Local Transit Policies and the Complexity of BGP Stability Testing

Marco Chiesa Luca Cittadini Giuseppe Di Battista Stefano Vissicchio
Dept. of Computer Science and Automation, Roma Tre University
{chiesa,ratm,gdb,vissicch}@dia.uniroma3.it

***Abstract*—BGP, the core protocol of the Internet backbone, is renowned to be prone to oscillations. Despite prior work shed some light on BGP stability, many problems remain open. For example, determining how hard it is to check that a BGP network is safe, i.e., it is guaranteed to converge, has been an elusive research goal up to now.**

In this paper, we address several problems related to BGP stability, stating the computational complexity of testing if a given configuration is safe, is robust, or is safe under filtering. Further, we determine the computational complexity of checking popular sufficient conditions for stability.

We adopt a model that captures Local Transit policies, i.e., policies that are functions only of the ingress and the egress points. The focus on Local Transit policies is motivated by the fact that they represent a configuration paradigm commonly used by network operators. We also address the same BGP stability problems in the widely adopted SPP model.

Unfortunately, we find that the most interesting problems are computationally hard even if policies are restricted to be as expressive as Local Transit policies. Our findings suggest that the computational intractability of BGP stability be an intrinsic property of policy-based path vector routing protocols that allow policies to be specified in complete autonomy.

I. Introduction

The *Border Gateway Protocol (BGP)* [1] is not guaranteed to converge: it can fail to find a *stable routing* either because there does not exist any [2] or because of bad ordering of messages [3].

Since the effects of BGP *oscillations* can range from performance degradation [4] to denial of service [5], BGP *stability* attracted lots of research interest. Necessary [6] and sufficient [7], [3] conditions for stability have been found and changes to the protocol [8] or limitations to the expressiveness of the policies [9], [10] have been proposed. However, changing the protocol faces severe deployment issues. Moreover, enforcing sufficient conditions for stability may be incompatible with the need for expressiveness and autonomy that BGP was designed to address. For these reasons, the problem of checking a given BGP configuration for stability has also been deeply studied. As an example, it has been shown that deciding whether a given BGP configuration admits a stable routing is NP-hard [11], [3].

In this paper we consider several fundamental problems related to BGP stability: (*i*) SAFETY [11], [3], i.e., the problem of verifying that a BGP configuration is guaranteed to converge. (*ii*) SUF [6] and ROBUSTNESS [3], i.e., the problems of verifying that a safe BGP configuration is guaranteed to converge under any filtering action and any link failure, respectively. (*iii*) NO-DW [3] and NO-DR [12] i.e., the problems of verifying that a BGP configuration does not contain a *dispute wheel* and a *dispute reel*, respectively. The absence of a dispute wheel is a sufficient condition for SAFETY while the absence of a dispute reel is a characterization for SUF.

We study the complexity of the above problems within three different models for BGP policies: (*i*) The widely adopted SPP model [3], that captures arbitrarily complex BGP policies. (*ii*) The 3-SPP model, that captures the so-called Local Transit policies [13], a very common configuration paradigm where policies are functions only of the ingress and the egress points. (*iii*) The 2-SPP model, a simplified version of 3-SPP, where either only the ingress point, i.e., the BGP neighbor that announced the route, is considered. In all three cases we adopt the well-known SPVP [3] model for BGP dynamics.

We exploit those models to study how expressive BGP policies can be in order to allow an efficient static assessment of BGP stability, assuming that ASes fully preserve their autonomy. Unfortunately, we find that the most interesting problems are computationally hard even if policies are restricted to be Local Transit only. First, solving a long standing open problem [11], [3], we prove that SAFETY is coNP-hard both in SPP and in 3-SPP. Second, we prove that SUF is coNP-complete in SPP and that ROBUSTNESS is coNP-hard both in SPP and in 3-SPP. Third, we show that even the NO-DW problem, which can be solved efficiently in SPP [7], is coNP-complete in 3-SPP. Also, we find that the NO-DR problem is coNP-complete both in SPP and in 3-SPP. As a side effect, since any 3-SPP configuration can be expressed in the model proposed in [11] without changing the size of the input, our negative results can be extended to the model in [11].

Stimulated by the above list of negative results, we investigate whether stability problems can be made tractable by sacrificing the expressive power of policy configurations, while preserving ASes' autonomy. Eventually, we find that SAFETY is solvable in polynomial time in 2-SPP, where policies are so restricted that they are unsuitable for practical uses.

The rest of the paper is organized as follows. Section II reviews related work in the field of BGP stability. In Section III, we introduce the models we use in this paper. In Section IV and V, we study the complexity of SAFETY and NO-DW respectively, while Section VI studies NO-DR, SUF, and ROBUSTNESS. Finally, we conclude in Section VII.

II. Related Work

In [11] a BGP model is proposed where policies are described by means of functions that implement import and export filters, similarly to real-world BGP configuration languages. Several important complexity results are proved: (*i*) checking if a BGP network has a stable routing (SOLVABILITY) is NP-complete, (*ii*) deciding whether a BGP network can be trapped in a permanent oscillation is NP-hard, and (*iii*) deciding whether a BGP network has a stable routing, i.e., it is solvable, under any combination of link failure is NP-hard. The complexity of SAFETY is left open.

In [3] the SPP model is introduced. BGP policies are expressed by explicitly enumerating and ranking all the permitted paths. In this setting, it is shown that SOLVABILITY is NP-hard. This result could not be evinced from [11], as translation from one model to the other might take exponential time. The complexity of SAFETY and ROBUSTNESS is left open.

In [14] it is proved that the coexistence of two stable states implies the existence of an oscillation. Policies are modeled with SPP. Although the model for BGP dynamics is slightly different from SPVP, the result also holds in SPVP [14].

In [10] a model is used in which BGP policies are applied consistently network-wide based on a classification of neighbors into groups. In this setting, a polynomial time algorithm is given to check whether the structure of the classes can lead to specific BGP policies in which oscillations are possible. The 3-SPP model is similar to the one used in [10] in that it also limits the expressiveness of BGP policies, however it is more general since it allows each AS to preserve its autonomy.

In [15] SAFETY is claimed to be PSPACE-complete. However, the adopted model assumes that ASes are omniscient, that is, upon activation they can immediately know the AS-paths that are being used by every other AS, without the need to exchange BGP messages. This assumption makes it very hard to apply the results to any realistic model of BGP.

III. BGP Models

This section describes the well-known *Stable Paths Problem* (SPP) formalism [3] and defines k-SPP, a variation of SPP which is suitable to study how policy expressiveness impacts the computational complexity of stability problems.

A. SPP

SPP models a BGP network as an undirected graph $G = (V, E)$, where vertices $V = \{0, 1, \ldots, n\}$ represent ASes and edges in E correspond to BGP peerings between ASes. Vertex 0 is a special vertex in that it is the destination every other vertex attempts to establish a path to. Since different destinations are independently handled by BGP [1], 0 is assumed to be the only destination, without loss of generality. A path P is a sequence of $k+1$ vertices $P = (v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$, $v_i \in V$, such that $(v_i, v_{i-1}) \in E$ for $i = 1, \ldots, k$. Vertex v_{k-1} is the *next hop* of v_k in P. For $k = 0$ we obtain the trivial path (v_0) consisting of vertex v_0 alone. The empty path represents inability to reach the destination and is denoted by ϵ. The *concatenation* of two nonempty paths $P = (v_k\ v_{k-1}\ \ldots\ v_i)$, $k \geq i$, and $Q = (v_i\ v_{i-1}\ \ldots\ v_0)$, $i \geq 0$, denoted as PQ, is the path $(v_k\ v_{k-1}\ \ldots\ v_i\ v_{i-1}\ \ldots\ v_0)$. We assume that $P\epsilon = \epsilon P = \epsilon$, that is, the empty path can never extend or be extended by other paths.

