Stable Path(s) Assignment for Inter-domain Routing

Rachit Agarwal  
Department of Electrical and Computer Engineering,  
University of Illinois at Urbana-Champaign,  
IL, USA  
agarwa16@illinois.edu

Virajith Jalaparti, Brighten Godfrey, Matthew Caesar  
Department of Computer Science,  
University of Illinois at Urbana-Champaign,  
IL, USA  
{jalapar1, pbg, caesar}@illinois.edu

Abstract

The Border Gateway Protocol (BGP) is the inter-domain routing protocol in the Internet that allows each autonomous system (AS) to select routes to the destinations based on locally determined policies. It has been shown that the policy autonomy exercised by ASes may result in persistent oscillations in BGP. Current solutions either rely on globally consistent policy assignments (which are hard to achieve in a distributed fashion), or require significant deviations from locally assigned policies (which reduce flexibility).

In this paper, we take a different approach. Namely, we propose multipath routing to resolve the conflict with policy autonomy and system stability. We design an algorithm STABLE PATH(S) ASSIGNMENT (SPA), that provably detects persistent oscillations and eliminates these oscillations by assigning multiple paths to some ASes in the network. We design a distributed protocol for SPA and present tight bounds on the number of paths assigned by the algorithm to the ASes. Using simulations on the AS graph, we show that SPA assigns at most two paths to any AS in the network (in 99.9% of the instances), while assigning a single path in absence of persistent oscillations. Our evaluation results suggest that SPA can effectively detect networks that have a stable state but can potentially face persistent oscillations, and assigns a single path to the ASes in such networks.

Keywords

Internetworking, Routing, Algorithms, Protocols

1. Introduction

The Internet connects thousands of independently operating networks, known as Autonomous Systems (ASes), corresponding to Internet service providers, companies, universities, etc. Individual ASes must cooperate to construct routes, yet at the same time, they often compete for business and have very different operational goals. To allow this highly diverse set of networks to cooperate, the Border Gateway Protocol (BGP) was developed. BGP is a routing protocol that computes paths across ASes, and comes with a highly flexible set of policy knobs and configuration parameters to allow individual ASes to express their routing goals and preferences. However, it has been shown that this freedom of route selection exercised by the ASes can cause instability in inter-domain routing manifesting in the form of persistent route oscillations [1], [2]. Several studies have demonstrated that such routing oscillations can significantly degrade the end-to-end performance of the Internet [3], [4].

Currently, there are two main approaches to avoid persistent route oscillations in BGP. The first approach is to enforce restrictions on the set of paths an AS can select its route from [5]. Doing this consistently across ASes (which do not share a global view of the topology) in a distributed fashion is a challenging problem. The other approach requires ASes to dynamically detect persistent oscillations and deviate from local preferences, when oscillations do occur [6]. This approach, while interesting in the sense that a limitation on policy autonomy is imposed only in the presence of persistent oscillations, requires significant deviations from locally assigned policies.

In this paper, we take a different approach to resolving the conflict between policy autonomy and system stability based on the use of multipath routing. The key observation in our approach is that the fundamental source of oscillations lies in the interdependence of routes across ASes, and this interdependence can be broken by propagating additional paths, while allowing each AS to route its own traffic through its highest preferred available path. Instead of requiring an AS to change its preferences over the set of paths, our approach propagates a small number of additional paths. Propagating multiple paths may be a desirable approach to this problem, as recent work has demonstrated that multipath routing has several benefits in its own right, including improved security, availability, and flexibility [7], [8], [9].

To formalize the idea of stability using multipath routing, we present the Multiple Stable Paths Problem (MSPP) [4], where an AS is allowed to be assigned a set of paths. We present a centralized algorithm STABLE PATH(S) ASSIGNMENT (SPA) [4], that given an instance of MSPP, assigns stable paths to each AS in the network in a way that guarantees that (1) each AS is assigned its highest preferred path among the set of available paths and (2) the set of assigned paths is small. If the set of assigned paths are all of size one, then the solution corresponds to the route assignment using BGP. The two key challenges addressed by SPA are: one, in absence of routing oscillations, the solution of SPA must correspond to that of BGP and two, a way of keeping the size of the set of assigned paths small when routing oscillations do occur, which corresponds to taking away only a little autonomy from the ASes.
In order to understand the behavior of SPA in distributed settings, we design a distributed protocol for SPA \cite{6} and evaluate the performance of SPA on the AS graph \cite{6}. Our evaluation results suggest three main conclusions; first, misconfigurations in the network can have significant impact on the routing stability in the network. In particular, we show that even with 1% of the ASes misconfigured, the network may have dispute wheels with overwhelming probability. Second, assigning at most one extra path to very few ASes in the network may be sufficient to guarantee routing stability. In particular, in 99.9% of the networks that had a dispute wheel, SPA assigned at most one extra path to any AS in the network. Furthermore, in 99.9% of the instances when SPA assigned more than one path to any AS, less than 0.04% of the ASes were assigned the extra path.

Finally, our evaluation results also suggest that SPA can detect (with high probability) the networks that have a dispute wheel and also have a stable state as well. In particular, assuming that all the networks that had a dispute wheel also had a stable state, SPA detected (and assigned) the stable state for more than 91% of these networks. This means that for more than 91% of the networks that had a dispute wheel, SPA assigned a single path to each AS in the network.

2. Related Work

The possibility of persistent oscillations in BGP was first noticed in \cite{1}. Griffin et al. \cite{2} introduced Stable Paths Problem as a formal model for policy routing with path vector protocols. They proved that a sufficient condition for guaranteed convergence of BGP is the absence of dispute wheels, a structure that captures the conflicting routing policies of the ASes that are involved in the oscillations. The essence of \cite{2}, and a series of theoretical results that followed \cite{10, 11, 12}, is that the problem of routing oscillations is a fundamentally inherent property of any path-vector routing protocol that allows each node to select its highest preferred available path. Hence, by definition, any scheme that guarantees stability would require to limit the autonomy of the ASes in some form. However, the degree to which autonomy must be limited to achieve stability is an open problem.

