An Analysis of BGP Convergence Properties

Timothy G. Griffin Gordon Wilfong
griffin@research.bell-labs.com, gtw@research.bell-labs.com
Bell Laboratories, Lucent Technologies
600 Mountain Avenue, Murray Hill NJ

Abstract

The Border Gateway Protocol (BGP) is the *de facto* interdomain routing protocol used to exchange reachability information between Autonomous Systems in the global Internet. BGP is a path-vector protocol that allows each Autonomous System to override distance-based metrics with policy-based metrics when choosing best routes. Varadhan *et al.* [18] have shown that it is possible for a group of Autonomous Systems to independently define BGP policies that together lead to BGP protocol oscillations that never converge on a stable routing. One approach to addressing this problem is based on *static analysis* of routing policies to determine if they are safe. We explore the worst-case complexity for convergence-oriented static analysis of BGP routing policies. We present an abstract model of BGP and use it to define several global sanity conditions on routing policies that are related to BGP convergence/divergence. For each condition we show that the complexity of statically checking it is either NP-complete or NP-hard.

1 Introduction

Dynamic routing protocols for IP come in two basic flavors. *Interior Gateway Protocols* (IGPs) are used for routing within Autonomous Systems (ASes) while *Exterior Gateway Protocols* (EGPs) are used for global routing between ASes [15, 9]. Currently, there is only one EGP in use — the Border Gateway Protocol (BGP) [16, 7, 17]. BGP is a path-vector protocol, which is a type of distance-vector protocol where best route selection for an Autonomous System is a function of its routing policies and the best routes of its neighbors. BGP allows routing policies to override distance-based metrics with policy-based metrics.

While the routing policies of each individual AS may be locally reasonable, there is no guarantee that the interaction of independently defined policies will be globally reasonable. We are not referring to the misconfiguration of BGP. The BGP policies are currently implemented locally with little global knowledge. A collection of locally *well-configured* policies can still give rise to global routing anomalies.

While most routing policy conflicts are of a manageable nature, with BGP there is the possibility that they could lead the protocol to *diverge*. That is, such inconsistencies could cause a collection of ASes to exchange BGP routing messages indefinitely without ever converging on a set of stable routes. While pure distance-vector protocols such as RIP [8] are guaranteed to converge, the same is not true for BGP. The proof of convergence for RIP-like protocols (see for example [3]) relies on the monotonicity of the distance metric. Since BGP allows individual AS routing policies to override the shortest path choice, this proof technique cannot be extended to BGP. Indeed, Varadhan *et al.* [18] have shown that there are routing policies that can cause BGP to diverge.

Is it possible to guarantee that BGP will not diverge? We call this the *BGP convergence problem*. BGP divergence could introduce a large amount of instability into the global routing system. Several studies [6, 12, 14, 13] have examined the dynamic behavior of interdomain routing and have highlighted the negative impact of unstable routes. However, we are not aware of any instance where routing instability has been caused by protocol divergence, and it is impossible to say if divergent BGP systems will arise in practice. On the other hand, given the economic importance of the Internet, we believe that it is worthwhile to consider worst-case scenarios and to provide safeguards where possible.

Broadly speaking, the BGP convergence problem can be addressed either *dynamically* or *statically*. A *dynamic* solution to the BGP divergence problem is a mechanism to suppress or completely prevent at "run time" those BGP oscillations that arise from policy conflicts. Using route flap dampening [19] as a dynamic mechanism to address the BGP convergence problem has two distinct drawbacks. First, route flap dampening cannot eliminate BGP protocol oscillations, it will only make these oscillations run in "slow motion". Second, route flapping events do not provide network administrators with enough information to identify the source of the route flapping. Route flapping caused by policy conflicts will look the same as route flapping caused by unstable routers or defective network interfaces. In Section 6 we consider what is required of a dynamic solution.

A static solution is one that relies on programs to analyze routing policies to verify that they do not contain policy conflicts that could lead to protocol divergence. This is essentially the approach advocated by the Route Arbiter Project as described in Govindan *et al.* [5]. This project has three components. First, the Routing Policy Specification Language (RPSL)[1] is a high-level vendor-independent language for specifying interdomain routing policies. Second, Internet

Route Registries (IRRs) [10] are used to store and distribute RPSL specifications. Third, a collection of software tools, called the RAToolSet [2], gives network administrators the ability to manipulate and analyze RPSL specifications that have been stored in the IRR. For example, the tool `RtConfig` generates low-level router configuration files from high-level RPSL specifications.

In this paper, we explore the worst-case complexity for convergence-oriented static analysis of BGP routing policies. We present an abstract model of BGP and use it to define several conditions on routing policies that are related to BGP convergence/divergence. For each condition we show that the complexity of statically checking it is either NP-complete or NP-hard. These results suggest that the static analysis approach to the BGP convergence problem may not be a practical one.

Given that routing policies may not be publicly available, if unsolvable BGP policies are causing route-flapping, then the "source" of the problem may be difficult to locate and may require a high degree of inter-AS cooperation to debug. This argues in favor of using a high-level specification language, such as RPSL [1], for interdomain routing policies. High-level policy specifications could be easily shared, at least when the need arises.

Paper outline. In Section 2 we introduce an *abstract BGP* to provide a framework in which we can formalize our complexity results. Abstract BGP is simpler that the real-world protocol. For example, we ignore address aggregation. The model is based on an *evaluation graph* that captures all possible asynchronous executions of the abstract BGP protocol. If this graph contains a state with a stable routing, then the system has a *solution* and the corresponding set of routing policies is *solvable*. Otherwise, the set of policies is *unsolvable*, which means that the protocol can never converge.

In Section 3 we present several examples of BGP systems that illustrate distinct types of routing anomalies. The system called BAD GADGET is unsolvable, and is similar in spirit to examples of [18]. Operationally, BAD GADGET causes a divergence of the BGP protocol. The system SURPRISE is *solvable* and will always converge on a stable routing. However, a single link failure is enough to transform SURPRISE into one that will never converge. This shows that solvability is not *robust* under link failure. The system DISAGREE demonstrates the fact that BGP systems can have multiple distinct solutions and that a solvable system does not *always* converge on a solution. However, the type of divergence possible with DISAGREE might be called *weak divergence* (in contrast to the *strong divergence* of BAD GADGET) since it is possible to exit the "evaluation cycle" and arrive at a solution. Operationally, DISAGREE could give rise to a *transient* route oscillations that are unlikely to persist. Finally, the system PRECARIOUS shows that a BGP system may have a solution *and* contain a "trap" that leads to strong divergence and persistent route oscillations.

The complexity results are presented in Section 4. Among the problems we consider are

REACHABILITY Given a BGP system, will AS X be able to import routes originated from AS Y?

ASYMMETRY Does a BGP system allow an asymmetric routing?

SOLVABILITY Does a given BGP system have a solution?

SINGLE DESTINATION SOLVABILITY Does a given BGP system with a single destination originated by a single AS have a solution?

UNIQUENESS Does a given BGP system have a unique solution?

ROBUSTNESS Given a solvable BGP system, will it remain so after any possible failure of k links?

We show that SINGLE DESTINATION SOLVABILITY, ASYMMETRY, and REACHABILITY are NP-complete, while SOLVABILITY, UNIQUENESS, and ROBUSTNESS are NP-hard.

In Section 5 we discuss how abstract BGP of Section 2 and the complexity results of Section 4 carry over to the more complex *real-world* BGP. We argue that the complexity results for each question provide a *lower bound* on the complexity of answering this questions for real-world BGP policies.

2 An abstract model of BGP

This section presents an abstract version of BGP that is designed for a formal investigation of properties related to protocol convergence. The reader not familiar with the BGP protocol may wish to consult [16, 7, 17]. The model abstracts away many of the nitty-gritty details of BGP, making it easier to address convergence-related issues. The most important simplifications we have made are (1) network addresses are treated as a flat space, ignoring containment and aggregation, (2) the attributes **MED**, **ORIGIN**, **ATOMIC AGGREGATE**, and **AGGREGATOR** are ignored, (3) we assume that there is at most one link between any two ASes, (4) all issues relating to internal BGP (iBGP) are ignored, (5) default routes are ignored, (6) we assume that no two ASes can originate the same destination address, (7) we assume that there is one global default value for the **LOCAL PREFERENCE** attribute.

