. Therefore, $P^w_{\text{old}} \dashrightarrow P_{\text{new}}$ is a dispute arc in the dispute digraph. By consistency of $s(t)$ we know that $T(h^w)$ is a trail in the dispute digraph ending in path P^w_{old}, and so we conclude that $T(h_{\text{new}})$ is a trail in the dispute digraph.

Case 2 ($\lambda^u(P_{\text{old}}) > \lambda^u(P_{\text{new}})$) and Case 3 ($\lambda^u(P_{\text{old}}) = \lambda^u(P_{\text{new}})$) proceed in a manner similar to Case 1. See [6] for complete details. ■

Theorem 3 (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 SPVP$_2(S, \sigma, s_0, t)\!\downarrow$. Let $\vec{P} = (P_1, \cdots, P_n)$ where $\text{path}(\text{rib}(i, t)) = P_i$. Then $\vec{P}$ is a solution for the specification S. In addition, the protocol SPVP$_1$ converges, to this same solution.

D. Completeness of SPVP$_2$

Suppose s_0 is a consistent state, σ is a fair activation sequence with respect to s_0, and that SPVP$_2(S, \sigma, s_0)\!\uparrow$. 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 $t > t_f$.

Suppose at time t we have $\mathrm{rib}(u,t) = (P,\ h)$ where $h = \ e\, h'$ and $h' \neq \diamond$. By construction, it can be seen that the history h' is always taken from a message from some $w \in \mathrm{peers}(u)$. In this case we say that u's history h *extends* the history h' from node w. Clearly, $|h| = |h'| + 1 > |h'|$.

Lemma 4: Suppose $t > t_f + 1$, $w \in \mathcal{C}$, u a peer of w, and u changes its best path at time t. Then $\mathrm{history}(\mathrm{rib}(u,t))$ cannot extend a history of w.

For any time t, define $M(t)$ to be the set of all messages m such that for some $u, w \in \mathcal{O}$, $m \in \mathrm{pipe}(u \Leftarrow w, t)$. Then let $\mathrm{minl}(t)$ be the length of the shortest history of any message in $M(t)$. For any t, define $\mathrm{shortest}(t)$ to be the set of all messages m in $M(t)$ such that $|\mathrm{history}(m)| = \mathrm{minl}(t)$.

Lemma 5: For all $t > t_f$, $\mathrm{minl}(t) \leq \mathrm{minl}(t+1)$.

Lemma 6: For all $t_1 > t_f$, there exists a $t_2 > t_1$ such that $\mathrm{minl}(t_2) > \mathrm{minl}(t_1)$.

Theorem 7 (Completeness Theorem) Let s_0 be a consistent state and σ an activation sequence fair with respect to s_0. Suppose that $\mathrm{SPVP}_2(S, \sigma, s_0)\Uparrow$. Then there is a time t' such that for every oscillating node $u \in \mathcal{O}$ and every time $t > t'$, the history of $\mathrm{rib}(u,t)$ contains an event cycle.

Proof: Since $\mathrm{SPVP}_2(S, \sigma, s_0)\uparrow$ we know that $\mathcal{O}$ is not empty. Let u be any node in $\mathcal{O}$. By Lemma 6 we know that the history of $\mathrm{rib}(u)$ grows in an unbounded manner. By Theorem 2 (Soundness), we know that each history h in the system corresponds exactly to a trail $T(h)$ in the dispute digraph. Therefore, since there are only finitely many nodes in the dispute digraph, there is some time t_u such that for all $t > t_u$, the history $\mathrm{history}(\mathrm{rib}(u,t))$ contains at least one cycle. Let t' be the maximum value over all such t_u. ■

This implies that if a specification has an acyclic dispute digraph (as with GOOD GADGET), then SPVP will never diverge.

E. $SPVP_3$ is Safe

Let $\mathrm{B}(u,t)$ denote the set $\mathrm{B}(u)$ at time t. The network state, $s(t)$, now includes all sets $\mathrm{B}(u,t)$. We assume that these sets are initially empty.

Theorem 8 (Safety) Let S be any instance of the Stable Paths Problem. Let s_0 be a consistent state and σ an activation sequence fair with respect to s_0. Then there exists a time t such that $\mathrm{SPVP}_3(S, \sigma, s_0, t)\Downarrow$. Furthermore, the path assignment corresponding to network state $s(t)$ is a solution to a subproblem of S. This subproblem S' is defined by replacing each $\mathcal{P}^u$ with $\mathcal{P}^u - \mathrm{B}(u,t)$.

Proof (sketch): Note that once u adds path P to $\mathrm{B}(u)$, it is never the case afterwards that $P = P_{\mathrm{new}}$ at u. In particular, once u adds P to $\mathrm{B}(u)$, it will never do so again. Therefore, there is a time t_1 after which no node u adds a path to $\mathrm{B}(u)$.

By fairness of σ, there is a time $t_2 > t_1$ such that no rib or communications link contains a message of the form $(QP,\ h)$, where P is a path in $\mathrm{B}(u)$ for some node u. In other words, at time t_2 all suppressed paths have been flushed from the system in such a way that they cannot be involved in the construction of new best paths.

We can transform the state $s(t_2)$ into a state s'_0 that is consistent for S' by erasing all but the most recent events from all histories. (This is to guarantee that we start in a state in which all histories correspond to trails in the dispute digraph for the new Stable Paths Problem S'.) Let $\sigma'(t) = \sigma(t + t_2)$. We can evaluate SPVP_2 for the Stable Paths Problem S' with the initial state s'_0 using σ' as the activation sequence. Note that the evaluation of SPVP_3 (using σ) after t_2 parallels step-by-step the evaluation of SPVP_2 (using σ') after time 0.