A natural approach to avoid persistent oscillations is to restrict the set of paths an AS can select its path from. Doing this consistently across ASes (which do not share a global view of the topology) is a challenging problem. Gao and Rexford \cite{5} presented a set of restrictions on the set of paths that guarantee the convergence of BGP. While these path restrictions are captured in the structure and economics of today’s commercial Internet, violations of these assumptions due to complex business agreements and misconfigurations can still induce persistent oscillations. Another downside of such restrictions is that the policy autonomy of ASes is significantly limited: an AS can not choose a higher preferred path that is restricted, even when there are no policy conflicts in the network.

A solution to this problem is to dynamically detect persistent routing oscillations and limit autonomy only when persistent oscillations are detected in the network. Such schemes would induce restrictions on local policies at some ASes when a dispute wheel does occur \cite{6, 13, 14}. While this restricts limiting AS autonomy only when necessary, the proposed schemes require ASes to select a lower preferred path for their own traffic as well as for the transit traffic. Our solution, on the other hand, allows each AS in the network to route its traffic through its highest preferred available path, while requiring (very few) ASes to route some of the transit traffic through a lower ranked path. Also, we show using simulations on the AS graph that our scheme detects with high probability whether a network with a dispute wheel has a stable state or not, and limits no autonomy when the network does have both, a dispute wheel and a stable state. Moreover, we show through simulations that our scheme can guarantee stability while requiring use of lower-ranked paths on only an extremely small fraction of ASes: this extent of autonomy loss was not evaluated in \cite{6, 13, 14}.

Exploiting multipath routing as a tool to design stable routing protocols has been explored in at least two other solutions. Haxell and Wilfong \cite{15} introduced a fractional model of BGP, where an AS is assigned multiple paths to the same destination. They showed that every instance of a network under Fractional-BGP model is solvable. One of the issues with Fractional-BGP is that some ASes in the network may be required to be assigned exponentially many paths in order to guarantee convergence, even when each AS is involved in a single minimal dispute wheel (see Appendix A). Furthermore, the proof of \cite{15} was non-constructive and it was proved in \cite{16} that in fact the path assignment for Fractional-BGP is “hard”\cite{2}. In contrast, our scheme guarantees stability by assigning no more than a single extra path per minimal dispute wheel an AS is involved in (for a comparison, see Appendix A).

Recently, Wang et al. \cite{17} introduced a neighbor-specific routing model, NS-BGP \cite{17} that allows an AS to be assigned multiple paths, each of which is used by the AS to transit at least one of its neighbor’s traffic. It was shown that such a path-assignment protocol is inherently more stable when compared to BGP. However, despite multiple path assignments (an AS in NS-BGP may still be assigned as many paths as the number of its neighbors), NS-BGP does not guarantee stability without any restriction on the set of paths an AS can select its path from, leading to similar issues as with \cite{5}. Our algorithm SPA, on the other hand, guarantees stability and as discussed earlier, evaluation results suggest that SPA assigns just a single extra path to very few ASes in the network.

1. We will use “node” and “AS” interchangeably.
2. A PPAD-complete problem
3. Multiple Stable Paths Problem

We start by giving an example as an intuition to why multipath routing may help in stabilizing the routing process. Using this example, we discuss some of the issues with designing an efficient multipath routing solution to BGP stability problem. We then give a formal definition of Multiple Stable Paths Problem (MSPP), and summarize our results from the following sections.

Assume, for the following example, that all an AS cares about is to route its own traffic through its most preferred available path.

**Example 1:** Consider an example network from [2], the BAD GADGET, shown in Fig. 1. It is shown in Appendix B that this network has no stable state if each node insists on getting its highest ranked available path. However, if we allow some nodes in the network to be assigned multiple paths, it is not very difficult to realize that persistent oscillations in the network can be avoided. For instance, one of the path assignment using STABLE PATH(S) ASSIGNMENT for MSPP is \{130, 10\}, 210, 30, 430\}, i.e., node 1 is assigned paths \{130, 10\}, node 2 is assigned path 210 and so on. To see this, note that each of the nodes in the network is assigned its highest ranked available path to the destination (paths 3420 and 420 are not available).

![BAD GADGET](image)

**Fig. 1. BAD GADGET.** The set next to the node denotes the set of permissible paths at that node in decreasing order of ranking.

There are several important issues that need to be addressed in light of the above example. As discussed earlier, MSPP requires some ASes (node 1 in Example 1) in the network to select and advertise multiple routes. This can be regarded as limiting autonomy of an AS in the following sense: An AS advertising multiple routes to its neighbors might require to route its neighbor’s traffic through a path which is not the highest preferred path of that AS. For instance, node 1 in Example 1 routes 2’s traffic through path 1d, which is not 1’s highest preferred path. Note that, in the absence of a monetary payment, node 1 has no incentive to do that, other than stability. We remark, however, that in all the earlier schemes node 1 will route its traffic and 2’s traffic through path 1d. In this sense, our scheme for the network of Example 1 limits less autonomy than all the previous schemes.

Let us revisit the assumption made before the example. In practice, an AS cares not only about the path taken by its own traffic, but also about the path taken by the transit traffic. Hence, we require that while our scheme guarantees stability, it assigns as few paths as possible to an AS in the network so as to limit as less an autonomy as possible. We revisit this in the next section, where we give a tight upper bound on the number of paths assigned by our algorithm and also in [6] where we show that our algorithm, with extremely high probability, assigns a single extra path to ASes in the network while guaranteeing stability. We start the discussion on MSPP by describing the notation used in this paper.

3.1. Preliminaries

For sake of consistency, we follow the same notation as in [2]. The network is represented as an AS graph \( G = (V, E) \), where \( V = \{0, 1, \ldots, n\} \) is the set of ASes and each edge \( e = (u, v) \in E \) represents a BGP session between ASes \( u \) and \( v \). We assume that AS 0 is the destination AS. A path in \( G \) is either the empty path, denoted by \( e \), or a sequence of ASes, \((v_k, v_{k-1}, \ldots, v_0)\). If \( P \) and \( Q \) are paths with \( u \) and \( v \) as their last and first ASes respectively, and if \( (u, v) \in E \), we denote by \( P(u, v)Q \) the path formed by the concatenation of these paths. Similarly, we use the notation that \( eP = P e = P \).