2.1 Networks and Routes

The Internet is modeled as an undirected graph $G = (V, E)$, called the *AS graph*, where the vertices V represent Autonomous Systems and the edges E represent peering relationships. Every vertex v "learns" a set of *route announcements* from its immediate neighbors. Route announcements undergo transformations as they pass from one autonomous system to the next.

Route announcements are records with the following attributes

nlri	:	network layer reachability information (a destination's network address)
next_hop	:	next hop (vertex number)
as_path	:	ordered list of vertices traversed
loc_pref	:	local preference

We will often call a route announcement simply a *route*. The **as_path** attribute records the path that a route has traversed through G as it passes from one vertex to the next. Suppose vertex v has learned a route r with $r.\textbf{as\_path} = [v_k, \cdots, v_2, v_1]$. This indicates r was originated at v_1, was passed to v_2, and so on, until finally v_k passed it to v. In this case $r.\textbf{next\_hop}$ is the vertex that passed this information to v, and so this should be the same as v_k, the first vertex in $r.\textbf{as\_path}$. In other words, **next_hop** is not actually needed in our model, but we retain it here to simplify the presentation. The local preference attribute, **loc_pref**, is used only within an AS and is not passed between ASes. This attribute is used to indicate the relative ranking of different paths to a destination. We assume that there is a default value for the **loc_pref** attribute, **dlp**.

2.2 Best Route Selection

If vertex v has a choice between two different routes r_1 and r_2 with the same network destination, $r1.\textbf{nlri} = r2.\textbf{nlri}$, then it uses the choice function **Select**, to select one *best route.* If we assume $r_1.\textbf{next\_hop} \neq r_2.\textbf{next\_hop}$, then the function $\textbf{Select}(r_1, r_2)$ is defined to be

1. if $r_1.\textbf{loc\_pref} \neq r_2.\textbf{loc\_pref}$, then pick r_i with highest **loc_pref**,
2. otherwise, if length$(r_1.\textbf{as_path}) \neq$ length$(r_2.\textbf{as_path})$, then pick r_i with shortest **as_path**,
3. otherwise, pick r_i with lowest **next_hop** (break tie).

It is rule (1) that allows policy-based metrics to override distance-based metrics.

A set of routes R is *consistent* if for any two distinct routes r_1 and r_2 in R, if $r1.\textbf{nlri} = r2.\textbf{nlri}$, then $r_1.\textbf{next\_hop} \neq r_2.\textbf{next\_hop}$. If R is consistent, then the notation $\textbf{Select}(R)$ denotes that subset of R arrived at by repeatedly applying **Select** to pairs of routes with the same **nlri** value. It is easy to see that $\textbf{Select}(R)$ cannot contain distinct routes r_1 and r_2 with $r1.\textbf{nlri} = r2.\textbf{nlri}$.

2.3 Route Record Transformations

Two types of transformations can be applied to the route records as they pass from one vertex to the next. The first is a *BGP-specific* transformation that is fixed in the protocol definition. The second type is encoded in *transformation policies* which are defined independently by each AS.

If $v, w \in V$ and R is a set of routes, then $\textbf{PVT}(w \leftarrow v)[R]$ represents the BGP-specific *path-vector transformation* of R as the routes of R pass from v to w. This transformation enforces three rules: (1) the **as_path** records the path of vertices that a record has traversed, (2) **loc_pref** values are not passed from one vertex to another, and (3) a vertex v never accepts a route r if $r.\textbf{as\_path}$ contains v. This is defined on singleton sets as follows. If $w \in r.\textbf{as\_path}$, then $\textbf{PVT}(w \leftarrow v)[\{r\}] = \{\}$. Otherwise $\textbf{PVT}(w \leftarrow v)[\{r\}] = \{r'\}$, where $r'.\textbf{nlri} = r.\textbf{nlri}$, $r'.\textbf{as\_path} = v \parallel r.\textbf{as\_path}$, $r'.\textbf{next\_hop} = v$, and $r'.\textbf{loc\_pref} = \textbf{dlp}$. Here the notation $v \parallel l$ is used to denote the list formed by prepending v onto the beginning of l. We then define $\textbf{PVT}(v \leftarrow w)[\{r_1, \cdots, r_n\}]$ to be $\textbf{PVT}(v \leftarrow w)[\{r_1\}] \cup \cdots \cup \textbf{PVT}(v \leftarrow w)[\{r_n\}]$.

For each $\{v, w\} \in E$ there are four *transformations policies.* These are the import and export transformations for AS v on this link and the import and export transformations for w on this link. The notation $\textbf{import}(v \leftarrow w)$ denotes the transformation policy that defines how routes are transformed when importing them into v from w. The notation $\textbf{export}(v \rightarrow w)$ defining how routes are transformed when exporting them from v to w. If R is a set of routes, then the sets $\textbf{import}(v \leftarrow w)[R]$ and $\textbf{export}(v \rightarrow w)[R]$ denote the application of the corresponding transformation policies to the routes in R.

A transformation policy P is defined by a list of *policy rules*

$$\begin{array}{ccc} H_1 & \Longrightarrow & A_1 \\ H_2 & \Longrightarrow & A_2 \\ \vdots & \vdots & \vdots \\ H_n & \Longrightarrow & A_n \end{array}$$

The head of each rule H_i is a boolean predicate formed over route attributes. We will consider only predicates constructed from basic predicates $\textbf{nlri} = d$, $v \in \textbf{as\_path}$, and $\textbf{as\_path} = [p_1, p_2, \cdots, p_n]$, where each *pattern* p_i is either an AS number or the *wildcard* ?. Basic predicates are then closed under the boolean operations of conjunction, disjunction, and negation. The action of each rule A_i is either **reject**, **allow**, or an attribute assignment (with an implicit **allow**). Only assignments to the attribute **loc_pref** are allowed. These policy rules define a function on sets of routes in the following way. If R is a set of routes, then $r' \in P(R)$ if and only if there is an $r \in R$ and for some i, $H_i(r)$ is true, while $H_j(r)$ is false for $j < i$, and $r' = A_i(r)$. (Note that $\textbf{allow}(\mathbf{r}) = \mathbf{r}$ and that $\textbf{reject}(\mathbf{r})$ is not defined.) An example of an import transformation is

$$\begin{array}{rcl} 17 \in \textbf{as\_path} & \Longrightarrow & \textbf{reject} \\ \textbf{as\_path} = [12, ?, 16] & \Longrightarrow & \textbf{loc\_pref} := \textbf{dlp} + 1 \\ \textbf{true} & \Longrightarrow & \textbf{loc\_pref} := \textbf{dlp} \end{array}$$

This policy would reject any route with AS 17 in the **as_path**. The **as_path** value $[12, ?, 16]$ includes a *wildcard* ? that matches any number. So the second rule accepts any route r with $r.\textbf{as\_path}$ of length three that starts with 12 and ends with 16. It then sets **loc_pref** attribute to $\textbf{dlp} + 1$. Finally, this policy would accept all other routes records with the default value for **loc_pref**.

2.4 Evaluation States

Each $i \in V$ has an initial set of route records, c_i^0, which models the destinations originated by i. We will assume that for any $r \in c_i^0$ that $r.\textbf{as\_path}$ is empty and that **next_hop** is not defined. We will also assume that no two i and j in V originate the same destinations. That is, we assume that for each $r_1 \in c_i^0$, and $r_2 \in c_j^0$, that $r1.\textbf{nlri} \neq r2.\textbf{nlri}$.

Suppose $G = (V, E)$ and $V = \{0, 1, \cdots, n\}$. An *evaluation state* is a tuple $\langle c_0, c_1, \cdots, c_n \rangle$ where each c_i is a set of route records that represents the "contents" of AS i in this state. This set can be empty, meaning that AS i has not learned of or accepted any routes. The evaluation state $\langle c_0^0, c_1^0, \cdots, c_n^0\rangle$ is called the *initial state*, and is denoted by s_0.

2.5 Dynamic Behavior

A *BGP system*, $S = \langle G, \text{Policy}(G), s_0\rangle$, comprises an AS graph $G = (V, E)$, a set Policy(G) containing the import and export policies for every $i \in V$, and an initial state s_0 that defines the destinations originated by each $i \in V$.