SPP models BGP import/export policies and the BGP decision process by explicitly listing and ranking all permitted paths. More precisely, each vertex $u \in V$ is assigned a set of *permitted paths* $\mathcal{P}^u$ which represent the paths that u can use to reach 0. All the paths in $\mathcal{P}^u$ are simple (i.e., without repeated vertices), start from u and end in 0. The empty path, representing unreachability of 0, is permitted at each vertex $u \neq 0$. Vertex 0 can reach itself only directly, hence $\mathcal{P}^0 = \{(0)\}$. Let $\mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u$. For each $u \in V$, a *ranking function* $\lambda^u : \mathcal{P}^u \rightarrow \mathbb{N}$ determines the relative level of preference $\lambda^u(P)$ assigned by u to path P. If $P_1, P_2 \in \mathcal{P}^u$ and $\lambda^u(P_2) < \lambda^u(P_1)$, then P_2 is *preferred* over P_1. Let $\Lambda = \{\lambda^u | u \in V\}$.

The following conditions hold on permitted paths of each vertex $u \in V - \{0\}$:

(*i*) $\forall P \in \mathcal{P}^u, P \neq \epsilon$: $\lambda^u(P) < \lambda^u(\epsilon)$ (unreachability of 0 is the last resort);

(*ii*) $\forall P_1, P_2 \in \mathcal{P}^u, P_1 \neq P_2 : \lambda^u(P_1) = \lambda^u(P_2) \Rightarrow P_1 = (u\ v)P_1', P_2 = (u\ v)P_2'$, (strict ranking is assumed on all paths but those with the same next hop).

An instance S of SPP is a triple $(G, \mathcal{P}, \Lambda)$.

A *path assignment* is a function π that maps each vertex $v \in V$ to a path $\pi(v) \in \mathcal{P}^v$, representing the fact that the BGP process running at vertex v is selecting $\pi(v)$ as its preferred path to reach the destination. We always have $\pi(0) = (0)$.

BGP dynamics are modeled by a distributed algorithm called *Simple Path Vector Protocol* (SPVP) [3], which computes a path assignment π_t at each iteration t. Since we consider discrete time, iterations and time are interchangeable concepts. SPVP works as follows (details can be found in [3]). Vertex 0 keeps announcing its presence to its neighbors. Every other vertex u collects announcements from its neighbors, discards those announcements containing paths that are not in $\mathcal{P}^u$, and stores non-discarded announcements in a data structure called rib_in$_t$. In particular, rib_in$_t(u \Leftarrow v)$ contains the latest accepted announcement from neighbor v. Thus, u can select a path in the following set:

$$\text{choices}_t(u) = \begin{cases} \{(u\ v)\ \text{rib\_in}_t(u \Leftarrow v)\} & \text{if } u \neq 0 \\ \{(0)\} & \text{if } u = 0 \end{cases}$$

Let W be the set of paths accepted by u from its neighbors. At this point, u selects the best ranked path in W according to its ranking function λ^u:

$$\text{best}(W, u) = \begin{cases} \arg\min_{P \in W} \lambda^u(P) & \text{if } W \neq \oslash \\ \epsilon & \text{if } W = \oslash \end{cases}$$

If this operation updated u's selected path, then u sends announcements to all its neighbors. Notice that, at any time t, SPVP computes a path assignment π_t such that each vertex selects the best available path.

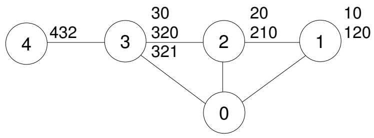

Fig. 1. An instance of the 3-SPP model.

Given an SPP instance, we say that π_t is a *stable path assignment* if, $\forall u \in V$: $\pi_t(u) = \text{best}(\text{choices}_t(u), u)$, that is, every vertex has settled to the best possible choice and cannot switch to a better ranked alternative. The order in which announcements are exchanged among vertices is modeled in SPVP by *activation sequences* [3]. When edge (u, v) is *activated*, vertex v receives u's best path, stores it in $\text{rib\_in}_t(v \Leftarrow u)$, and recomputes its best path. Simultaneous activations are allowed. An activation sequence is *fair* if any edge (u, v) is eventually activated after u has sent a message to v. In the following, we consider only fair activation sequences.

As shown in [16], simplifying the definition of activation sequences (e.g. by activating vertices instead of edges or by not allowing simultaneous activations) leads to inability to model a subset of the possible routing oscillations. It has been shown [7] that, possibly depending on the specific activation sequence, the SPVP algorithm might oscillate indefinitely, never converging to a stable state. An SPP instance S is *safe* [7] if SPVP is guaranteed to eventually reach a stable path assignment on S for any fair activation sequence.

B. 3-*SPP and* k-*SPP*

SPP can model every possible BGP policy specification. However, since it requires explicit listing and ranking of all paths, it is mostly a theoretical model. In fact, network operators configure BGP policies without knowing the entire network topology. Also, the *size* of an SPP instance is bound to the size of $\mathcal{P}$, which can be exponential in $|V|$.

Several models have been proposed in alternative to SPP, either adopting a more realistic specification of policies (e.g., [11]) or limiting the expressiveness of the policies that can be expressed in the model (e.g., [10]). However, these models either have the same expressive power of SPP or have important limitations (see Section II).

We now describe 3-SPP, a variant of SPP in which vertices are forced to rank and filter announcements on the basis of the first 3 hops in the path. This model allows ASes to specify Local Transit policies [13], that is, policies in which ASes define filters based on neighbor pairs (e.g., paths received from neighbor x should not be exported to neighbor y).

We now formally define the 3-SPP model. Let $G = (V, E)$ be defined as for standard SPP instances. Each vertex $u \in V$, with $u \neq 0$, is assigned a set of *permitted path fragments* $\tilde{\mathcal{P}}^u$ such that $(u\ 0)$ can be in $\tilde{\mathcal{P}}^u$ if $(u, 0) \in E$ and paths $(u\ v\ w)$ can be in $\tilde{\mathcal{P}}^u$ if u, v, and w are distinct vertices in V and $(u, v), (v, w) \in E$. The only permitted path fragment at vertex 0 is $\tilde{\mathcal{P}}^0 = \{(0)\}$. To reach 0, a vertex $u \in V - \{0\}$ can use any path $P = (u\ v_1\ \dots\ v_n\ 0)$, starting with a fragment in $\tilde{\mathcal{P}}^u$ and obtained by concatenating any permitted fragment at each vertex v_i, with $i = 1, \dots, n$. Path fragments contain exactly 3 vertices except for the case of 0 and of its neighbors, which can reach 0 directly. Let $\tilde{\mathcal{P}} = \bigcup_{u \in V} \tilde{\mathcal{P}}^u$.

Each vertex $u \in V - \{0\}$ ranks path fragments in $\tilde{\mathcal{P}}^u$ according to a function $\tilde{\lambda}^u : \tilde{\mathcal{P}}^u \to \mathbb{N}$ which assigns a level of preference to paths starting with a fragment in $\tilde{\mathcal{P}}^u$. Namely, if $\tilde{\lambda}^u((u\ v\ w)) < \tilde{\lambda}^u((u\ x\ y))$ then any path starting with $(u\ v\ w)$ is preferred to any path starting with $(u\ x\ y)$.

Similarly to the SPP model, the empty path is always permitted, i.e., $\epsilon \in \tilde{\mathcal{P}}^u$, $\forall u \in V - \{0\}$, and unreachability is the last resort, i.e., $\forall P \in \tilde{\mathcal{P}}^u, P \neq \epsilon$: $\tilde{\lambda}^u(P) < \tilde{\lambda}^u(\epsilon)$.

Differently from the SPP model, two path fragments can have the same rank even if they have a different next hop. Moreover, paths through the same neighbor always have the same rank, i.e., if $(u\ v\ w)$ and $(u\ v\ z)$ are two path fragments in $\tilde{\mathcal{P}}^u$ then $\tilde{\lambda}^u((u\ v\ w)) = \tilde{\lambda}^u((u\ v\ z))$. Any deterministic criterion (e.g., shortest path) can be used to break ties.

An instance $\tilde{S}$ of 3-SPP is a triple $(G, \tilde{\mathcal{P}}, \tilde{\Lambda})$. An example 3-SPP instance is depicted in Fig. 1 using a graphical convention analogous to that used in [3]. The list beside each vertex u represents the permitted path fragments in $\tilde{\mathcal{P}}^u$ sorted by increasing values of $\tilde{\lambda}^u$. For example, vertex 2 can use path fragments in $\tilde{\mathcal{P}}^2 = \{(2\ 1\ 0), (2\ 0)\}$ to reach 0 and prefers $(2\ 0)$. The empty path and $\tilde{\mathcal{P}}^0$ are omitted for brevity. Vertex 3 decides not to propagate the path received from 0 to 2, and permitted paths fragments at vertex 2 result from filtering action performed by 3 and ranking configured at 2. Observe that path fragment 432 at vertex 4 models two distinct paths from 4 to 0, namely 4320 and 43210, that have the same rank.