For each \( v \in V \), let \( P_v^u \) denote the set of permitted paths from \( v \) to the destination (AS 0). For the destination AS, we assume that \( P_0^0 = \{\{0\}\} \). Let \( P = \bigcup_v P_v^u \) be the union of all sets of permitted paths. For each \( v \in V \), there is a non-negative integer valued ranking function \( \lambda^v \), defined over \( P_v^u \), which represents how \( v \) ranks its permitted paths. We assume strictness of the ranking functions, namely, for any \( AS v \) and two paths \( P_1, P_2 \in P_v^u \), \( \lambda^v(P_1) = \lambda^v(P_2) \) if and only if \( P_1 = P_2 \). Furthermore, for any two paths \( P_1, P_2 \in P_v^u \) and \( \lambda^v(P_1) < \lambda^v(P_2) \), then \( P_2 \) is said to be preferred over \( P_1 \). We assume that for each \( AS v \in V - \{0\} \), the empty path is contained in the set of permitted paths and for each path \( P \in P_v^u - e \), \( \lambda^v(P) > \lambda^v(e) \). Let \( \Lambda = \{\lambda^v|v \in V - \{0\}\} \).

An instance of the MSPP, \( I = (G, P, \Lambda) \), is an undirected graph \( G \) together with the set of permitted paths at each AS \( P \) and the ranking functions for each AS \( \Lambda \).

3.2. Stable Path Assignment

We describe the path assignment process at each AS in the network and the collective outcome of the processes at each AS: a stable path assignment.

**Definition 1 (Path Assignment):** Given an instance \( I \) of MSPP, a path assignment \( \pi : V \to P \) is a function \( \pi \) that maps each AS \( u \in V \) to a set of paths \( \pi(u) \subseteq P_v^u \). By definition, \( \pi(0) = \{\{0\}\} \).

We will generally write a path assignment as a vector of sets, \( (\pi(0), \pi(1), \ldots, \pi(n)) \), where \( \pi(u) \subseteq P_v^u \) (recall that \( \pi(0) = \{\{0\}\} \)). The set of choices that an AS \( u \in V \) gets to choose a path from are given by:

\[
\text{choices}(\pi, u) = \begin{cases} 
\{(u, v)\pi(v) | (u, v) \in E \} & \text{if } u \neq 0 \\
\emptyset & \text{otherwise}
\end{cases}
\]

3. An AS may apply its local policies to decide a subset of available paths that are allowed to be used at that AS. Such paths are referred to as permitted paths at the ASes.
Note that \( \pi(v) \) could possibly be a set of paths and hence AS \( u \) could potentially have several choices of paths that have as next hop the same neighbor \( v \). For any AS \( u \in V \), we define the best available path (denoted as \( \text{best}(\text{choices}(\pi, u)) \)) as the highest preferred path in \( \text{choices}(\pi, u) \). Note that the best “available” path depends on the path assignment \( \pi \). When the context is clear, we will simply denote the best available path for an AS \( u \) as \( P^*(u) \).

**Definition 2 (Stable Path Assignment):** Given an instance \( I \) of MSPP, a path assignment \( \pi \) is said to be stable at AS \( u \) if \( \pi(u) \) contains \( \text{best}(\text{choices}(\pi, u)) \). The path assignment \( \pi \) is stable if it is stable at each AS in the network.

Note that the autonomy issues discussed earlier are captured within the definition of MSPP stable path assignment. Indeed, the only restriction we put for stability of an instance of MSPP is that a AS has its highest ranked available path in the set of paths assigned to itself.

### 3.3. Summary of Results

We first summarize the results presented in the following sections.

- We present an algorithm **Stable Path(s) Assignment** (SPA), that produces a path assignment \( \pi \) that is a stable, as defined in Definition 2.
- SPA imposes no restriction on the set of permitted paths at any AS in the network, and assigns each AS its highest preferred available path.
- If the network has no dispute wheel, SPA assigns paths exactly as BGP would.
- If the network does have a dispute wheel, then SPA requires that some ASes support routing along paths other than their most preferred path. We give a tight upper bound on the number of paths assigned by SPA to any AS in the network in the worst-case. However, using simulations on the AS graph, we show that in 99.9% of the cases, SPA assigns no more that two paths (i.e., one extra path) to any AS in the network.
- We design a distributed protocol for SPA.

### 4. Stable Path(s) Assignment

In this section, we present **Stable Path(s) Assignment** (SPA), a centralized algorithm which when given an instance \( I = (G, P, \Lambda) \) of MSPP, assigns paths to each AS in the network in such a way that the final path assignment corresponds to a stable path assignment of MSPP. We also introduce a graph theoretic structure dispute graph and show that a directed cycle in a dispute graph corresponds to the possibility of persistent routing oscillations in the network. We then exploit this observation to prove proofs of correctness of SPA. We close the section with some properties of SPA including a tight bound on the maximum number of paths assigned by SPA to any AS in the network.

To start with, we give some definitions which will allow us to succinctly describe the algorithm.

**Definition 3 (Partial Path Assignment):** A partial path assignment \( \pi = \{(0), \pi(1), \pi(2), \ldots, \pi(n)\} \) for \( V' \subseteq V \) is a path assignment such that for every \( u \in V' \), every AS in \( \pi(u) \) is in \( V' \).

**Definition 4 (Consistent Path):** For an AS \( u \), a path \( P = (u, v_1, v_2, \ldots, v_k, 0) \in P^u \) is said to be consistent with path assignment \( \pi \) if for \( u \) and for each \( v_i \in P \), the subpath \( P_i = (v_i, v_{i+1}, \ldots, v_k, 0) \) is either in \( \pi(v_i) \) or is ranked higher than each of the paths in \( \pi(v_i) \). If \( P \in \pi(v_i) \), then \( P \) is said to be a direct path, else we call it an indirect path.