Informally, if the BGP system is in a state s and $i \in A$, then i is *activated* and can compute its best routes based on the routes it obtains from its immediate neighbors, after the appropriate transformations. The set of choices available to i in state $s = \langle c_0, c_1, \cdots, c_n\rangle$, denoted $\mathbf{Choices}(i, s)$, is defined to be

$$\bigcup_{\{i,j\}\in E} \mathbf{import}(i \leftarrow j)[\mathbf{PVT}(i \leftarrow j)[\mathbf{export}(j \rightarrow i)[c_j]]]$$

An evaluation state s' is *reachable* from s_0 if $s' = s_0$, or if there is a reachable state s, and a non-empty set $A \subseteq V$ of activated ASes, such that $s \xrightarrow{A} s'$, where this *transition relationship*

$$\langle c_0, c_1, \cdots, c_n\rangle \xrightarrow{A} \langle c_0', c_1', \cdots, c_n'\rangle$$

is defined as

$$c_i' = \begin{cases} c_i & \text{if } i \notin A, \\ c_i^0 \bigcup \mathbf{Select}(\mathbf{Choices}(i, s)) & \text{otherwise.} \end{cases}$$

It is easy to prove, by induction, that the set to which the best route selection function is applied is always consistent in the sense of Section 2.2.

With this definition it is clear that the import policy $H \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$ is equivalent to $H \Longrightarrow \textbf{allow}$. Also note that if all import and export rules are of the form $\textbf{true} \Longrightarrow \textbf{allow}$, then the dynamic model reduces to a pure distance-vector protocol.

In terms of the terminology of [16], the set c_i corresponds to the LOC-RIB of AS i, the set $\mathbf{import}(i \leftarrow j)[\mathbf{PVT}(i \leftarrow j)[\mathbf{export}(j \rightarrow i)[c_j]]]$ corresponds to AS i's ADJ-RIBs-IN for peer j, while the set $\mathbf{export}(j \rightarrow i)[c_j]$ corresponds to AS j's ADJ-RIBs-OUT for peer i.

Note that all of the $i \in A$ are activated simultaneously. The intent is to model all possible evaluations in an asynchronous distributed execution of the BGP protocol. An *activation sequence* is any path, finite or infinite, starting at the initial state, $s_0 \xrightarrow{A_1} s_1 \xrightarrow{A_2} s_2 \xrightarrow{A_3} \cdots$. Since the initial state is unique, we will identify an activation sequence with the sequence of subsets $A_1, A_2, A_3, \cdots$. An infinite activation sequence $A_1, A_2, A_3, \cdots$ is *fair* if no $i \in V$ is ever "starved out." That is, for any $i \in V$, and for any t, there must be a $k > 0$ such that $i \in A_{t+k}$.

2.6 Evaluation Graphs, Solvable and Unsolvable BGP Systems

It is easy to check that there are only finitely many reachable states. For a system S, the *evaluation graph*, Eval(S), is defined to be the directed graph, where for each reachable state s, there is a vertex labeled by s in Eval(S), and if s and s' are reachable states and $s \xrightarrow{A} s'$ for some non-empty $A \subseteq V$, then there is a directed edge from s to s' labeled by $s \xrightarrow{A} s'$ in Eval(S). A state $s \in$ Eval(S) is called a *final state* if $s \xrightarrow{\{v\}} s$ for every $v \in V$. That is, if the system S enters a final state, then no matter what collection of ASes $A \subseteq V$ is activated, the system will remain in that state. Note that in a final state some nodes may have no routes to certain destinations.

A BGP system is *solvable* if there is at least one final state in Eval(S). Otherwise, system S is *unsolvable*. Notice that an unsolvable system can never converge to a stable state, since none exists. Operationally, this means that for unsolvable systems, BGP is likely to go into a "protocol oscillation" that will never terminate. For an example of an unsolvable system, see BAD GADGET in Section 3.

2.7 Solutions and Routing Trees

Suppose $r \in c_i^0$ is an initial route record for destination $d = r.\mathbf{nlri}$ and that $s = \langle c_0, \cdots, c_n\rangle$ is any state in Eval(S). Vertex $j \in V$, $(i \neq j)$, is said to have a path to d in state s if there is a route record $r' \in c_j$ with $r'.\mathbf{nlri} = d$. Note that it will always be the case that $r'.\textbf{as\_path} = [\cdots, i]$. It is also easy to see that for each state s and each destination d there is a subgraph of G defined by the **as_path** at each node for the corresponding route record. This graph is denoted routing(d, s). More formally, routing$(d, \langle c_0, c_1, \cdots, c_n\rangle) = (V', E')$, where

$$\begin{aligned} V' &= \{j \in V \mid \exists r \in c_j, r.\mathbf{nlri} = d\} \\ E' &= \textstyle\bigcup_{j\in V'}\{\{w, v\} \in E \mid \exists r \in c_j, r.\mathbf{nlri} = d \\ &\qquad \wedge r.\textbf{as\_path} = [v_1, v_2, \cdots, v_k] \\ &\qquad \wedge((w = j \wedge v = v_1) \vee (w = v_t \wedge v = v_{t+1}))\} \end{aligned}$$

This graph is connected, since each path ends in the vertex that originates d. Next, we show that this graph is a *routing tree* when s is a final state.

Theorem 2.1 *Suppose s is a final state. For every possible destination d, the graph* routing(d, s) *is a tree (an acyclic, connected graph).*

Proof: Suppose that s is a final state and d is a destination originated by vertex i. We know that routing$(d, s) = (V', E')$ is connected (see discussion above), and so all that needs to be shown is that routing(d, s) is acyclic. We assume that there is a cycle in routing(d, s) and derive a contradiction. If there is a cycle then there must be two paths in routing(d, s), $p_1 = [v_1, v_2, \cdots, v_k, i]$ and $p_2 = [w_1, w_2, \cdots, w_m, i]$, whose union contains a cycle. That is, there is a vertex $v \in V'$ such that $v = v_x = w_y$ for some x and y, but $v_{x+1} \neq w_{y+1}$. Since AS v_1 thinks that v's route to d is $[v_x, v_{x+1}, \cdots, v_k, i]$ while AS w_1 thinks that v's route is $[w_y, w_{y+1}, \cdots, w_m, i]$ at least one of them must be wrong. Without loss of generality, assume that it is v_1 that is wrong. Let $r \geq 1$ be the largest index less than x such that v_r's route in s is $[v_r, v_{r+1}, \cdots, v_k, i]$. There is always such an index since v_1 has such a route but v_r is the one that is closest to $v = v_x$ on the path $[v_1, v_2, \cdots, v_k]$. Therefore in s, v_{r+1} does not have the route $[v_{r+1}, v_{r+2}, \cdots, v_k, i]$. We can conclude that if v_r is activated, then it must change its route because it learns that the one it currently has is no longer valid. ■

We now show how a routing tree can be used to construct an activation sequence that will arrive in a final state. Suppose T is a tree with root i of height m. Let $A(j, i, T)$, for $1 \leq j \leq m$ be the set of vertices in T that are distance j from root i. Define the activation sequence $A(i, T)$ to be $A(1, i, T), A(2, i, T), \cdots, A(m, i, T)$. That is, $A(i, T)$ is the sequence that activates the vertices of T in a breadth-first order w.r.t. vertex i. Let $s_0 \stackrel{A(i,T)}{\longrightarrow} s_m$ denote the path $s_0 \stackrel{A(1,i,T)}{\longrightarrow} s_1 \stackrel{A(2,i,T)}{\longrightarrow} \cdots \stackrel{A(m,i,T)}{\longrightarrow} s_m$ in Eval(S).

Theorem 2.2 *Let S be a BGP system that contains a single destination d. Suppose that destination d is originated by vertex i and that s is a final state. Let $T = \text{routing}(d, s)$ be the routing tree rooted at i. Then $s_0 \stackrel{A(i,T)}{\longrightarrow} s$.*

Proof: By induction on the depth of T. If at any point in the activation sequence of length k the routing graph is not the subtree of T of height k, then we can find a vertex that will switch routes in state s. This means that s is not a final state, which is a contradiction. ■

Let S be a BGP system that contains a single destination d. The analysis above suggests the following brute-force algorithm for checking solvability of S:

1. Enumerate all subgraphs T of G that are trees rooted at d.
2. Attempt to find a tree T such that $s_0 \stackrel{A(d,T)}{\longrightarrow} s$, where s is a final state.
3. The system is solvable if and only if at least one such tree exists.