The 3-SPP model can be generalized to the k-SPP model, where permitted path fragments defined at each vertex contain k hops. The number of path fragments at each vertex is $O(n^{k-1})$, where $n = |V|$, hence the size of an instance of k-SPP is $O(n^k)$. It is easy to verify that, given a specific tie break criterion, an instance of k-SPP can be uniquely translated to an instance of SPP (e.g., by concatenating path fragments to generate permitted paths at each node), while the opposite is in general not true. In other words, k-SPP allows us to trade policy expressiveness for policy succinctness.

IV. THE COMPLEXITY OF THE SAFETY PROBLEM

The SAFETY problem is defined as follows: given an SPP instance, is it safe? In this section we study the computational complexity of SAFETY in the SPP model, in the 3-SPP model, and in the 2-SPP model.

A. *Safety is coNP-Hard in the SPP and in the* 3-*SPP models*

We now prove that SAFETY is coNP-hard in the SPP model using a reduction from SAT COMPLEMENT [17]. In order to prove such a result, we first need to show some technical properties regarding the SPP instance of Fig. 2, which we call TWISTED gadget. TWISTED has vertex set $V = \{0, x, \bar{x}, a, b, c_1, \dots, c_m\}$ and edge set $E = \{(0, a), (0, b), (a, x), (b, \bar{x}), (x, \bar{x})\} \cup \{(c_1, x), (c_1, \bar{x}), \dots, (c_m, x), (c_m, \bar{x})\}$. Policies are as described in Fig. 2. Vertices c_i, with $i = 1, \dots, m$, also have

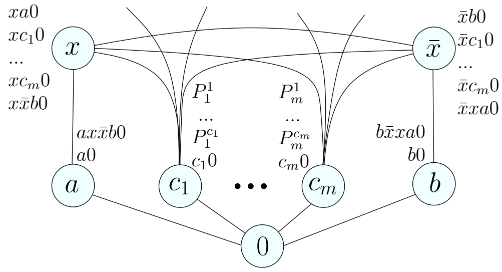

Fig. 2. TWISTED gadget.

links to another portion of the network not explicitly shown in Fig. 2. Each path P_i^j passes through the portion of the network that is not shown and is ranked better than $(c_i\ 0)$.

We now prove two important properties of TWISTED.

Lemma 4.1: For each activation sequence, there do not exist two instants t' and t'' such that $\pi_{t'}(x) = (x\ \bar{x}\ b\ 0)$ and $\pi_{t''}(\bar{x}) = (\bar{x}\ x\ a\ 0)$.

Proof: Suppose, for a contradiction, that there exists an activation sequence such that $\pi_{t'}(x) = (x\,\bar{x}\,b\,0)$ and $\pi_{t''}(\bar{x}) = (\bar{x}\,x\,a\,0)$. Denote by t_P the first time when path $P = (v\ \ldots\ 0)$ is selected by vertex v. By definition of SPVP, we have that $t_{a0} < t_{xa0} < t_{\bar{x}xa0}$ and $t_{b0} < t_{\bar{x}b0} < t_{x\bar{x}b0}$. Since vertex 0 can never withdraw path (0), vertex a (b) cannot select the empty path after t_{a0} (t_{b0}).

Suppose $t_{x\bar{x}b0} \geq t_{xa0}$. Note that, after t_{a0}, vertex a can withdraw path $(a\ 0)$ only by announcing path $(a\ x\ \bar{x}\ b\ 0)$. However, a cannot select path $(a\ x\ \bar{x}\ b\ 0)$ because this would imply $t_{ax\bar{x}b0} \leq t_{xa0} \leq t_{x\bar{x}b0} < t_{ax\bar{x}b0}$, hence a contradiction. On the other hand, if vertex a does not withdraw path $(a\ 0)$ then vertex x never selects path $(x\ \bar{x}\ b\ 0)$ because of the availability of the better ranked path $(x\ a\ 0)$.

Then it must be $t_{x\bar{x}b0} < t_{xa0}$ and, by symmetry, $t_{\bar{x}xa0} < t_{xb0}$. A contradiction: $t_{xa0} < t_{\bar{x}xa0} < t_{\bar{x}b0} < t_{x\bar{x}b0} < t_{xa0}$. ■

Lemma 4.2: For each fair activation sequence, if a vertex c_j and a time t' exist such that $\forall t > t'\ \pi_t(c_j) = (c_j\ 0)$, then a time t'' exists such that $\forall t > t''\ \pi_t(x) = (x\ a\ 0)$ and $\pi_t(\bar{x}) = (\bar{x}\ b\ 0)$.

Proof: By definition of fair activation sequence, there must exist a time $t_1 > t'$ after which paths $(x\ c_j\ 0)$ and $(\bar{x}\ c_j\ 0)$ are always available to vertices x and $\bar{x}$, respectively. This indefinitely prevents vertex x from selecting path $(x\ \bar{x}\ b\ 0)$ and vertex $\bar{x}$ from selecting path $(\bar{x}\ x\ a\ 0)$.

As a consequence and because of the fairness, there must exist a time $t_2 > t_1$ such that vertex a can only select path $(a\ 0)$ and vertex b can only select path $(b\ 0)$. Analogously, there must exist a time $t_3 > t_2$ after which paths $(x\ a\ 0)$ and $(\bar{x}\ b\ 0)$ are always available at vertices x and $\bar{x}$.

The statement follows by noting that $(x\ a\ 0)$ is the most preferred by x and $(\bar{x}\ b\ 0)$ is the most preferred by $\bar{x}$. ■

We now use the TWISTED gadget and the results from Lemmas 4.1 and 4.2 to reduce the opposite of the SAT problem, namely SAT COMPLEMENT, to SAFETY. Let F be a logical formula in conjunctive normal form with variables $X_1 \ldots X_n$ and clauses $C_1 \ldots C_m$. We construct an SPP instance S in polynomial time with respect to the size of the SAT COMPLEMENT instance as follows (see Figure 3).

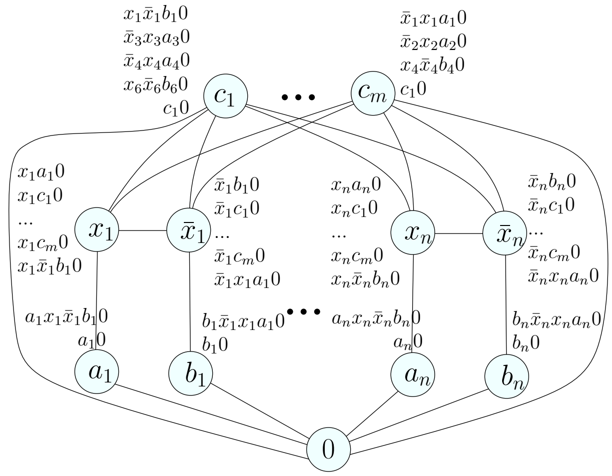

Fig. 3. Reduction from SAT COMPLEMENT to SAFETY.

For each clause C_i, add a vertex c_i to S. For each variable X_i, add a copy of the TWISTED gadget with x, $\bar{x}$, a, and b replaced by x_i, $\bar{x}_i$, a_i, and b_i, respectively. In the copy, for each clause C_j, $(x_i\ c_j\ 0) \in \mathcal{P}^{x_i}$ and $(\bar{x}_i\ c_j\ 0) \in \mathcal{P}^{\bar{x}_i}$. For each vertex c_j, path $(c_j\ x_i\ \bar{x}_i\ b_i\ 0) \in \mathcal{P}^{c_j}$ if literal X_i is in C_j and path $(c_j\ \bar{x}_i\ x_i\ a_i\ 0) \in \mathcal{P}^{c_j}$ if literal $\bar{X}_i$ is in C_j. Path $(c_j\ 0)$ is the least preferred path at each vertex c_j, while the relative preference among other paths is not significant.