**Example 2 (Example 1 continued):** Consider again the network **BAD GADGET** shown in Fig. 1 and a run of BGP, as shown in Appendix C (Table 2). In Table 2 consider the path assignment of iteration 7, where we denote as \( \pi = \{0, 1, 20, 30, 430\} \). First of all, note that for \( V' = \{0, 1, 2, 3\} \), the path assignment can be called a partial path assignment. However, for \( V' = \{0, 1, 2\} \), the path assignment is not a partial path assignment. Consider node 1, that has \( \pi(1) = 130 \). We claim that path 1d is inconsistent with the path assignment \( \pi \). This is due to the fact that \( \lambda_1(10) < \lambda^1(\pi(1)) \). Next consider nodes 4 and 3. The claim is that path 420 is a direct path for node 4 that is consistent with \( \pi \), while the path 3420 is an indirect path for node 3 that is consistent with \( \pi \). We leave it as an exercise to confirm the above claims.

It is easy to see and particularly important to note that for two partial path assignments, \( \pi_1 \) and \( \pi_2 \), if for each AS \( u \), \( \pi_1(u) \subseteq \pi_2(u) \) and if a path \( P \) is not consistent with \( \pi_1 \), then \( P \) cannot be consistent with \( \pi_2 \).

**Definition 5 (Stable and Unstable Sets):** The stable set \( S \) is defined to be the set of ASes \( u \in V \), for which the highest ranked path consistent with \( \pi \) is already assigned to \( u \). The unstable set \( \bar{S} \) is the set of ASes that are not in the stable set.

**Definition 6 (Dispute Wheel):** A dispute wheel \( W = \{u_1, u_2, \ldots, u_k\}, (Q_1, Q_2, \ldots, Q_k) \} \) is a sequence of nodes \( u_1, u_2, \ldots, u_k \), a sequence of non-empty paths \( Q_1, Q_2, \ldots, Q_k \) and a sequence of paths \( R_1, R_2, \ldots, R_k \) such that for each \( 1 \leq i \leq k \), we have (1) \( R_i \) is a path from \( u_i \) to \( u_{i+1} \), (2) \( Q_i \in P^u \), (3) \( R_{i+1} \in P^u \), and \( \lambda^{u_i}(Q_i) < \lambda^{u_i}(R_{i+1}) \) (all subscripts to be taken modulo \( k \)).

**Definition 7 (Minimal Dispute Wheel):** A minimal dispute wheel \( W \) is a dispute wheel in which for each \( 1 \leq i \leq k \), either \( R_iR_{i+1}Q_{i+2} \) is not permitted at \( u_i \) or \( \lambda^{u_i}(R_iR_{i+1}Q_{i+2}) < \lambda^{u_i}(R_iQ_{i+1}) \).

An example that illustrates the idea of dispute wheels and minimal dispute wheels is given in Appendix D.

### 4.1. Stable Path(s) Assignment Algorithm

We present **Stable Path(s) Assignment** (SPA) algorithm. To give an intuition for the underlying functioning of the algorithm, we first consider a simpler case, when the instance \( I \) of MSPP contains a single minimal dispute wheel. We then give a simple generalization to the case of network having multiple minimal dispute wheels.
The algorithm (for the case of single minimal dispute wheel) runs in two phases: in the first phase INITIAL PATH ASSIGNMENT, each AS is assigned a single path to the destination (some ASes may be assigned empty paths); in the second phase STABLE PATH(S) ASSIGNMENT – SMDW (SMDW for single minimal dispute wheel), each AS (that is in the unstable set) is assigned its highest ranked path consistent with the current path assignment. Both the phases assign paths in a greedy fashion, first assigning paths to the ASes whose highest ranked path consistent with current path assignment is a direct path. If no such AS is available, the algorithm will arbitrarily choose an AS that has a direct path to the destination and assign this path to the AS.

The first phase INITIAL PATH ASSIGNMENT, constructs a sequence of subsets of \( V \), \( \{0\} = V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_k = V \), together with a sequence of partial path assignments \( \pi_0, \pi_1, \ldots, \pi_k \), where each \( \pi_i \) is a partial path assignment for \( V_i \). Let \( D_i \) be the set of ASes \( u \in V - V_i \) that have a direct path consistent with \( \pi_i \). Let \( S_i \subseteq D_i \) be the set of ASes whose highest ranked path consistent with \( \pi_i \) (i.e., best(\text{choices}(\pi_i, u))) is a direct path.

The algorithms INITIAL PATH ASSIGNMENT and STABLE PATH(S) ASSIGNMENT – SMDW are given in Fig. 2 and Fig. 3 respectively.

Example 3: Consider again the BAD GADGET shown in Fig. 1 Table II gives the step-wise path assignment when the algorithm STABLE PATH(S) ASSIGNMENT – SMDW is run on the BAD GADGET. It is easy to see that the final path assignment using the algorithm, \( \pi = \{0, \{10, 130\}, 210, 30, 430\} \), is indeed a stable path assignment for MSPP.

In the following subsections, we prove correctness of the algorithm, give a simple generalization to the case of a network having multiple minimal dispute wheels and discuss some of the properties of SPA solutions.

4.2. Correctness of SPA

In order to prove the correctness of the algorithm, we define dispute graph, a directed graph that captures the set of conflicting rankings between ASes in a given instance of MSPP. We show that edge-disjoint cycles in a dispute graph correspond to distinct minimal dispute wheels in a single dispute wheel and exploit this to give a simple proof for the correctness of the algorithm.

4.2.1. Dispute Graph. Given an instance \( \mathcal{I} = (G, \mathcal{P}, \Lambda) \) and a path assignment \( \pi \), a dispute graph corresponding to \( \pi \) is
a directed graph $G(I, \pi)$ constructed as follows: For each node $u \in V \setminus d$, we create a node in the dispute graph. Let $P^*(u) = (u, v_1, v_2, \ldots, v_k, 0)$ denote the highest ranked path of $u$ consistent with $\pi$. For each intermediate node $v_i$, we create an edge $(u \ v_i)$ in the dispute graph.