Given any one tree, checking step (2) is polynomial in the size of the system S since checking if a state is final requires only checking that it does not change if all nodes are fired at the same time. However, in the worst-case, step (1) is exponential in the number of ASes.

3 A Managerie of BGP Systems

This Section presents several examples of BGP systems that illustrate distinct types of routing anomalies.

3.1 BAD GADGET

We now present an example of an unsolvable BGP system, called BAD GADGET, which is similar in spirit to examples of [18]. That is, no execution of the BGP protocol can possibly arrive at a stable routing. BAD GADGET is presented in Figure 1, where each node represents an AS. Suppose that there is a single destination with d originated by AS 0.

Each AS prefers the counter-clockwise route of length 2 over all other routes to the 0 That is, AS 3 prefers the route $3-2-0$ over $3-0$. The policies of BAD GADGET can be implemented as follows. Let $i \oplus 1$ denote $i+1 \pmod 3$ and $i \ominus 1$ denote $i-1 \pmod 3$. All export rules have the form $\mathbf{nlri} = d \Longrightarrow \mathbf{allow}$. For import, each node i will have the following set of policies: $\mathbf{import}(i \leftarrow i \oplus 1)$ is

$$\begin{array}{rcl} \left(\begin{array}{c} \mathbf{nlri} = d \quad \wedge \\ \mathbf{as\_path} = [i \oplus 1, 0] \end{array}\right) & \Longrightarrow & \mathbf{loc\_pref} := \mathbf{dlp} + 1 \\ \mathbf{nlri} = d & \Longrightarrow & \mathbf{loc\_pref} := \mathbf{dlp} \end{array}$$

while $\mathbf{import}(i \leftarrow i \ominus 1)$ is $\mathbf{nlri} = d \Longrightarrow \mathbf{allow}$. That is, each AS assigns the counter-clockwise path of length 2 a **loc_pref** of $\mathbf{dlp} + 1$, while all other routes get the default value **dlp**.

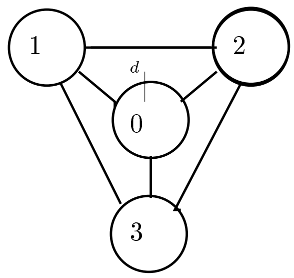

Figure 1: The AS graph for BAD GADGET.

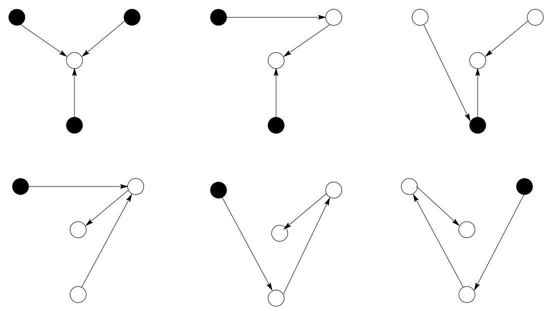

Figure 2: Possible routing trees for BAD GADGET.

We now show that BAD GADGET has no solution. For BAD GADGET, we need only consider the sixteen spanning trees rooted at AS 0. Any tree that does not span this graph cannot be a solution because AS 1, 2 and 3 each has a direct route to d. Furthermore, since this system is symmetric, we need only consider the six cases presented in Figure 2. In this figure we have marked with a solid circle those ASes that will change their selection of the best route to d. Each marked AS will either pick the counter-clockwise route of length 2, or revert to the direct route to d. It is easy to see that the system has no solution.

3.2 SURPRISE

The BAD GADGET example was constructed to illustrate a very simple unsolvable system. Certainly from a network operator's perspective it might seem rather contrived. We now present what is arguably a more plausible system, called SURPRISE, that is "good" in the sense that it is solvable. However, after a single link failure this reasonable system falls into one that is equivalent to BAD GADGET, and so becomes unsolvable. (This system inspires the definition of the "k-ROBUST" problem of Section 4.)

Figure 3 presents a system called SURPRISE. AS 5 originates destination d, a popular web portal. The labels $C2$, $C3$, and $C4$ indicate the capacity of three links, with $C2$ having a low capacity, $C3$ an intermediate capacity, and $C4$ the highest capacity. These capacities will be relevant to the backup policies of AS 1, 2, and 3.

AS 4 is a high quality network, with a very high speed link to AS 5. AS 1, 2, and 3 all have high speed links directly to AS 4. On the other hand, AS 0 has a low quality link to AS 5, and for this reason AS 1, 2, and 3 all prefer to go through AS 4 to reach destination d, rather than using AS 0.

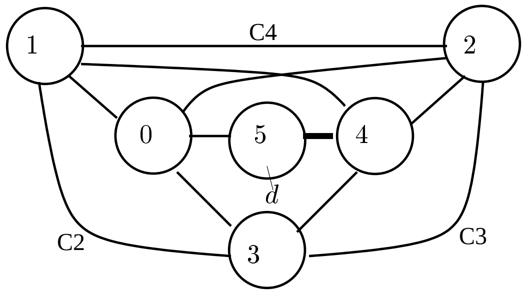

Figure 3: AS graph for SURPRISE.

AS 3 has a simple backup plan for destination d, designed with the failure of link $\{3, 4\}$ in mind. AS 3 would like to go through AS 2 if link $\{3, 4\}$ fails for two reasons. First, as already mentioned, nobody likes AS 0 (at least not for destination d). Second, AS 3 decides in favor of AS 2 over AS 1 simply because its link to AS 2 is of higher capacity. Implicit in AS 3's backup plan is the assumption that AS 2's direct link to AS 4 will most likely remain operational in the event that $\{3, 4\}$ fails.

In the policy notation described in detail in Section 2, the relevant import policy fragment for AS 3 is

$$\begin{array}{rcl} \mathbf{as\_path} = [4,5] & \Longrightarrow & \mathbf{loc\_pref} := \mathbf{dlp} + 2 \\ \mathbf{as\_path} = [2,?,5] & \Longrightarrow & \mathbf{loc\_pref} := \mathbf{dlp} + 1 \\ \mathbf{true} & \Longrightarrow & \mathbf{loc\_pref} := \mathbf{dlp} \end{array}$$

The first rule matches any route record that was originated in AS 5 and traversed AS 4 before arriving at AS 3. Any such record has its LOCAL PREFERENCE incremented by 2 (over some default LOCAL PREFERENCE **dlp**). The second rule matches any route record that was originated in AS 5, traversed any AS (indicated by the wildcard ?), then traversed AS 2 before arriving at AS 3. Such records have their LOCAL PREFERENCE incremented by 1. The reason that the second rule uses a wildcard in the AS-PATH attribute, rather than explicitly writing the path $[2, 4, 5]$, is that the network administrator who implemented this rule had heard that AS 4 might change its AS number (since it recently merged with a larger ISP) and thought it best not to hard-code the AS number into the rule. This administrator was unaware of the direct link that AS 2 maintains with AS 0. Finally, the last rules accepts all other routes with default value for LOCAL PREFERENCE.

AS 1 and 2 implement the same policies (modulo rotational symmetry) in the same way, each preferring their counter clockwise peer in the backup case.

This system is clearly solvable, since the system will enter a stable state as soon as AS 1, 2 and 3 have each picked their two-hop routes to d going through AS 4. We would argue that these policies are fairly plausible. One objection that might be raised is that for AS 1, the choice of AS 3 as a backup does not seem right, since the link to AS 2 is of higher capacity. That observation is correct, but the explanation is simple. Link $\{1, 2\}$ has only recently been upgraded from a very low $C1$ capacity (lower than $C2$) to the high-speed capacity $C4$. AS 2's network administrators have updated their policy to reflect this, but the overworked administrators of AS 1 have been fighting other fires and simply haven't had time to update this (as yet never-used) backup policy.