Theorem 4.1: SAFETY is coNP-hard in the SPP model.

Proof: Consider a logical formula F and construct the corresponding SPP instance $S = ((V, E), \mathcal{P}, \Lambda)$ as described above. We now prove the statement in two parts.

If F is unsatisfiable then S is safe.

Consider any fair activation sequence and assume that all vertices c_j select a path $P \neq (c_j\ 0)$ infinite times. Let $W = \{x_i \in V \mid \exists c_j, \exists t : \pi_t(c_j) = (c_j\ x_i\ \bar{x}_i\ a_i\ 0)\}$ and $Z = \{\bar{x}_i \in V \mid \exists c_j, \exists t : \pi_t(c_j) = (c_j\ \bar{x}_i\ x_i\ b_i\ 0)\}$. Consider the boolean assignment M such that X_i is assigned to TRUE if $x_i \in W$, and X_i is assigned to FALSE if $\bar{x}_i \in Z$. Lemma 4.1 ensures that $Z \cap W = \oslash$. By construction of S, each clause in F is satisfied by at least a variable in M, that is a contradiction.

Then there must exist a time t' and a vertex c_k such that $\forall t > t'\ \pi_t(c_k) = (c_k\ 0)$. By Lemma 4.2, this implies that there exists a time $t'' > t'$ after which each vertex x_i always selects path $(x_i\ a_i\ 0)$ and each vertex $\bar{x}_i$ always selects path $(\bar{x}_i\ b_i\ 0)$. The fairness of the activation sequence guarantees that, eventually, each vertex c_j permanently selects $(c_j\ 0)$, each vertex a_i permanently selects $(a_i\ 0)$, and each vertex b_i permanently selects $(b_i\ 0)$. It is easy to check that such a path assignment is stable. Since any fair activation sequence leads to a stable path assignment, if F is unsatisfiable then S is safe.

If F is satisfiable then S is not safe.

Let M be a boolean assignment that satisfies F. We now show that S has at least two stable path assignments.

Let π' be a path assignment such that $\pi'(x_i) = (x_i\ a_i\ 0)$, $\pi'(\bar{x}_i) = (\bar{x}_i\ b_i\ 0)$, $\pi'(a_i) = (a_i\ 0)$, $\pi'(b_i) = (b_i\ 0)$, and

$\pi'(c_j) = (c_j\ 0)$, where $i = 1, \ldots, n$ and $j = 1, \ldots, m$. It is easy to check that π' is a stable path assignment.

Also, consider path assignment π'' defined as follows. For each variable X_i such that $M(X_i) = \top$, let $\pi''(x_i) = (x_i\ \bar{x}_i\ b_i\ 0)$, $\pi''(\bar{x}_i) = (\bar{x}_i\ b_i\ 0)$, $\pi''(a_i) = (a_i\ x_i\ \bar{x}_i\ b_i\ 0)$, $\pi''(b_i) = (b_i\ 0)$. For each variable X_i such that $M(X_i) = \bot$, let $\pi''(\bar{x}_i) = (\bar{x}_i\ x_i\ a_i\ 0)$, $\pi''(x_i) = (x_i\ a_i\ 0)$, $\pi''(a_i) = (a_i\ 0)$, $\pi''(b_i) = (b_i\ \bar{x}_i\ x_i\ a_i\ 0)$. Each vertex c_j selects in π'' the most preferred among paths in set $R_j = \{(c_j\ x_i\ \bar{x}_i\ b_i\ 0) \in \mathcal{P}^{c_j} | M(X_i) = \top\} \cup \{(c_j\ \bar{x}_i\ x_i\ a_i\ 0) \in \mathcal{P}^{c_j} | M(X_i) = \bot\}$.

Observe that $\forall j\ R_j \neq \oslash$ since each clause is satisfied by at least one variable in M. We now show that path assignment π'' is stable. Each vertex c_j, $j = 1, \ldots, m$, selects the best ranked path in R_j and, by construction, no better alternative is available at c_j. For each variable X_i such that $M(X_i) = \top$ ($M(X_i) = \bot$) vertices a_i (b_i) and $\bar{x}_i$ (x_i) select their best ranked path, while vertices b_i (a_i) and x_i ($\bar{x}_i$) cannot select any other path except the one defined by π''.

We conclude that, if F is satisfiable, then S has two stable path assignments. The statement follows by Theorem 3.1 of [14], which proves that any SPP instance with two distinct stable path assignments is not safe. ■

Theorem 4.2: SAFETY is coNP-hard in the 3-SPP model.

Proof: We can use the same reduction from SAT COMPLEMENT to SAFETY applied in Theorem 4.1. In fact, the SPP instance constructed in the reduction can be easily translated into a 3-SPP instance, since every permitted path at each vertex is uniquely identified by the first three hops in the path. The reduction proves the statement. ■

B. Safety can be efficiently checked in the 2-SPP model

The 2-SPP model allows ASes to only specify path fragments of length 2. In other words, policies can be specified only on a per-neighbor basis: all paths from the same neighbor are either accepted or filtered and are equally preferred. As in 3-SPP, any arbitrary deterministic criterion can break ties. By applying the technique in [18], it can be shown that every 2-SPP instance has at least a stable path assignment π and π can be computed in polynomial time. Observe, however, that 2-SPP allows configurations that are not safe, e.g., the famous SPP instance DISAGREE [11] can be represented in 2-SPP.

Given a 2-SPP instance $\tilde{S} = (G = (V, E), \tilde{\mathcal{P}}, \tilde{\Lambda})$, a path fragment $(u\ v)$, with $u, v \in V$, is *consistent* if there exists a sequence of permitted path fragments $P_1, P_2, \ldots, P_n$ in $\tilde{\mathcal{P}}$ such that $(u\ v)P_1P_2 \ldots P_n(0)$ is a simple path on G. Consistency of a given path fragment can be trivially checked in polynomial time. In the following, we consider only 2-SPP instances in which all permitted path fragments are consistent.

We show an algorithm, called NH-GREEDY, that efficiently solves SAFETY in 2-SPP. NH-GREEDY is an adaptation of the greedy algorithm in [3]. NH-GREEDY incrementally grows a set of *stable* vertices for which convergence is guaranteed. The set of stable vertices at iteration i of NH-GREEDY is denoted by V_i. At iteration i NH-GREEDY also computes a partial path assignment π_i^*, that is, a path assignment where $\forall u \notin V_i$ $\pi_i^*(u) = \epsilon$. At the beginning, $V_0 = \{0\}$ and $\pi_0^*(0) = (0)$. Let H_i be the set of vertices $u \notin V_i$ such that the most preferred path fragment is either $B^u = \epsilon$ or $B^u = (u\ v)$, where $v \in V_i$. If H_i is not empty, then $V_{i+1} = V_i \cup H_i$, $\pi_{i+1}^*(u) = \pi_i^*(u)$ if $u \in V_i$, and $\pi_{i+1}^*(u) = B^u \pi_i^*(u)$ for each $u \in H_i$. Otherwise, if H_i is empty, NH-GREEDY terminates. At each iteration, NH-GREEDY either inserts at least one vertex in V_i or terminates, hence it terminates after at most $|V|$ iterations. If NH-GREEDY terminates after k iterations with $V_k = V$ then we say that it *succeeds*, otherwise it *fails*. Being derived from the algorithm in [3], NH-GREEDY inherits the properties shown in [19]. In particular, this implies that NH-GREEDY is correct, i.e., after k iterations each vertex $v \in V_k$ is guaranteed to eventually select path $\pi_k^*(v)$ in any fair activation sequence. As a consequence, if NH-GREEDY succeeds then the 2-SPP instance is safe. We now show that if NH-GREEDY fails the instance is not safe.