**Example 4:** Consider again the network of Fig. 1. The dispute graph for the network is as shown in Fig. 4. Note that the dispute graph has two cycles $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ and $1 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

**Lemma 1:** Let $I = (G, P, \Lambda)$ be an instance of MSPP and let $G(I, \pi)$ be the dispute graph for $I$ corresponding to a path assignment $\pi$. Then if $G(I, \pi)$ has $k$ edge-disjoint cycles, $I$ has at least $k$ minimal dispute wheels.

**Proof:** We show that each edge-disjoint cycle corresponds to a minimal dispute wheel in the network. Consider a cycle $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_k \rightarrow u_1$ in $G(I)$. Then, by definition of a dispute graph, for any node $u_i$ in the cycle, $u_i+1$ must be in the highest rated path of $u_i$. This implies that one can construct a minimal dispute wheel as follows: Let $P^*(u_i)$ denote the highest rated path of $u_i$ and let $R_i$ be the subpath of $P^*(u_i)$ from $u_i$ to $u_{i+1}$ and let $Q_i$ be the remaining subpath of $P^*(u_i)$. This will give a minimal dispute wheel $W = \{u_1, u_2, \ldots, u_k\}, \{Q_2, Q_2, \ldots, Q_k\}, \{R_2, R_2, \ldots, R_k\}\). \[\Box\]

**4.2.2. Correctness of SPA.** We now formally prove the correctness of the algorithm.

**Lemma 2:** Let $U$ denote the unstable set after INITIAL PATH ASSIGNMENT and let $W$ be a dispute wheel in $U$. For each $u \in W$, let $P^*(u)$ denote the highest ranked path of $u$ consistent with $\pi$. Then, $P^*(u)$ must contain at least 4 nodes from $V$.

**Proof:** Consider any node $u \in W$ and let $P^*(u)$ be its highest rated path. Note that since $u$ is in a dispute wheel $P^*(u)$ must not be a direct path for $u$. Clearly, $P^*(u)$ must contain at least two nodes $u$ and 0. If $\pi(u) = \epsilon$, then $(u 0) \notin P^u$ and if $\pi(u) = \epsilon$, then $\lambda^u(\pi(u)) \geq (u 0)$, hence $P^*(u)$ must contain at least three nodes. We need to show that if $u \in W$, $P^*(u)$ must contain at least four nodes. For sake of contradiction, assume that it contains only three nodes. Then, $P^*(u) = uu'0$ for some $u' \in V$. Clearly, $(u' 0) \in \pi(u')$ since $P^*(u)$ is consistent with $\pi$. But then, $P^*(u)$ is a direct path for $u$ leading to the contradiction that $u \in W$. Hence, $P^*(u)$ must have at least 4 nodes.

**Lemma 3:** If $G$ is a $k$-regular directed graph with no parallel edges, then $G$ contains at least $5k/2 - 2$ edge-disjoint cycles.

**Theorem 1:** The algorithm STABLE PATH(S) ASSIGNMENT − SMDW solves all instances $I = (G, P, \Lambda)$ of MSPP if $I$ does not contain multiple minimal dispute wheels.

**Proof:** Consider the first iteration of STABLE PATH(S) ASSIGNMENT. If there is no dispute wheel in $U$, then clearly each node is assigned a stable path. However, if there is a dispute wheel in $U$, then we claim that there must be multiple minimal dispute wheels in the network.

To prove this, let $I = (G, P, \Lambda)$ be an instance of MSPP and let $G(I, \pi)$ be the dispute graph for $I$ corresponding to current path assignment. Let $W$ be a dispute wheel in the unstable set $U$ after INITIAL PATH ASSIGNMENT. By Lemma 2 the highest ranked path of each node in $W$ must have at least four nodes, out of which two nodes must be different from $u$ and the destination node. Hence, by construction, each node in $G(I, \pi)$ must have an out-degree of at least 2. Then, by Lemma 3 there must be at least three edge-disjoint cycles in $G(I, \pi)$. This in turn, by Lemma 1 means that there must be multiple minimal dispute wheels in $I$. \[\Box\]

**4.3. Case of Multiple Minimal Dispute Wheels**

When there are multiple minimal dispute wheels in $I$, one might need to assign more than two paths to a single node. This can be achieved by running algorithm INITIAL PATH ASSIGNMENT every time a dispute wheel is found in the unstable set $U$. Each iteration of INITIAL PATH ASSIGNMENT will result in breaking at least one minimal dispute wheel. However, we believe (and show using evaluation results) that for most of the networks, multiple minimal dispute wheel case

**TABLE 1. Iterations of STABLE PATH(S) ASSIGNMENT − SMDW for network of Fig. 1**

<table>
<thead>
<tr>
<th>iteration</th>
<th>$V_i$</th>
<th>$\pi_i$</th>
<th>$D_i$</th>
<th>$S_i$</th>
<th>$U$</th>
<th>$U_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0}</td>
<td>{0, 0, 0}</td>
<td>{1, 2, 3}</td>
<td>{}</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>{0, 1}</td>
<td>{0, 0, 0}</td>
<td>{2, 3}</td>
<td>{}</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>2</td>
<td>{0, 1, 2}</td>
<td>{0, 10, 1, -}</td>
<td>{3}</td>
<td>{}</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>{0, 1, 2, 3}</td>
<td>{0, 10, 210, 30}</td>
<td>{4}</td>
<td>{}</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>4</td>
<td>{0, 1, 2, 3, 4}</td>
<td>{0, 10, 210, 30, 430}</td>
<td>{}</td>
<td>{}</td>
<td>{1}</td>
<td>{}</td>
</tr>
<tr>
<td>5</td>
<td>{0, 1, 2, 3, 4}</td>
<td>{0, 10, 210, 30, 430}</td>
<td>-</td>
<td>-</td>
<td>{}</td>
<td>{1}</td>
</tr>
<tr>
<td>6</td>
<td>{0, 1, 2, 3, 4}</td>
<td>{0, 10, 210, 30, 430}</td>
<td>-</td>
<td>-</td>
<td>{1}</td>
<td>{}</td>
</tr>
</tbody>
</table>
is also resolved in a single run of INITIAL PATH ASSIGNMENT. The pseudo-code for STABLE PATH(S) ASSIGNMENT algorithm is shown in Fig. 5.