Now suppose that the BGP session on link $\{4, 5\}$ is lost. This could be the result of a cable cut or a misbehaving router. Note that the backup policies of AS 1, 2, and 3 have not explicitly planned for this failure. It is not hard to see that this system is immediately transformed into one equivalent to BAD GADGET, and so becomes unsolvable. The same thing happens if links $\{1, 4\}$, $\{2, 4\}$, and $\{3, 4\}$ all go down at the same time (this could happen if at some point they are all riding over the same strand of physical fiber and this fiber is cut).

3.2.1 DISAGREE

The system DISAGREE, presented in Figure 4, illustrates the fact that a BGP system can have multiple solutions.

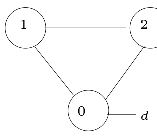

Figure 4: AS graph for DISAGREE.

Only AS 0 originates a single destination with $\mathbf{nlri} = d$. In this system, both AS 1 and AS 2 will prefer to go through the other to get to d (hence the name DISAGREE for this system). The rules implementing this policy are as follows. For $i, j \in \{0, 1, 2\}, i \neq j$, $\mathbf{export}(i \to j)$ is $\mathbf{true} \Longrightarrow \mathbf{allow}$. For $i \in \{1, 2\}$, $\mathbf{import}(i \leftarrow 0)$ is

$$\mathbf{nlri} = d \wedge \mathbf{as\_path} = [0] \Longrightarrow \mathbf{loc\_pref} := \mathbf{dlp}.$$

Finally, $\mathbf{import}(1 \leftarrow 2)$ is

$$\mathbf{nlri} = d \wedge \mathbf{as\_path} = [2, 0] \Longrightarrow \mathbf{loc\_pref} := \mathbf{dlp} + 1$$

and $\mathbf{import}(2 \leftarrow 1)$ is

$$\mathbf{nlri} = d \wedge \mathbf{as\_path} = [1, 0] \Longrightarrow \mathbf{loc\_pref} := \mathbf{dlp} + 1.$$

The initial state of the system is $\langle\{r\}, \{\}, \{\}\rangle$, where $r.\mathbf{nlri} = d$, $r.\mathbf{as\_path} = [\ ]$, and $r.\mathbf{loc\_pref} = \mathbf{dlp}$. Figure 5 shows the evaluation graph for DISAGREE (all arcs with labels containing a 0 have been ignored for simplicity, since AS 0 does not import any routes). The contents of each node indicate the paths to destination d from AS 1 and 2. For example, a state containing $2 - 1 - 0$ indicates AS 2 has a route to d with **as_path** $= [1, 0]$. Final states are indicated with double borders. Note that there are two final states in this graph. For each final state, the corresponding routing tree is depicted to its right.

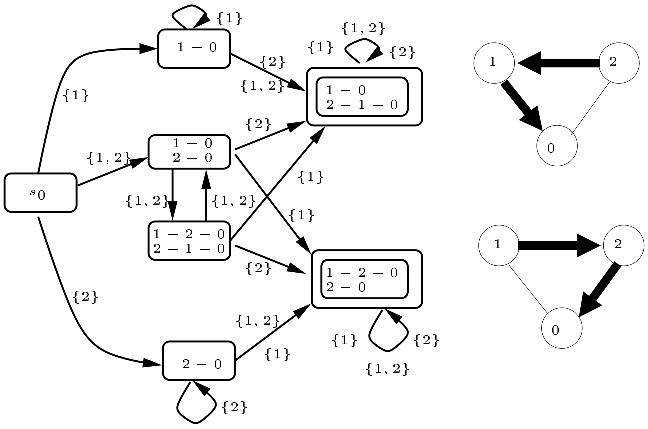

Figure 5: Evaluation graph and two routing trees for DISAGREE.

Note also that the fair activation sequence $\{1, 2\}, \{1, 2\}, \{1, 2\}, \cdots$ leads to protocol divergence. This sequence seems unlikely to arise in practice because it relies on a precise sequence of events occurring repeatedly. If either AS 1 or AS 2 ever activates by itself, then the system will converge on a solution.

3.3 PRECARIOUS

Solvable systems are "good" in the sense that there is at least one activation sequence that will converge on a solution. However, as the DISAGREE example illustrated, having a solution does not mean that every activation sequence will converge on a solution.

The "weak divergence" of DISAGREE should be contrasted with the "strong divergence" of BAD GADGET, whose evaluation graph has a "trap" from which it can never exit to a final state. We can formally define a *trap* to be a subgraph of Eval(S) that (1) does not contain a final state, and (2) has no arcs directed out of this subgraph. An unsolvable systems is truly pathological in the sense that its entire evaluation graph is a trap.

Can a solvable BGP system have a trap in its evaluation graph? The answer is "yes." Consider the system PRECARIOUS, whose AS graph is presented in Figure 6. A subgraph of this system is equivalent to the system DISAGREE presented above, with AS 0 originating the single destination d_1. Recall that DISAGREE has two distinct routing trees associated with destination d_1. Attached to AS 1 is a link that goes to the center of BAD GADGET 1, which accepts a route to d_1 only if AS 1 takes the direct route to d_1. BAD GADGET

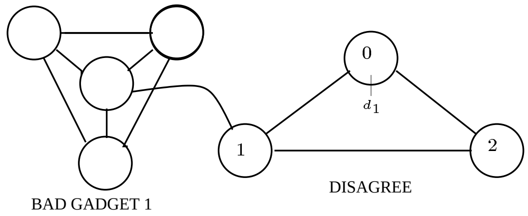

Figure 6: AS graph for PRECARIOUS.

1 is configured to oscillate only when its center node accepts this route to d_1. Therefore, this system has only one solution: when AS 2 accepts the direct route to d_1 and AS 1 accepts the route through AS 2 to d_1. This example shows that a solvable system — one with a unique solution — can contain a trap. Starting at the initial state of the evaluation graph, any arc labeled A, where $1 \in A$ and $2 \notin A$, will lead to a trap.

4 Complexity Results

This section presents our complexity results concerning various global properties of BGP systems. For a review of NP-completeness, the reader is encouraged to consult [4]. We will consider the following problems.

- REACHABILITY: "Given a system S, AS v in S, AS w in S and a destination d originated by w, does there exist a final state in $s \in$ Eval(S) in which v is a node in routing(d, s)?"
- ASYMMETRY: "Given a system S, AS v in S, AS w in S, a destination d_1 originated by w, a destination d_2 originated by v, does there exist a final state in $s \in$ Eval(S) in which the route from v to d_1 is not the reverse of the route from w to d_2?
- SOLVABILITY: "Given a system S, does there exist a final state in Eval(S)?"
- UNSOLVABILITY: "Given a system S, is there no final state in Eval(S)?"
- SOLVABILITY (SD): "Given a system S having only a single destination d originated by some AS w, does there exist a final state in Eval(S)?"
- UNSOLVABILITY (SD): "Given a system S having only a single destination d originated by some AS w, is there no final state in Eval(S)?"
- TRAPPED: "Given a system S, does Eval(S) contain a trap?" (See Section 3.3 for the definition of a trap.)
- k-ROBUST: "Given a solvable system S having only a single destination d originated by some AS w, will S remain solvable under any possible failure of k links?"
- UNIQUE: "Given a system S, does there exist exactly one final state in $Eval(S)$?"
- UNIQUE (SD): "Given a system S having only a single destination d originated by some AS w, does there exist exactly one final state in $Eval(S)$?"