Let $G' = (V, E')$ be the directed graph such that $(u, v) \in E'$ iff $(u\ v) \in \tilde{\mathcal{P}}^x$. Given a partial path assignment π and a vertex u such that $\tilde{\mathcal{P}}^u \neq \{\epsilon\}$ and $\pi(u) = \epsilon$, the *ideal path* P_u^π of u in π is the simple path from u to 0 obtained by performing a depth-first visit on G' starting from u. Vertices are visited according to $\tilde{\Lambda}$, i.e., the neighbor with the highest preference is visited first. By definition, $P_u^\pi = (w_1\ \ldots\ w_n\ v_1\ \ldots\ v_m)$, where $w_1 = u$, $v_m = 0$, $n \geq 1$, $m \geq 1$, $(u\ w_1)$ is the most preferred fragment in $\mathcal{P}^u$, $\pi(w_i) = \epsilon$ for $i = 1, \ldots, n$, and $\pi(v_j) = (v_j\ \ldots\ v_m)$ for $j = 1, \ldots, m$. Intuitively, the ideal path of u traverses the best ranked neighbor of u and such that all vertices $w_i \in P_u^\pi$ select the best-ranked simple path that extends a path in π and ends in 0. Observe that such a path must exist because all path fragments are assumed to be consistent, i.e., $(u\ w_1)$ generates at least a path on G.

Assume that NH-GREEDY fails on a 2-SPP instance $\tilde{S}$ after k iterations with partial path assignment π_k^* and let u be any vertex in $V - V_k$.

Lemma 4.3: There exists a stable path assignment $\bar{\pi}$ on $\tilde{S}$ such that u selects its ideal path, i.e., $\bar{\pi}(u) = P_u^{\pi_k^*}$.

Proof: We construct a sequence of partial path assignments $\pi_1, \pi_2, \ldots, \bar{\pi}$ by iteratively growing π_k^*. Let $P_u^{\pi_k^*} = (u\ w_1\ \ldots\ w_n\ v_1\ \ldots\ v_m)$ be the ideal path of vertex u in π_k^*. Let $\pi_1(u) = P_u^{\pi_k^*}$, for each $w_i \in P_u^{\pi_k^*}$ let $\pi_1(w_i) = (w_i\ \ldots\ w_n\ v_1\ \ldots\ v_m)$ and for each $v \in V_k$ let $\pi_1(v) = \pi_k^*(v)$. Then, we consider any other vertex z such that $\pi_1(z) = \epsilon$ and $z \in V - V_k$ (if one exists, otherwise stop). Given $P_z^{\pi_1}$ the ideal path of z, we construct the (partial) path assignment π_2 by extending π_1 as above. Since V is finite, we eventually find a path assignment $\bar{\pi}$ defined for each $v \in V$.

We now show that $\bar{\pi}$ is stable. Suppose, for a contradiction, that there exists a vertex v that has an alternative path towards 0 that is preferred to $\bar{\pi}(v)$. By construction, v must either be in V_k or be part of the ideal path of some vertex x. In the first case, being $\bar{\pi}$ an extension of π_k^*, v is guaranteed to select path $\bar{\pi}(v)$. In the latter case, by definition of ideal path, v can not have a better-ranked alternative, since the depth-first visit analyzes paths at each vertex in a decreasing order of preference. In both cases, we have a contradiction. ■

Theorem 4.3: SAFETY can be solved in polynomial time in the 2-SPP model.

Proof: Given a 2-SPP instance S, S is safe if and only if NH-GREEDY succeeds. We have already discussed that if NH-GREEDY succeeds S is safe. On the other hand, if NH-GREEDY fails after k iterations, it is possible to build two distinct stable path assignments. In fact, let u be any vertex in $V - V_k$. Lemma 4.3 ensures that there exists a stable path assignment π' such that $\pi'(u) = P_u^{\pi_k^*}$. Path $P_u^{\pi_k^*}$ must be in the form $P_u^{\pi_k^*} = P'(z\ v)P''$ where $z \notin V_k$ and $v \in V_k$. Observe that $z \neq u$, since $u \notin V_k$. Consider the stable path assignment π'' such that $\pi''(z) = P_z^{\pi_k^*}$, constructed as in Lemma 4.3. Obviously, $\pi' \neq \pi''$ at least for vertex z since $z \notin V_k$. Since two distinct stable path assignments exist, by Theorem 3.1 of [14] S is not safe. ■

V. SEARCHING FOR DISPUTE WHEELS

In Section IV we proved that SAFETY turns out to be a computationally hard problem. A possible way to overcome the unfeasibility of testing SAFETY could be verifying if at least sufficient conditions for SAFETY are satisfied.

In [3] a celebrated sufficient condition for SAFETY has been introduced. Namely, it has been shown that safety is guaranteed if the BGP network does not contain a *dispute wheel* (*DW*), a particular structure that involves cyclic preferences which cannot be simultaneously satisfied. In the SPP model, a DW $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ is a sequence of vertices $\vec{\mathcal{U}} = (u_0\ u_1 \dots u_{k-1})$ and sequences of nonempty paths $\vec{\mathcal{Q}} = (Q_0\ Q_1 \dots Q_{k-1})$, called *spoke paths*, and $\vec{\mathcal{R}} = (R_0\ R_1 \dots R_{k-1})$, called *rim paths*, such that:

(*i*) R_i is a path from u_i to u_{i+1}
(*ii*) $Q_i \in \mathcal{P}^{u_i}$
(*iii*) $R_i Q_{i+1} \in \mathcal{P}^{u_i}$
(*iv*) $\lambda^{u_i}(Q_i) \geq \lambda^{u_i}(R_i Q_{i+1})$

where all indexes are to be intended modulo k. Since an instance of k-SPP can be uniquely translated into an SPP instance, we can extend the definition of DW as follows: we say that an instance of k-SPP *contains* a DW if its translation to SPP contains a DW. Verifying the absence of a DW in a BGP network is referred to as the NO-DW problem. In the SPP model NO-DW can be solved in polynomial time [7] by finding a cycle in an auxiliary graph called *dispute digraph*, whose construction takes polynomial time.

In the following, we analyze the computational complexity of NO-DW in the 3-SPP model. We do it in two steps. First, we deal with the basic problem of deciding whether a given vertex of a 3-SPP instance can establish a path to 0. We call this problem PATH and we show that it is NP-complete. Second, we exploit such a result to prove that NO-DW in the 3-SPP model is coNP-complete.

PATH is NP-complete since it is possible to reduce 3-SAT to PATH. Let F be a 3-SAT formula with variables $X_1, \dots, X_n$ and clauses $C_1, \dots, C_m$. We construct a 3-SPP instance as follows. For each variable X_i we insert vertices v_i, x_i, and $\bar{x}_i$, and we build a gadget having edges (v_i, x_i) and $(v_i, \bar{x}_i)$. For each clause C_j we build a gadget consisting of vertices

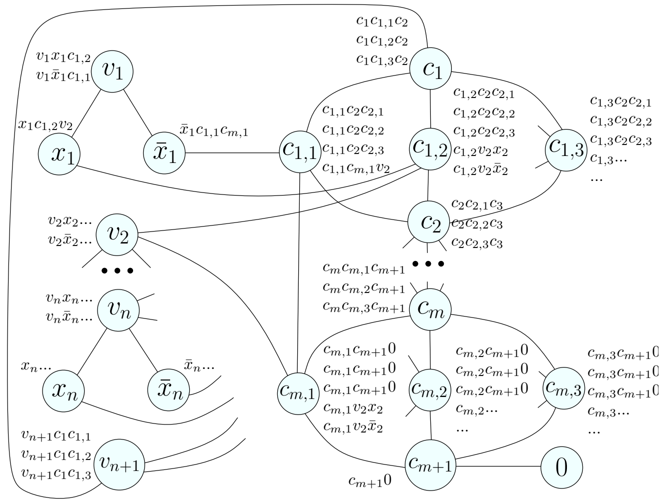

Fig. 4. Reduction from 3-SAT to PATH.

c_j and $c_{j,k}$ and edges $(c_j, c_{j,k})$, $(c_{j,k}, c_{j+1})$ with $k = 1, 2, 3$. Also, we add to the instance vertices v_{n+1}, c_{m+1} and 0, and edges $(c_{m+1}, 0)$ and (v_{n+1}, c_1). Fig. 4 shows an example of the construction, where variable gadgets are on the left side while clause gadgets are on the right side.

Intuitively, vertex v_i attempts to establish a path to 0 via x_i ($\bar{x}_i$) if the corresponding 3-SAT variable X_i is TRUE (FALSE). Vertices $c_{j,k}$ are called *literal* vertices because each of them represents one of the three literals that appear in clause C_j.