**Theorem 2:** The algorithm STABLE PATH(S) ASSIGNMENT-MMDW solves all instances \((G, \mathcal{P}, \Lambda)\) of MSPP.

### 4.4. Properties of SPA

In this section, we comment on some of the features of solutions of MSPP.

**Claim 1:** In the absence of a dispute wheel, STABLE PATH(S) ASSIGNMENT assigns a single path to each node. Moreover, the paths assigned correspond to the paths assigned by BGP.

**Proof:** First we note that a network that does not have a dispute wheel has a unique stable state \([2]\). Hence, any algorithm for stable path assignment that assigns a single path to each node will converge to the unique stable state. Then, it suffices to show that no node is assigned more than one path for such a network. To see this, note that a node is assigned multiple paths only if \(S_i = \emptyset\) for some \(i\) in INITIAL PATH ASSIGNMENT. If \(S_i \neq \emptyset\) for any \(i\), all nodes are assigned a single path. The claim follows by noting that \(S_i = \emptyset\) if and only if there is a dispute wheel in the network \([2]\).

The above claim implies that in the absence of a dispute wheel, STABLE PATH(S) ASSIGNMENT assigns exactly those paths that BGP would assign. If the network does have a dispute wheel, the number of paths assigned are given by the following two claims:

**Claim 2:** No node in (a single minimal dispute wheel of) a network is assigned more than two paths by STABLE PATH(S) ASSIGNMENT – SMDW.

**Proof:** To prove this, we note that INITIAL PATH ASSIGNMENT results in a partial path assignment \(\pi\) in which every node is assigned exactly one path. However, some of these nodes might be in the unstable set \(U\). Hence, it suffices to show that no node in \(U\) is assigned more than one path in addition to the path assigned by INITIAL PATH ASSIGNMENT. This is easy to see, since every node in \(U\) is assigned another path only if it is included in set \(U_j\) for some \(j\). Also, at this step, at most one path is assigned to every node in \(U_j\), which is precisely the highest ranked path for the node that is consistent with the partial path assignment \(\pi\). The claim follows by noticing that once a node in \(U\) is included in the set \(U_j\) for some \(j\), it is discarded from the set \(U\) and can not be assigned any more paths.

Finally, we give a tight bound on the number of paths assigned by STABLE PATH(S) ASSIGNMENT.

**Claim 3:** Given an instance \(I = (G, \mathcal{P}, \Lambda)\) of MSPP, let \(m(u)\) denote the number of minimal dispute wheels that an AS \(u\) is in. Then STABLE PATH(S) ASSIGNMENT assigns no more than \(m(u) + 1\) paths to \(u\). This is tight, in that there are instances in which at least one AS \(u \in G\) is assigned exactly \(m(u) + 1\) paths.

**Proof:** Please see Appendix D.

We make two observations. First, the bound of Claim 3 is valid for any multipath assignment algorithm that guarantees stability and that operates in a distributed fashion, in the sense that it does not restrict the order in which nodes in dispute wheel are assigned paths. Second, the number of paths assigned in the worst case could be as many as the number of nodes in the dispute wheel. Whether or not it is large is an interesting question; however, we show in \([6]\) that in almost all the cases, just one extra path suffices to guarantee stability. More so, our evaluation results suggest that very few ASes in the network (0.04% of the ASes) are required to be assigned an extra path to stabilize the network.

### 5. Distributed Stable Path(s) Assignment

Previous sections have described our overall approach. However, thus far we have assumed that each AS in the network can observe the entire graph. In this section, we describe some simple extensions to our technique to allow it to operate as a dynamic routing protocol in distributed environments. For simplicity, we concentrate only on sketching the main ideas of our approach, relying on previous work in

---

**Fig. 5. Pseudo-code for STABLE PATH(S) ASSIGNMENT**

```
STABLE PATH(S) ASSIGNMENT:
While \(S \neq V\):
    Run INITIAL PATH ASSIGNMENT with \(V = U\) to compute \(\pi\) and \(S\).
    Let \(U = V - S\).
    Initialize \(j\) to 0.
    Let \(U_0 \subseteq U\) be the set of nodes whose highest ranked path consistent with \(\pi\) is a direct path.
    While \(|U_j| \neq 0\):
        - Assign each node in \(U_j\) its highest ranked path consistent with \(\pi\).
        - \(U = U - U_j\).
        - Compute \(U_{j+1}\): the set of nodes whose highest ranked path consistent with \(\pi\) is a direct path.
        - \(j = j + 1\).
    Let \(S = V - U\).
```
detecting dispute wheels [13, 6], multipath routing [8, 19], and BGP implementations [20] to handle details not discussed here.

The distributed protocol is shown in Fig. 5. The algorithm consists of two key steps.

5.1. Dispute detection

First, each AS continuously checks to determine if it part of a dispute wheel. It does this by performing a consistency check whenever it receives a new routing update. We do this by adapting the SPVP formulation given in [13], which works by maintaining a route history and detecting cycles in the history. Using this technique, the AS can determine whether it is currently in dispute with other ASes (while it may also be possible to detect the number of ASes in the dispute and even which ASes are in dispute, a naive implementation of our scheme does not require any of this information).

5.2. Handling disputes

Once an AS has detected its involvement in a dispute, the protocol places a restriction on the set of paths assigned to the AS. The restriction disallows withdrawals, of any path within this set, that are caused by preference changes. When an AS processes a new (and higher preferred) path, it selects this path and forwards it as an additional path to break the dispute. In particular, when the restriction is enabled, the AS marks the current path that is advertised. The AS then additionally marks any new more-preferred paths that are received after the restriction is enabled. The AS then advertises the set of these paths to its neighbor. Note that this procedure does not affect how physical failures and other withdrawals are handled (see below)—only path changes due to receiving a more-preferred path are affected. On a failure (or a path withdrawn by the neighbor), the AS would receive a withdrawal message, causing the path to be unmarked and deleted from its own routing table (and subsequently deleted from the set advertised to the AS’s neighbors).