$2 \le i \le n$	$\textbf{import}(x_i \leftarrow x_{i-1})$	$\textbf{nlri} = d \wedge \overline{x}_{i-1} \notin \textbf{as\_path} \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(1)
$2 \le i \le n$	$\textbf{import}(x_i \leftarrow \overline{x}_{i-1})$	$\textbf{nlri} = d \wedge x_{i-1} \notin \textbf{as\_path} \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(2)
$2 \le i \le n$	$\textbf{import}(\overline{x}_i \leftarrow x_{i-1})$	$\textbf{nlri} = d \wedge \overline{x}_{i-1} \notin \textbf{as\_path} \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(3)
$2 \le i \le n$	$\textbf{import}(\overline{x}_i \leftarrow \overline{x}_{i-1})$	$\textbf{nlri} = d \wedge x_{i-1} \notin \textbf{as\_path} \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(4)
	$\textbf{import}(x_1 \leftarrow w)$	$\textbf{nlri} = d \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(5)
	$\textbf{import}(\overline{x}_1 \leftarrow w)$	$\textbf{nlri} = d \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(6)
	$\textbf{import}(z \leftarrow x_n)$	$\textbf{nlri} = d \wedge \overline{x}_n \notin \textbf{as\_path} \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(7)
	$\textbf{import}(z \leftarrow \overline{x}_n)$	$\textbf{nlri} = d \wedge x_n \notin \textbf{as\_path} \Longrightarrow \textbf{loc\_pref} := \textbf{dlp}$	(8)
$1 \le i \le n$	$\textbf{import}(x_i \leftarrow \overline{x}_i)$	$\textbf{nlri} = d \Longrightarrow \textbf{loc\_pref} := \textbf{dlp} + 1$	(9)
$1 \le i \le n$	$\textbf{import}(\overline{x}_i \leftarrow x_i)$	$\textbf{nlri} = d \Longrightarrow \textbf{loc\_pref} := \textbf{dlp} + 1$	(10)

Figure 7: Import policies of ASSIGN (n).

MULTIPLE: "Given a system S, does there exist more than one final state in $Eval(S)$?"

MULTIPLE (SD): "Given a system S having only a single destination d originated by some AS w, does there exist more than one final state in $Eval(S)$?"

Our proofs will rely on a reduction from 3-SAT, a well-known NP-complete problem. An instance of 3-SAT consists of a set of boolean variables and a formula based on these variables and their negations where the formula has the form of a conjunction of terms each of which is a disjunction of three literals (a literal is either a variable or its negation). The 3-SAT problem asks if there exists a satisfying assignment for a given instance. For example, an instance of 3-SAT might consist of variables x_1, x_2, x_3 and the formula

$$(x_1 \vee x_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$$

where $\overline{x}_i$ denotes the negation of variable x_i. Notice that setting x_1 to be **true** and x_2 and x_3 to be **false** is a satisfying assignment for this formula.

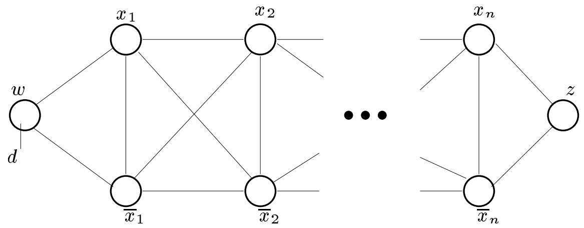

Figure 8: AS graph for ASSIGN (n).

Suppose we are given n variables, $X = \{x_1, x_2, \cdots, x_n\}$. In order to construct the reductions from 3-SAT we will use a BGP system, called ASSIGN (n), having nodes w and z where w originates a single route d. Furthermore, ASSIGN (n) is constructed such that (1) every boolean assignment for X corresponds to a unique path from z to w, and (2) every path from z to w corresponds to a unique boolean assignment for X.

The AS graph of ASSIGN (n) is presented in Figure 8. The policies of ASSIGN (n) must enforce the rules: (1) if x_i is in the **as_path** of a route to d, then $\overline{x}_i$ is not in the **as_path** of this route, (2) if $\overline{x}_i$ is in the **as_path** of a route to d, then x_i is not in the **as_path** of this route, (3) once a route to d is chosen, it can be "locked in" so that it will not change. In this way, the **as_path** of a route record at z will correspond to a assignment for the variables of X.

All export policies are **true** $\Longrightarrow$ **allow**. The import policies of ASSIGN (n) are defined in Figure 7. The predicates of the form $x_{i-1} \notin \textbf{as\_path}$ ($\overline{x}_{i-1} \notin \textbf{as\_path}$) guarantee that any path from z to w cannot contain both x_j and $\overline{x}_j$. Rules (9) and (10) are similar to the rules of DISAGREE, and they allow each pair x_i, and $\overline{x}_i$ to lock into one of two states.

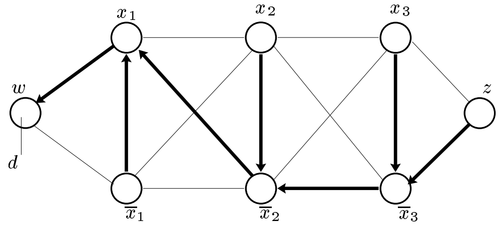

Figure 9: Example of ASSIGN (3).

For example, Figure 9 presents the system ASSIGN (3), and the routing tree that corresponds to the assignment $x_1 =$ **true**, $x_2 =$ **false**, $x_3 =$ **false**.

Note that the size of ASSIGN (n) must be polynomial in n. The construction of ASSIGN (n) demonstrates that a solvable system of size $\mathcal{O}(n)$ can have $\mathcal{O}(2^n)$ final states.

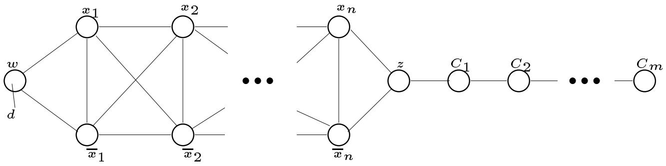

Figure 10: AS graph construction for REACHABILITY.

Theorem 4.1 REACHABILITY *is NP-complete.*

Proof: Consider any state s. To check whether s is a final state can be done in time polynomial in the size of the BGP system, since all we need do is activate each node and check if its state remains unchanged. Testing if v is a node in routing(d, s) just requires seeing if v contains a route to d in state s. Thus REACHABILITY is in NP.

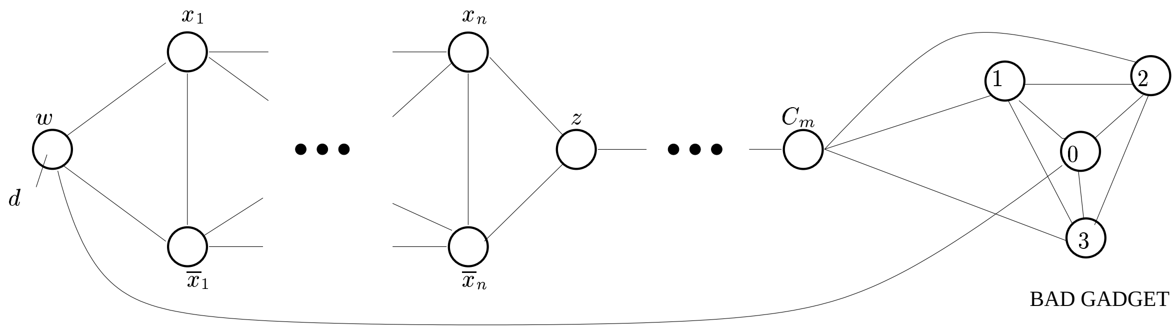

Figure 11: AS graph construction for SOLVABILITY (SD).

Notice that an instance of REACHABILITY is determined by a tuple (S, w, d, v) specifying the system S, AS w, destination d, and AS v. We describe the construction of $R = (S, w, d, v)$, an instance of REACHABILITY, from an instance I of 3-SAT such that I has a satisfying assignment if and only if d is reachable from v. The instance R is constructible in time polynomial in the size of I.

Let I consist of clauses $C_1, C_2, \ldots, C_m$, where each C_i is a disjunction of three literals and each literal is one of $x_1, \overline{x}_1, \ldots, x_n$ or $\overline{x}_n$. The graph $G = (V, E)$ defining the topology of the system we wish to construct has vertex set $V = \{w, z, x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n, C_1, \ldots, C_m\}$, and edge set depicted in Figure 10. Notice that this graph is ASSIGN (n) extended with the nodes C_i and the corresponding edges.