Consider literal X_i, $i = 1, \dots, n$. Let $P = (v_i\ x_i\ c_{j_1,k_1}\ \dots\ c_{j_n,k_n}\ v_{i+1})$ be the path from vertex v_i to vertex v_{i+1} that traverses all the literal vertices c_{j_p,k_p} such that the corresponding literal in clause C_{j_p} is $\bar{X}_i$. If there are no such literals, then path P simply consists of edges (v_i, x_i) and (x_i, v_{i+1}). We add to the graph constructed so far all the edges of P. We apply exactly the same procedure for literal $\bar{X}_i$. We then get from path P all the ordered triples of consecutive vertices and add each triple $(u\ v\ w)$ to $\tilde{\mathcal{P}}^u$. For example, in Fig. 4 there is a path $(v_1\ \bar{x}_1\ c_{1,1}\ c_{m,1}\ v_2)$ because we assume, without loss of generality, that the first literal both in C_1 and in C_m is X_1. For each vertex c_j, set $\tilde{\mathcal{P}}^{c_j}$ only contains paths $(c_j\ c_{j,k}\ c_{j+1})$, with $k = 1, 2, 3$ and for each vertex $c_{j,k}$, we add to $\tilde{\mathcal{P}}^{c_{j,k}}$ paths $(c_{j,k}\ c_{j+1}\ c_{j+1,l})$, with $l = 1, 2, 3$. This construction ensures that if vertex v_i attempts to establish a path to 0 via x_i ($\bar{x}_i$), it cannot use a path including $c_{j,k}$ iff $\bar{X}_i$ (X_i) is the k-th literal in C_j, representing the fact that clause C_j cannot be satisfied by literal $c_{j,k}$. We define $\tilde{\mathcal{P}}^{v_{n+1}} = \{(v_{n+1}\ c_1\ c_{1,k}) | \forall k = 1, 2, 3\}$ and $\tilde{\mathcal{P}}^{c_{m+1}} = \{(c_{m+1}\ 0)\}$.

Function $\lambda^{c_{j,k}}$, where $j = 1, \dots, m$ and $k = 1, 2, 3$, is such that paths $(c_{j,k}\ c_{j+1}\ c_{j+1,l})$, with $l = 1, 2, 3$, are better ranked than others. Preferences at vertices v_i, x_i, $\bar{x}_i$ and c_j, where $i = 1, \dots, n+1$ and $j = 1, \dots, m+1$, can be assigned arbitrarily. It is easy to check that the instance of 3-SPP can be built in polynomial time.

Lemma 5.1: PATH is NP-complete in the 3-SPP model.

Proof: Consider the construction depicted in Fig. 4. We now show that vertex v_1 can establish a path to 0 iff the corresponding 3-SAT formula F is satisfiable.

Observe that every path P from v_1 to 0, if any, must be in the form $P = AB$ where $A = (v_1 \dots v_2 \dots v_{n+1})$ and $B = (v_{n+1}\ c_1\ c_{1,j_1}\ \dots c_m\ c_{m,j_m} 0)$. Since vertex v_i must choose either x_i or $\bar{x_i}$ and there is only one path connecting x_i ($\bar{x_i}$) to v_{i+1}, path A can be mapped to a boolean assignment for F. By construction, only literal vertices $c_{j,k}$ can appear twice in P, since they can appear both in A and in B.

Now, if $P = AB$ exists, then every c_j can reach 0 via one of its neighbors $c_{j,1}$, $c_{j,2}$ and $c_{j,3}$ which is not traversed by path A. By construction, this implies that the boolean assignment mapped to path A satisfies at least one literal in every clause, hence F is satisfiable.

On the other hand, if there is no path P from v_1 to 0, then for any choice of path A there exists a vertex c_j that is unable to reach 0 via any of its neighbors because they all appear in A. By construction, this implies that for each boolean assignment there exists a clause C_j that is false, hence F is unsatisfiable.

The above arguments prove that PATH is NP-hard. NP-completeness follows by noting that a path P from v_1 to 0 is a succinct certificate for PATH because P has polynomial size and it takes polynomial time to check if P can be generated by any fragment of v_1. ∎

We now use the reduction as above for proving that NO-DW is coNP-complete. First of all, we the 3-SPP instance built in the reduction does not contain any DW.

Lemma 5.2: The 3-SPP instance S constructed in the reduction from 3-SAT to PATH (see Fig. 4) contains no DW.

Proof: Suppose, for a contradiction, that S contains a DW and assume that no vertex c_i can appear in any rim path. We now show that rim paths of such a DW do not form a cycle, that is a contradiction since concatenating rim paths must result in a cycle by definition of DW (each rim path connects a pivot vertex with its successor).

By construction, permitted paths of all the vertices in S are subpaths of $P = P_1\ \dots\ P_n\ (v_{n+1}\ c_1)\ Q_1\ \dots\ Q_m\ (c_{m+1}\ 0)$. Paths Q_i are such that $Q_i = \{c_i\ c_{i,j}\ c_{i+1}\}$, where j is either 1, 2, or 3. Each path P_i starts at v_i, ends in v_{i+1}, and traverses x_i ($\bar{x}_i$) and all the vertices $c_{j,k}$ such that the corresponding literal in clause C_j is $\bar{X}_i$ (X_i). This implies that $P_j \cap P_{j+1} = \{v_{j+1}\}$ for each j, and $P_j \cap P_k = \oslash$, if $k \notin \{j, j+1\}$. Since no rim path can contain a node c_i, all the rim paths must be subpaths of $P_1\ P_2\ \dots\ P_n$. However, since vertices v_i are ordered and all paths P_i intersects only at vertices v_i, no cycle among rim paths can be built, yielding a contradiction.

The proof is completed by showing that no vertex c_i can appear in any rim path of any DW Π. In fact, suppose that there exists a non empty set of vertices $Z = \{c_j, \dots, c_k\}$ such that each vertex $c_i \in Z$ appears in one or more rim paths. Obviously, c_{m+1} cannot belong to Z. Consider, among all the vertices in Z, the vertex c_h with the highest index. Let R be a rim path in which appears c_h and let $R[c_h]$ be the subpath of R starting from c_h. By definition of c_h and by construction of S, $R[c_h]$ can only be $(c_h\ c_{h,h'})$, with $h' = 1, 2, 3$. In fact, all permitted paths at c_h are sequences of vertices c_i and $c_{i,j}$, such that $i > h$ and c_{h+1} cannot appear in $R[c_h]$ by definition of c_h. Hence, vertex $c_{h,h'}$ must be a pivot vertex of Π, and its spoke path must be a path $(c_{h,h'}\ c_{h+1}\ \dots\ 0)$ since it must be extended by a permitted path of c_h. By definition of DW, the rim path of $c_{h,h'}$ should be one among paths $(c_{h,h'}\ c_{h+1}\ \dots\ 0)$, that is, c_{h+1} is also on a rim path. This leads to a contradiction, because c_h is defined to be the vertex with the highest index among those appearing in a rim path. ∎

Theorem 5.1: NO-DW is coNP-complete in the 3-SPP model.

Proof: We prove the statement by reducing 3-SAT COMPLEMENT to NO-DW. Let F be a logical formula with variables $X_1, \dots, X_n$ and clauses $C_1, \dots, C_m$. We construct an instance $\tilde{S} = ((V, E), \tilde{\mathcal{P}}, \tilde{\Lambda})$ of 3-SPP as follows. Let $\tilde{S}' = ((V', E'), \tilde{\mathcal{P}}', \tilde{\Lambda}')$ be the 3-SPP instance constructed as above (see Fig. 4). Let $V = V' \cup \{1, 2\}$, let $E = E' \cup \{(1, v_1), (1, 2), (2, 0)\}$, let $\tilde{\mathcal{P}} = \tilde{\mathcal{P}}' \cup \{(1\ v_1\ x_1), (1\ v_1\ \bar{x_1}), (2\ 0), (1\ 2\ 0), (2\ 1\ v_1)\}$ and let $\tilde{\Lambda} = \tilde{\Lambda}' \cup \{\tilde{\lambda}^1, \tilde{\lambda}^2\}$, where $\tilde{\lambda}^1$ is such that path $(1\ 2\ 0)$ is most preferred and $\tilde{\lambda}^2$ is such that path $(2\ 1\ 0)$ is most preferred.