If SPVP_3 diverges, then so does SPVP_2. However, by Theorem 7 we know that some node u will construct a path whose history contains a cycle. The same would happen for SPVP_3, and so some new path would be added to some $\mathrm{B}(u)$, which is a contradiction. ■

V. Towards a safe BGP

In this section we model a collection of BGP routing policies as an instance of the Stable Paths Problem. We then outline how SPVP_3 can be used to design an extension to BGP to make it safe. We do not intend to give a complete description of a safe BGP here. Rather, we mean only to sketch the basic ideas. Given the complexity of BGP, a complete treatment would be quite involved. We make several simplifying assumptions. First, we ignore address aggregation. Second, we ignore all issues related to internal BGP (iBGP). In effect, this means that we pretend as if each autonomous system is implemented with one BGP speaking router. In this case, the graph G of the corresponding Stable Paths Problem is exactly a graph of autonomous systems, and paths correspond to AS-paths. If one considers iBGP, then this picture would become more complicated. The nodes of G would then correspond to border routers, and multiple paths in G could correspond to the same AS-path.

First, we briefly review the route selection process of BGP [12], [7], [13]. BGP employs a number of attributes to convey information about paths to a given destination. For example, one BGP attribute, **as_path**, 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 *routing policies*.

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

nlri	:	network layer reachability information (address block for a set of destinations)
next_hop	:	next hop (address of next hop router)
as_path	:	ordered list of vertices traversed
local_pref	:	local preference
med	:	multi-exit discriminator
c_set	:	set of community tags

The local preference attribute **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

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

For a given destination d, the records r with $d = r.\mathbf{nlri}$ are ranked using lexical ordering on rank-tuple(r). This induces an integer-valued function, lex-rank(r), such that if r_1 lexically precedes r_2, then lex-rank(r_1) $<$ lex-rank(r_2). The best route selection procedure for BGP [12] picks routes with the highest rank. In other words, if two route records share the same

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 **as_path** is preferred. If these paths have the same length, then the record with the lowest **med** value is preferred. Finally, ties are broken with preference given to the record with the lowest IP address for its **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 **nlri** and a given **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, $r_1 = \text{export}(w \to u, r)$ 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 **as_path** of r_1, filters out the record if its **as_path** contains u, and deletes some attribute values such as **local_pref**. 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 **local_pref** value for r_3. We call the composition of these transformations the *peering transformation*, $\text{pt}(u \Leftarrow w, r)$, defined as

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

Suppose 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.

Given the graph G, whose nodes correspond to autonomous systems and whose edges represent BGP peering relationships, we can define the corresponding Stable Paths Problem as follows. 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 the *ranking function*, $\lambda^{u_k}(P)$, on paths permitted at u_k as $\lambda^{u_k}(P) = \text{lex-rank}(r(P))$. In terms of BGP, this translates 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.

We need to lift the notion of path history from SPVP_3 up to a notion of *route history* that can be used at the level of BGP. There are any number of ways to do this. For example, one choice is to add a new optional attribute, **history**, to route records. If r is a record, then $r.\mathbf{history}$ is a sequence of *route change events*. A route change event is a pair $(s,\ r')$, where $s \in \{+,\ -\}$ and r' is a record. Note that this attribute cannot be changed by import and export rules and it cannot be included in the records of a history. In addition, the *withdraw* message of BGP must be extended with an (optional) history attribute.

One problem with this design is that it exports attributes (such as **local_pref**) that should be private to an autonomous system. One solution is to replace route records in histories with "record designators" of the form (w, n), where w is the autonomous system creating this node in the history and n is a "magic number" that w has generated to identify the route record involved. This requires each autonomous system w to maintain a mapping $\text{r-id}(w, r) = n$. Furthermore, this mapping must have the property that $\text{r-id}(w, r_1) = \text{r-id}(w, r_2)$ if and only if $r_1 = r_2$.

VI. Open Problems and Future Work

The main challenge is to minimize the impact of a history-based extension on BGP. Ideally, we would like to allow an implementation of BGP to operate exactly as it does now if nothing "strange" happens. One way to do this might be with *triggered histories*, where the construction of histories is only initiated when a node experiences a conflict. In other words, with triggered histories the dispute trails would always begin with a dispute arc and never a transmission arc.

As with route flap dampening, history-based route suppression should be subject to *policy controls*. Although the detection of dispute cycles in a history does indicate that a policy based oscillation has occurred, it may be that such an oscillation is *transient* in nature (as is possible with NAUGHTY GADGET, for example). In practice it may be more useful to suppress routes after they are seen to contain some number of dispute cycles, or after the length of the history has exceeded some limit.

Finally, we need to address the manner in which policy-based oscillations are debugged. The events contained in histories record policy disputes at a very low level. Since most routing policies are expressed at a higher level, tools are needed to map the low level disputes up to the level of routing policies.

References

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

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

[3] 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.

[4] T. Griffin, F.B. Shepherd, and G. Wilfong. Policy disputes in path-vector protocols. In *ICNP'99*, 1999.

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

[6] T. Griffin and G. Wilfong. A safe path vector protocol. Bell Labs Technical Memorandum, 1999.

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

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

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

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

[11] John T. Moy. *OSPF : Anatomy of an internet routing protocol*. Addison-Wesley, 1998.

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

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

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

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