All nodes export all routes to all immediate neighbors. For the portion of the system that implements ASSIGN (n), the import policy rules are given in Figure 7. If $C_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$, then for $2 \leq i \leq m$, define $\textbf{import}(C_i \leftarrow C_{i-1})$ to be

$$\begin{pmatrix} l_{i,1} \in \textbf{as\_path} \vee \\ l_{i,2} \in \textbf{as\_path} \vee \\ l_{i,3} \in \textbf{as\_path} \end{pmatrix} \implies \textbf{loc\_pref} := \textbf{dlp} \qquad (1)$$

and $\textbf{import}(C_1 \leftarrow z)$ to be

$$\begin{pmatrix} l_{1,1} \in \textbf{as\_path} \vee \\ l_{1,2} \in \textbf{as\_path} \vee \\ l_{1,3} \in \textbf{as\_path} \end{pmatrix} \implies \textbf{loc\_pref} := \textbf{dlp} \qquad (2)$$

Our instance of REACHABILITY is $R = (S, w, d, v)$, where $v = C_m$. Our claim is that d is reachable from C_m if and only if I has a satisfying assignment. That is, there is an activation sequence p that results in a final state where a route r from C_m to d exists if and only if there is a satisfying assignment to I. First, assume that a satisfying assignment A exists. Let l_i be x_i if x_i is **true** in A and otherwise $l_i = \overline{x}_i$, $1 \leq i \leq n$. Consider the activation sequence

$$p = \{w\}, \{l_1\}, \ldots, \{l_n\}, \{z\}, \{C_1\}, \ldots, \{C_m\}.$$

It is easy to check that after completion of p the fact that A is a satisfying assignment implies that a route from C_m to w has been established. Suppose some node $\overline{l}_i$ is activated after p. Then due to the settings of the **loc_pref**'s (see Figure 7) $\overline{l}_i$ will always choose l_i as its next hop. Define activation sequence $p' = \{\overline{l}_1\}, \ldots, \{\overline{l}_n\}$. Consider the activation sequence $p||p'$ obtained by performing p' after p. We claim that this results in a final state. By the construction of ASSIGN (n), activating any of w, z, x_i or $\overline{x}_i$ will not cause a change and a trivial inductive argument shows that no C_i can change its state if activated. Thus the activation sequence $p||p'$ results in a final state. By the policy rules for the C_i's, it is the case that the path from w to z passes through a satisfying assignment for I, each C_i will accept the route announced to it by C_{i-1} (and similarly C_1 accepts the route announced by z). Thus the activation sequence $p||p'$ results in a final state in which a route has been established from C_m to d.

Suppose on the other hand that there is some activation sequence p leading to a final state in which C_m has a route to d. It is trivial to check that any such route must go through each C_i and z. The policy rules of the C_i's guarantee that at least one of $l_{i,j}$, $1 \leq j \leq 3$, must be in the route. The policy rules guarantee that the route from z to d does not contain both x_i and $\overline{x}_i$ in the route. Thus setting x_i to **true** if x_i is in the route and to **false** otherwise (that is, if $\overline{x}_i$ is in the route) gives a satisfying assignment for I. ■

Theorem 4.2 ASYMMETRY *is NP-complete.*

Proof: Modify the construction of Figure 10 in the following way. Change destination d to d_1, and have node C_m announce destination d_2 (C_m will be node v in the statement of the ASYMMETRY problem). Add the link $\{w, C_m\}$ to E. Node w's import rules prefer the direct route to d_2 while C_m's import rules prefer the route to d_1 that traverses the ASSIGN (n) subgraph. It then follows that an instance of 3-SAT is satisfiable if and only if there is a final state of this system with an asymmetric routing between destinations d_1 and d_2. To complete the proof we observe that checking any state to see if it is a final state with asymmetric routing for two destinations can be done in polynomial time. ■

Theorem 4.3 SOLVABILITY (SD) *is NP-complete.*

Proof: As argued in the previous proof, determining if a state is a final state can be done in polynomial time and hence SOLVABLE (SD) is in NP. We use the construction in the proof of Theorem 4.1 and BAD GADGET from Section 3.1 to define a reduction from 3-SAT to SOLVABILITY (SD). That is, the reduction will be such that the instance of 3-SAT will have a solution if and only if there is a final state for the system.

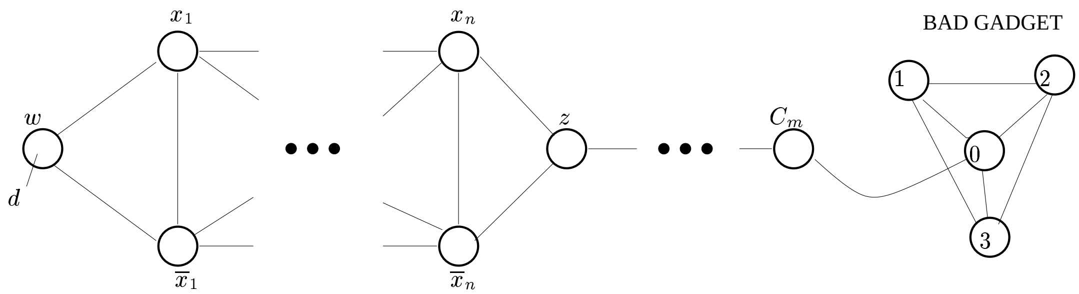

Figure 12: AS graph construction for TRAPPED.

The construction of an instance C of SOLVABILITY (SD) from an instance I of 3-SAT is shown in Figure 11. Node 0 has a policy rule stating that it accepts a route if and only if it is from w. Thus, 0 obtains a route to d the first time it is activated, and this will cause BAD GADGET to diverge unless something is done to interrupt this loop. To accomplish this interruption, we give nodes i, $1 \leq i \leq 3$, additional policy rules that say that they will accept a route from C_m with highest local preference (a value greater than the local preference of any other connection). Therefore if there is an activation sequence that results in C_m having a route to d, then subsequent activations of 1, 2 and 3 result in establishing C_m as their next hop and they never change thereafter. It is easy to check that after executing the activation sequence $p||p'$ as given in the proof of Theorem 11, followed by activating 1, 2, and 3, the system has reached a final state. Any activation sequence that does not result in C_m establishing a route to d will not reach a final state because of the behavior of BAD GADGET. Thus if there is no activation sequence that results in establishing a route from C_m to d then the system is unsolvable. Therefore by previous arguments, I has a satisfying assignment if and only if C_m establishes a route to d, and we have shown that this is true if and only if there is a final state. ■

Corollary 4.4 *(1)* UNSOLVABILITY (SD) *is NP-hard, (2)* SOLVABILITY *is NP-hard, and (3)* UNSOLVABILITY *is NP-hard.*

Proof: Since SOLVABILITY (SD) and UNSOLVABILITY (SD) are complements, (1) follows from Theorem 4.3. Claims (2) and (3) follow from that fact that we have shown special cases of these questions to be NP-hard. ■

Theorem 4.5 TRAPPED *is NP-hard.*

Proof: We can use a construction similar to the one used to prove Theorem 4.3. Modify the AS graph of Figure 11 in the following way. First delete the arcs between C_m and AS 1, 2 and 3. Second, replace the arc between w and the center of BAD GADGET (AS 0) with an arc between C_m and AS 0. Finally, configure BAD GADGET to diverge if and only if C_m obtains a route to d. The resulting AS graph is pictured in Figure 12. This construction now allows us to reduce 3-SAT to TRAPPED. That is, an instance of 3-SAT is satisfiable if and only if the evaluation graph of the BGP system of Figure 12 has a trap. ■

Theorem 4.6 *For each $k > 0$, k-*ROBUST *is NP-hard.*

Proof: Let k be any integer greater than 0. Given an instance I of 3-SAT, we construct a BGP system whose AS graph is pictured in Figure 13. This system has a subgraph equivalent to the REACHABILITY construction used in Theorem 4.1 (Figure 10), where C_m can obtain a route to d if and only if I has a satisfying assignment. This graph is augmented with an instance of BAD GADGET, whose center AS is linked to w. In addition, AS 2 has k paths to w via ASes $w_1, \cdots, w_k$, and AS 3 has an edge directly to C_m.

The BAD GADGET subsystem is configured to diverge if and only if the edge $\{w, 0\}$ is the only edge connecting BAD GADGET to the rest of the graph. That is, AS 2 prefers every path through the w_i's over the path $[1, 0, w]$. AS 3 prefers to reach d through C_m over the path $[2, 0, w]$. Note that this system is always solvable, regardless of whether or not I is satisfiable.

We claim that I is satisfiable if and only if this BGP system is k-robust. Assume that I is satisfiable. Suppose that a subset $E' \subseteq E$ of k edges is removed from the BGP system, and that the resulting system is no longer solvable. This means that every path $p = [2, w_i, w]$, must include exactly one edge from E', otherwise the system would have a solution. But this means that no edges were removed from the rest of the AS graph, so there is a evaluation sequence that results in C_m having a route to d. In this case, the BAD GADGET subsystem will not diverge, since AS 3 will take the a route through C_m to d. Therefore, this system is solvable, and we have a contradiction. We conclude that there cannot exist such a set E', and that the BGP system is k-robust.