Note that the description above handles the network dynamics. Indeed, during the convergence process, an AS u that has detected its presence in a dispute wheel may either receive a path advertisement or a path withdrawal. A path advertisement does not effect u’s adhering to the restrictions imposed in order to handle the disputes; the path advertisement received by an AS under restriction will not cause path withdrawals. An explicit path withdrawal, on the other hand, may be due to a change in the topology of the underlying network or due to one of u’s neighbors, say AS v, withdrawing the path being advertised by u to its neighbors. Consider AS v. Either v is in the dispute wheel or not. If not, then the path withdrawal by AS v adheres to the handling dispute restriction (recall, the handling disputes above puts restriction only when an AS is in dispute wheel). If v is indeed in dispute wheel, at some point of time v will detect its presence in the dispute wheel and at that point of time, both u and v will stop withdrawing the paths. Following this process, at some point, all the ASes in the dispute wheel will stop withdrawing the paths and all the advertisements will eventually be consumed by the ASes in the network, resulting in the network reaching a stable state.

6. Evaluation Results

In this section, we report on our implementation of STABLE PATH(S) ASSIGNMENT and its evaluation on an AS-level network topology. We discuss below the methodology used for evaluating the performance of the algorithm, and report on three main results:

- The effect of misconfigurations on routing instability
- A surprising aspect of STABLE PATH(S) ASSIGNMENT: detection of a large fraction of naughty gadgets
- The amount of autonomy that STABLE PATH(S) ASSIGNMENT limits at each AS: in particular, the number of paths assigned by the protocol on an average.

6.1. Methodology

To evaluate the performance of STABLE PATH(S) ASSIGNMENT, we use an AS-level network topology from CAIDA [21]. Route dumps from January 2009 were used to construct an AS-level network which consists of 24113 ASes. Note that the performance of STABLE PATH(S) ASSIGNMENT depends heavily on the set of paths available to an AS and the set of preferences that the AS sets over this set of available paths.

In order to generate a list of paths at each AS, we implemented a policy-free path-vector protocol in which each
AS advertises at most 500 paths to a particular neighbor. Our implementation restricted each AS to advertise at most 5 paths of length greater than 20, while advertising all the paths of length less than 20. We believe this is not restrictive, since an AS has a path through each neighbor (which allowed us to implement next-hop based policies) and sufficiently many paths of varying lengths so as to give preference to the shorter paths through the same neighbor.

Our first metric of interest was the intensity of the routing instability problem. In particular, the effect of misconfigurations on the routing instability has not been evaluated in earlier works. Note that it is NP-hard to decide whether a given instance of a network has a stable state or not.

Our results suggest that misconfigurations could drastically affect the routing stability of the network. In particular, even with 0.05\% of the ASes misconfigured (10 out of 24,113 ASes), more than 5\% of the instances (out of 10,000 instances) had dispute wheels. Furthermore, the percentage of networks that contained dispute wheels increased monotonically with the number of misconfigured ASes. The variation of the fraction of instances that contained dispute wheels with the number of misconfigured ASes is shown in Fig. 7.

### 6.3. Bad and Naughty Gadgets

One surprising result that our evaluation results suggest is that STABLE PATH(S) ASSIGNMENT can detect Naughty Gadgets with a very high probability. To the best of our knowledge, this is not possible in any of the schemes proposed earlier. This allowed the network to converge with each AS having been assigned a single path as in BGP, despite the presence of dispute wheels in the network. Fig. 8 shows the fraction of instances (among instances with dispute wheel) that were detected as naughty gadgets by STABLE PATH(S) ASSIGNMENT. Note that even for large number of misconfigured ASes, more than 91\% of the networks that have a dispute wheel were detected to be naughty gadgets. We believe that the rest of the networks are indeed bad gadgets, which require to be assigned multiple paths in order to stabilize the network.

We emphasize that it is not clear how the schemes proposed earlier would detect a naughty gadget. We believe that these schemes will limit the autonomy of ASes even if the network is a naughty gadget, while STABLE PATH(S) ASSIGNMENT limits no autonomy in such networks, which happen to be a large fraction (91\% of all the networks that have a dispute wheel) of potentially unstable networks.

### 6.4. Number of Paths

Finally, we study our main performance metric for STABLE PATH(S) ASSIGNMENT: the number of paths assigned to ASes in the network. Note that the result of Claim 3 states that the number of paths assigned to an AS by STABLE PATH(S) ASSIGNMENT could be as large as the number of ASes in the dispute wheel. However, in (almost) all the instances, STABLE PATH(S) ASSIGNMENT stabilized the network with just a single extra path assigned to (very) few ASes in the network. In particular, in 99.9\% of the instances that had a dispute wheel, even with large number of misconfigured ASes, STABLE PATH(S) ASSIGNMENT assigned a single extra path to any AS in the network (see Fig. 10). Furthermore, the number of ASes that were assigned this extra path were an extremely
small fraction of the total number of ASes; in more than 99% of the networks in which any AS was assigned an extra path, at most 0.04% of the ASes were assigned an extra path. This resulted in extremely small number of paths assigned on an average (averaged over all the ASes in the network), as shown in the scatter plot of Fig. 9. Finally, for the networks in which the algorithm assigned more than two paths to any AS in the network, almost 97.5% of the instances were assigned at most three paths. The remaining instances were some of the unlucky instances, in which the algorithm ended up assigning four paths to some ASes in the network.

In all the 300,000 iterations (over 30 values of number of misconfigured ASes), not even a single AS (in any instance) was assigned more than four paths.

7. Discussion

In this paper, we present a solution towards resolving the conflict between policy autonomy of Autonomous Systems in the Internet and stability of the inter-domain routing protocol BGP. The key tool we leverage is the use of multipath assignment to nodes in conflict. In this direction, we have presented an algorithm that computes a stable path assignment for any given network while ensuring that nodes in conflict are not assigned too many paths. We start by analyzing a centralized version of this algorithm, then give a distributed version that performs dynamic path assignment across routers. Through simulations, we show that SPA assigns at most two paths to nodes in the network with high probability while guaranteeing stability of the network.