Intuitively, we added two extra vertices 1 and 2, and defined policies such that a DW exists in $\tilde{S}$ only if 1 can establish a path to 0. By applying the same arguments as in the proof of Lemma 5.1 we therefore have that $\tilde{S}$ has no dispute wheel iff F is unsatisfiable. This implies that NO-DW is coNP-hard in the 3-SPP model. The proof is complete by noting that a DW on $\tilde{S}$ is a succinct disqualification for NO-DW, that is, a succinct proof that $\tilde{S}$ is a negative instance. ∎

VI. Safety Under Filtering and Robustness

In this section we study the computational complexity of safety under filtering and robustness.

Problem SAFETY UNDER FILTERING (SUF) [6] is defined as follows: given an SPP instance S, will S remain safe under arbitrary filtering of paths? Similarly, the ROBUSTNESS problem [3] requires that the input SPP instance be safe even under arbitrary link failures. It has been proved in [12] that the two problems are distinct, as there exist SPP instances that are robust but not safe under filtering.

It is known [12] that an SPP instance is safe under filtering iff it does not contain a dispute reel (DR). Intuitively, a dispute reel is a dispute wheel such that spoke paths form a tree T and rim paths never intersect T nor contain more than two pivot vertices. Let $P[v]$ denote the subpath of P starting at vertex v. A *dispute reel* (DR) is a dispute wheel such that

(*i*) (*Pivot vertices appear in exactly three paths*) – for each $u_i \in \vec{\mathcal{U}}$, u_i only appears in paths Q_i, R_i and R_{i-1}.
(*ii*) (*Spoke and rim paths do not intersect*) – for each $u \notin \vec{\mathcal{U}}$, if $u \in Q_i$ for some i, then no j exists such that $u \in R_j$.
(*iii*) (*Spoke paths form a tree*) – for each distinct $Q_i, Q_j \in \vec{\mathcal{Q}}$, if $v \in Q_i \cap Q_j$, then $Q_i[v] = Q_j[v]$.

SUF, ROBUSTNESS and DR are defined in the SPP model. The definition of DR can be extended to k-SPP by translating the considered k-SPP instance to SPP. SUF and ROBUSTNESS are defined in 3-SPP as the problems of determining if an input 3-SPP instance is safe even under arbitrary filtering of path fragments or under arbitrary link failures, respectively. It is easy to check that a 3-SPP instance is robust iff the corresponding SPP instance is robust. On the contrary, it is not known if a SUF 3-SPP instance corresponds to a SUF SPP instance, nor if the absence of a DR is a characterization for SUF in the 3-SPP model.

A. No Dispute Reel is CoNP-Complete

We now prove that NO-DR is coNP-hard by reducing 3-SAT COMPLEMENT to SAT in polynomial time. Refer to Fig. 5 for an example of the reduction.

Let F be a logical formula, with variables $X_1, \ldots, X_n$ and clauses $C_1, \ldots, C_m$. For each variable X_i, we add to the SPP instance a gadget consisting of three vertices, namely a_i, x_i, and $\bar{x}_i$, and four edges, namely $(x_i\ 0)$, $(\bar{x}_i\ 0)$, $(a_i\ x_i)$ and $(a_i\ \bar{x}_i)$. Vertices x_i and $\bar{x}_i$ have no permitted paths other than $(x_i\ 0)$ and $(\bar{x}_i\ 0)$, respectively. Permitted paths at vertex a_i are $\mathcal{P}^{a_i} = \{(a_i\ x_i\ 0), (a_i\ \bar{x}_i\ 0)\}$ and the ranking among them is not significant. Intuitively, a_i represents variable X_i. Gadgets corresponding to variables are at the bottom of Fig. 5.

For each clause C_j, we add to the SPP instance a gadget containing vertices c_j, $c_{j,i}$, and edges $(c_j, c_{j,i})$ and $(c_{j,i}, c_{j+1})$, where $i = 1, \ldots, 3$. Intuitively, vertex c_j (*clause vertex*) represents clause C_j while vertex $c_{j,i}$ (*literal vertex*) represents the i-th literal in C_j. Further, if X_l appears in the i-th literal in C_j, then we add an edge $(a_l, c_{j,i})$, and we set $\mathcal{P}^{c_{j,i}} = \{(c_{j,i}\ c_{j+1}\ 0), (c_{j,i}\ a_l\ x_l\ 0)\}$ if literal represented by $c_{j,i}$ is X_l, $\mathcal{P}^{c_{j,i}} = \{(c_{j,i}\ c_{j+1}\ 0), (c_{j,i}\ a_l\ \bar{x}_l\ 0)\}$ otherwise. Among the two paths in $\mathcal{P}^{c_{j,i}}$, $(c_{j,i}\ c_{j+1}\ 0)$ is the most preferred. The permitted paths at vertex c_j are $(c_j\ 0)$ plus the extension of the longest path permitted at each vertex $c_{j,i}$, $i = 1, \ldots, 3$. Path $(c_j\ 0)$ is the least preferred path, while the ranking of other paths can be arbitrary. Gadgets corresponding to clauses are placed at the top of Fig. 5.

Observe that the SPP instance built in the reduction contains several DWs. Vertices a_i, x_i, $\bar{x}_i$ can not be pivot vertices of any dispute wheel, since they only have direct paths to 0. In fact, by arbitrarily picking exactly one literal vertex $c_{j,i}$ for each clause vertex c_j, we construct a DW where pivot vertices are all clause vertices and the selected literal vertices.

For any DW Π, each pivot appears in exactly three paths and spoke paths never intersect rim paths, hence conditions (*i*) and (*ii*) of the definition of DR are satisfied. However, spoke paths are not guaranteed to form a tree (condition (*iii*) of the definition of DR), so DWs are not guaranteed to be DRs.

Since spoke paths in Π only share vertices a_i, condition (*iii*) is satisfied only if there are no two distinct spoke paths Q_1 and Q_2 in Π such that $Q_1 = (\ldots a_i\ x_i\ 0)$ and $Q_2 = (\ldots a_i\ \bar{x}_i\ 0)$, which represents the fact that variable X_i cannot be TRUE and FALSE at the same time.

Theorem 6.1: NO-DR is coNP-complete in the SPP model.

Proof: Consider a logical formula F and construct the corresponding SPP instance S as described above.

If F is unsatisfiable then S does not contain a DR.

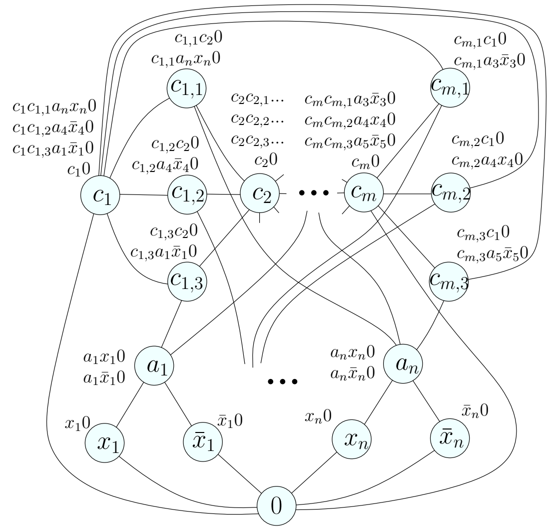

Fig. 5. Reduction from SAT COMPLEMENT to SUF.

Suppose, for a contradiction, that S contains a DR Π. Then, condition (*iii*) ensures that, for each a_i, either path $(a_i\ x_i\ 0)$ or path $(a_i\ \bar{x}_i\ 0)$ is a subpath of all spoke paths that traverse vertex a_i. This property allows us to construct a boolean assignment for F by setting variable X_i to TRUE if there exists a spoke path $Q' = (\ldots a_i\ x_i\ 0)$ or to FALSE if there exists a spoke path $Q'' = (\ldots a_i\ \bar{x}_i\ 0)$.

As we already observed, Π contains exactly one literal vertex for each clause vertex. By construction of S, we have that the boolean assignment corresponding to Π satisfies at least one literal in each clause in F, contradicting the hypothesis that F is unsatisfiable.