References

that this network does not have a stable state. The iteration s

\textbf{Appendix A}

\textbf{Comparison to Fractional-BGP}

Consider the network of Fig. 12 which uses \( n \) copies three-node \textbf{BAD GADGET} from [2] (we will call each such copy a gadget, the copy with center \( d_i \) as gadget \( \mathcal{G}_i \)). Each of the node in the network is trying to establish a path to destination node \( d_1 \). Let \( P_1 = \epsilon \) and for each \( i \geq 2 \), define \( P_i \) to be \( a_{i-1}d_{i-1}a_{i-2}d_{i-2} \ldots a_1d_1 \). Then, the preference list in gadget \( \mathcal{G}_i \) is given as follows: the preference list for node \( a_i \) is \( a_i d_i P_i \); for node \( b_i \) is \( b_i c_i d_i P_i \); for node \( c_i \) is \( c_i a_i d_i P_i \); and for node \( d_i \) is \( d_i P_i \). This network was used as an example network in [15] to demonstrate that not every solution for fractional model of BGP is half-integral; indeed, for this network the number of paths assigned in Fractional-BGP in the worst-case are exponential. Note that there are \( n \) node-disjoint minimal dispute wheels in the network. We show that using \textbf{STABLE PATH ASSIGNMENT} results in no node being assigned more than two paths in a stable path assignment. We choose the \( \sigma(\cdot) \) function so that each node has a path to the destination. It is easy to see that using \textbf{STABLE PATH ASSIGNMENT}, the paths assigned to gadget \( \mathcal{G}_i \) is as follows: node \( a_i \) is assigned both the paths in its preference list, i.e., \( \pi(a_i) = \{a_i d_i P_i, a_i d_i P_i\} \); node \( b_i \) is assigned \( \pi(b_i) = \{b_i d_i P_i\} \) and node \( c_i \) is assigned path \( \pi(c_i) = \{c_i a_i d_i P_i\} \). It is easy to see that this is a stable state for MSPP.

\textbf{Appendix B}

\textbf{Description of Example [1]}

Consider the network shown in Fig. 11. It was shown in [2] that this network does not have a stable state. The iterations for path assignment using BGP are shown in Table 2. Consider a state where nodes are assigned paths \{10, 20, 3420, 420\} as shown in the first line of the table. A node that changes its path in the next atomic step is underlined. Note that a node may change its path either if it has a higher ranked path that is available to the node (for example, node 2 in iteration 1) or its earlier path is no more available (for example, node 4 in iteration 2). Notice that iteration 1 and iteration 10 are identical, resulting in the fact that the network might oscillate forever.

\begin{table}[h]
\centering
\caption{Routing Oscillations in network of Fig. 11}
\begin{tabular}{|c|c|c|c|c|}
\hline
Iteration & Node 1 & Node 2 & Node 3 & Node 4 \\
\hline
1 & 10 & 20 & 3420 & 420 \* \\
2 & 10 & 210 & 3420 & 420 \\
3 & 10 & 210 & 3420 & 420 \\
4 & 10 & 210 & 30 & 420 \\
5 & 10 & 210 & 30 & 430 \\
6 & 130 & 210 & 30 & 430 \\
7 & 130 & 20 & 30 & 430 \\
8 & 130 & 20 & 30 & 420 \\
9 & 130 & 20 & 3420 & 420 \\
10 & 10 & 20 & 3420 & 420 \* \\
\hline
\end{tabular}
\end{table}

\textbf{Appendix C}

\textbf{Dispute Wheel and Minimal Dispute Wheels}

Consider the network shown in Fig. 13 which is slightly modified version of \textbf{BAD GADGET}. In particular, we have added link 40 to the set of permissible paths at node 4 and made this least ranked path for node 4 and added a path 340 to node 3 which is ranked higher than 30 but lesser than 3420. In this network, there is a dispute wheel.
\{\{1, 3, 4, 2\}, \{10, 30, 40, 20\}, \{13, 34, 42, 21\}\}. We claim that this is not a minimal dispute wheel. To see this, note that for node 3, the condition for minimality of a dispute wheel does not satisfy. Indeed, one of the minimal dispute wheels in the network is \{\{1, 3, 2\}, \{10, 30, 20\}, \{13, 342, 21\}\}.

Appendix D
Proof of Claim 3

The first part of the claim is proved by noticing that an AS can not be in a dispute wheel after \(m(u)\) runs of INITIAL PATH ASSIGNMENT (using arguments similar to the proof of Theorem 1).

To prove the tightness, we construct an instance of MSPP as follows: let \(G = (V, E)\), where \(V = \{u_0, u_1, u_2, \ldots, u_k\}\) with \(u_0\) as the destination and \(E = \{(u_i, u_{i+1}) : 1 \leq i \leq k\} \cup \{(u_i, 0) : 1 \leq i \leq k\}\). The set of permissible paths and the rankings are given as follows: the \(j\)-th ranked path (0 denoting the highest preferred path) of AS \(u_i\) is \((u_i, u_{i+1}, \ldots, u_{k-j}, u_0)\) (all indices to be taken modulo \(k\)). A sample network with 5 ASes is shown in Fig. 14.

![Fig. 14. A network with five nodes. The rankings for node 1 is given by \(\{12340, 1230, 120, 10\}\), rankings for node 2 is given by \(\{23450, 2340, 230, 20\}\), node 3 is \(\{34510, 3450, 340, 30\}\) and similarly for nodes 4 and 5. STABLE PATH(S) ASSIGNMENT assigns at least \(m(1) + 1 = 4\) paths to node 1 in the worst-case.](image)

Now, consider the following order in which ASes are assigned paths (in each run): 1, 2, \ldots, \(k\). It is easy to see that in each run of INITIAL PATH ASSIGNMENT, AS 1 will have its \(k - 1 - j\)-th path available. When the algorithm converges, AS 1 is assigned exactly \(k - 1\) paths. The claim is settled by noting that AS 1 forms a minimal dispute wheel with \(k - 2\) nodes: \(\{2, 3, \ldots, k - 1\}\).