If F is satisfiable then S contains at least one DR.

Consider a boolean assignment M that satisfies F. We will now show a DR $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ in S. Vertices c_j must be pivot vertices, that is, $u_{2j-1} = c_j$ and $Q_{2j-1} = (c_j\ 0)$ for $j = 1, \ldots, m$. For each literal vertex $c_{j,i}$, if its least preferred path is $(c_{j,i}\ a_i\ x_i\ 0)$ and $M(X_i) = \top$ then we set $u_{2j} = c_{j,i}$, $Q_{2j} = (c_{j,i}\ a_i\ x_i\ 0)$, $R_{2j-1} = (c_j\ c_{j,i})$, and $R_{2j} = (c_{j,i}\ c_{j+1})$. We set u_{2j}, R_{2j} and R_{2j-1} to the same values also if the least preferred path of $c_{j,i}$ is $(c_{j,i}\ a_i\ \bar{x}_i\ 0)$ and $M(X_i) = \bot$, however in this case we set a different spoke path $Q_{2j} = (c_{j,i}\ a_i\ \bar{x}_i\ 0)$. Whenever multiple literal vertices $c_{j,i}$ for the same clause vertex c_j satisfy the above conditions, we arbitrarily pick only one among them.

It is easy to see that, since each clause in F is satisfied by at least one literal, Π is a DW. Moreover, by construction of Π we have that for each vertex a_i only one among $(a_i\ x_i\ 0)$ and $(a_i\ \bar{x}_i\ 0)$ can be traversed by spoke paths in Π, hence satisfying condition (*iii*) of the definition of DR. Conditions (*i*) and (*ii*) are trivially satisfied by Π. Hence, Π is a DR.

CoNP-completeness follows from noting that a DR on S is a succinct disqualification for NO-DR. ■

We now state the complexity of NO-DR in 3-SPP.

Theorem 6.2: NO-DR is coNP-complete in the 3-SPP model.

Proof: Observe that all the permitted paths in SPP instance built in the reduction 3-SAT COMPLEMENT to 3-SPP are entirely identified by the first three hops. Hence, an analogous reduction can be applied from 3-SAT COMPLEMENT to 3-SPP. The statement follows from the fact that a DR on a 3-SPP instance is a succinct disqualification for NO-DR. ■

B. Complexity of Safety Under Filtering and Robustness

Since the absence of a DR is a characterization of SUF in the SPP model, we can state the following theorems.

Theorem 6.3: SUF is coNP-complete in the SPP model.

Proof: The statement directly follows from Theorem 6.1 considering that the absence of a DR is a necessary and sufficient condition for SUF in the SPP model [12]. ■

Theorem 6.4: SUF is coNP-hard in the 3-SPP model.

Proof: Let S be the SPP instance in Fig. 5 and construct the 3-SPP instance S' by truncating all paths in S with length greater than 3. Since each permitted path in S is identified by its first three hops, there is a one-to-one mapping between permitted paths in S and permitted paths in S'. This implies that each filter in S can be mapped to a unique filter in S'. We conclude that S' is SUF iff S is SUF, hence a construction analogous to that described in Section VI-A can be applied to reduce from 3-SAT COMPLEMENT to SUF in 3-SPP. ■

In general, SUF implies ROBUSTNESS, while the opposite does not hold [12]. However, observe that the SPP instance in Fig. 5 is SUF iff it is also robust. In fact, filtering a path $P = (u\ v\ \ldots 0)$ at vertex u is equivalent to removing edge (u, v) from the graph. This property allows us to reduce 3-SAT COMPLEMENT to ROBUSTNESS using the same reduction used in Theorem 6.3.

Theorem 6.5: ROBUSTNESS is coNP-hard in the SPP model.

Since a 3-SPP instance is robust iff the corresponding SPP instance is robust, we can directly extend Theorem 6.5.

Theorem 6.6: ROBUSTNESS is coNP-hard in the 3-SPP model.

VII. Conclusions

The design of BGP as a protocol where ASes interact in full autonomy poses a fundamental tradeoff between the expressiveness of routing policies and risks of routing oscillations. Restricting the expressiveness of routing policies can be done either dynamically, e.g., by extending the protocol with oscillation-detection capabilities, or statically, e.g., by limiting the expressive power of BGP configuration languages. Unfortunately, the first option is affected by severe deployment issues. Prior contributions that explored the second option (e.g., [9]) devised restrictions on BGP policies that guarantee convergence, but affect both the autonomy and the expressiveness, e.g., by forcing ASes to filter certain paths.

In this paper we take a different approach, which can be summarized by the following question: assuming that ASes preserve their autonomy, how expressive can policies be in order to allow an efficient static assessment of BGP stability?

	2-SPP	3-SPP	SPP
SAFETY	P	coNP-hard	coNP-hard
NO-DW	?	coNP-complete	P [7]
NO-DR	?	coNP-complete	coNP-complete
SUF	?	coNP-hard	coNP-complete
ROBUSTNESS	?	coNP-hard	coNP-hard

TABLE I
COMPLEXITY OF BGP STABILITY PROBLEMS IN DIFFERENT MODELS. P STANDS FOR POLYNOMIAL TIME SOLVABLE.

Unfortunately, we find that the most interesting problems about BGP stability are computationally intractable if ASes fully preserve their autonomy and are allowed to specify policies as expressive as Local Transit policies. Table I summarizes the results. While such results are primarily related to BGP, they can be generalized to any policy-based path vector routing protocol. Our findings show that computational tractability of BGP stability can be achieved by restricting the expressiveness of the policies alone, preserving ASes' autonomy. Determining whether there exist restrictions that keep the policies expressive enough for practical uses remains an interesting open problem.

References

[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, 2006.

[2] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," *Computer Networks*, vol. 32, no. 1, 2000.

[3] T. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. on Networking*, vol. 10, no. 2, pp. 232–243, 2002.

[4] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A measurement study on the impact of routing events on end-to-end Internet performance," in *Proc. SIGCOMM*, 2006.

[5] N. Kushman, S. Kandula, and D. Katabi, "Can you hear me now?! It must be BGP," in *Computer Communication Review*, 2007.

[6] N. Feamster, R. Johari, and H. Balakrishnan, "Implications of autonomy for the expressiveness of policy routing," *IEEE/ACM Trans. on Networking*, vol. 15, no. 6, 2007.

[7] T. Griffin, F. B. Shepherd, and G. Wilfong, "Policy disputes in path-vector protocols," in *Proc. ICNP*, 1999.

[8] C. T. Ee, V. Ramachandran, B.-G. Chun, K. Lakshminarayanan, and S. Shenker, "Resolving inter-domain policy disputes," in *Proc. SIGCOMM*, 2007.

[9] L. Gao and J. Rexford, "Stable internet routing without global coordination," in *Proc. SIGMETRICS*, 2000.

[10] A. D. Jaggard and V. Ramachandran, "Robustness of class-based path-vector systems," in *Proc. ICNP*, 2004.

[11] T. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *Proc. SIGCOMM*, 1999.

[12] L. Cittadini, G. D. Battista, M. Rimondini, and S. Vissicchio, "Wheel + ring = reel: the impact of route filtering on the stability of policy routing," in *Proc. ICNP*, 2009.

[13] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *Proc. SIGCOMM*, 2009.

[14] R. Sami, M. Schapira, and A. Zohar, "Searching for stability in interdomain routing," in *Proc. INFOCOM*, 2009.

[15] A. Fabrikant and C. Papadimitriou, "The complexity of game dynamics: Bgp oscillations, sink equilibria, and beyond," in *Proc. SODA*, 2008.

[16] L. Cittadini, "Understanding and detecting BGP instabilities," Ph.D. dissertation, Università degli Studi "Roma Tre", 2010.

[17] C. Papadimitriou, *Computational complexity*. Addison-Wesley, 1994.

[18] J. Feigenbaum, R. Sami, and S. Shenker, "Mechanism design for policy routing," *Distributed Computing*, vol. 18, no. 4, pp. 293–305, 2006.

[19] L. Cittadini, M. Rimondini, M. Corea, and G. Di Battista, "On the Feasibility of Static Analysis for BGP Convergence," in *Proc. IM*, 2009.