In the other direction, suppose that the BGP system of Figure 13 is k-robust. This means that we can remove the k links from AS 2 to the w_i and the system is still solvable. Therefore, AS 3 must have a path to d through C_m, which implies that I is satisfiable. ■

Theorem 4.7 UNIQUE (SD) *is NP-hard.*

Proof: The problem UNIQUE 3-SAT, is the problem of determining if there is a unique solution to a 3-SAT instance, and is know to be NP-complete [11]. Using the construction in Theorem 4.3 to transform instances of UNIQUE 3-SAT to UNIQUE (SD), we can conclude that UNIQUE (SD) is NP-hard. ■

Corollary 4.8 UNIQUE *is NP-hard.*

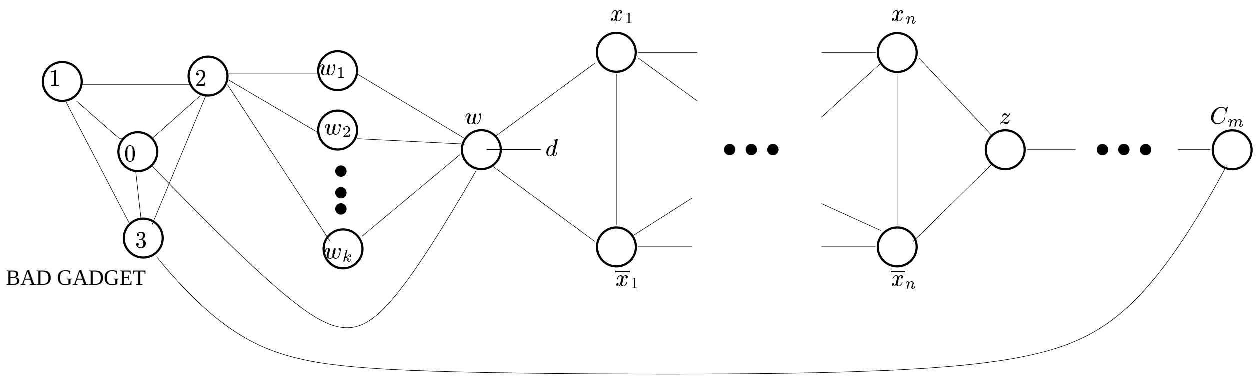

Figure 13: AS graph construction for k-ROBUST.

Theorem 4.9 MULTIPLE (SD) *is NP-complete.*

Proof: Verifying that two states are final states can be done in polynomial time and hence MULTIPLE (SD) is in NP. Note that the construction in the proof of NP-hardness of UNIQUE 3-SAT in [11] takes any instance I of 3-SAT and creates an instance U of UNIQUE 3-SAT that has exactly one solution if I has no solution and multiple solutions if I has one or more solutions. Therefore it is also the case that MULTIPLE 3-SAT, the problem of deciding if a 3-SAT instance has more than one solution, is also NP-hard. Then we can again use the construction in Theorem 4.3, this time to transform instances of MULTIPLE 3-SAT to instances of MULTIPLE (SD) to show that this problem is NP-hard. ■

Corollary 4.10 MULTIPLE *is NP-hard.*

5 What about real-world BGP?

The dynamic behavior of real-world BGP is considerably more complex than that of our model. When BGP peering is established (using TCP), BGP peers initially exchange all route information. After this initial exchange, only incremental "deltas" are exchanged. That is, each BGP border router must store in a local database all routes that it learns from its peers. Changes are accomplished by sending messages that announce new routes or withdraw routes that are no longer reachable. An implementation of BGP may involve many processes running asynchronously, performing tasks such as receiving messages from peers, processing routes according to import policies, choosing best routes, processing best routes according to export policies, and sending messages to peers. For these reasons a more complete model of real-world BGP dynamics requires a rather complex formalization.

In contrast, our simplified model assumes that the local database of route records is computed in one atomic step that includes the real-world operations of importing route records from all peers, selecting best routes, and exporting best routes to all peers. The model dispenses with update messages by assuming that a peer's best routes are "directly visible." In other words, the states in our evaluation graph correspond to states that could be arrived at with real-world BGP after all messages in transit or in queues have been processed.

We chose to present our analysis using the simplified model because it substantially reduced the complexity of stating and proving the complexity results. However it should be noted that despite this simplification, all of these results remain valid for a more complicated message-based mode of real-world BGP. To see this, consider the construction used in the proof of NP-hardness of REACHABILITY. The fact that C_m establishes a route to d if and only if the 3-SAT instance has a solution only depends on satisfying the conditions of the various import rules and is independent of the actual evaluation model. This same observation can be made for the other constructions described in Section 4 since they are based on the same basic construction used for REACHABILITY. Together with the simplifications listed at the beginning of Section 2, this implies that the complexity results of Section 4 provide *lower bounds* for the complexity of the corresponding questions for real-world BGP.

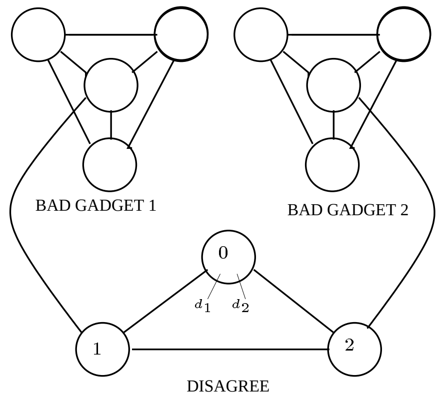

Figure 14: AS graph for INTERFERE.

We do not mean to imply that all types of analysis will carry over from our simplified model to real-world BGP. For example, when multiple destinations are considered, it may be that the set of solutions is model dependent. To illustrate this, consider the BGP system INTERFERE pictured in Figure 14. This system extends the AS graph of PRECARIOUS (Section 3.3), and AS 0 originates two destination, d_1 and d_2. BAD GADGET 2 is attached to AS 2 in the same way that BAD GADGET 1 is attached to AS 1, and it is configured to diverge only if AS 2 accepts the direct route to d_2.

In our simplified model, PRECARIOUS is solvable with the single destination d_1, or with the single destination d_2, but it has no solution for destinations d_2 and d_1 together. This is because when AS 1 or AS 2 update they compute best routes to both destinations. However, with a formalism closer to real-world BGP it would be possible to solve this system by a (somewhat improbable) sequence of events : (1) AS 2 establishes the direct route to d_1, (2) AS 1 establishes the indirect route to d_1. (3) AS 1 establishes the direct the route to d_2, and (4) AS 2 establishes the indirect route for to d_2.

6 Implications and Further Research

The static analysis approach to solving the BGP convergence problem faces two practical challenges. First, autonomous systems currently do not widely share their routing policies, or only publish incomplete specifications. We don't believe this situation will change. Second, even if there were complete knowledge of routing policies, the complexity results presented in this paper show that checking for various global convergence conditions is either NP-complete or NP-hard. Therefore, any approach based on static analysis would most likely rely on heuristics rather than exact solutions.

For these reasons, we believe that a practical solution to the BGP convergence problem must be a *dynamic* one. As we pointed out in the Introduction, route flap dampening [19] does not provide enough information to differentiate between policy-induced route flapping and other sources of routing instability. One possible solution is to *extend* the BGP protocol to carry additional information that would allow policy conflicts to be detected and identified at run time. Such an extension would have to supply network administrators with enough information to identify routing oscillations as being policy-induced, and to identify those autonomous systems whose policies are involved. Routers could then be configured to immediately stop announcing routes involved in policy-induced oscillations. If all routers where configured in this way, then we could guarantee that policy-induced protocol oscillations would not persist. Of course, it is a difficult challenge to design such a dynamic mechanism that is scalable, robust, and compatible with address aggregation.

Several theoretical problems remain open. We do not have a complexity bound for determining if a BGP system is inherently convergent. We also do not know if it is possible for an inherently convergent system to have more than one solution. Note that answers to these questions may depend on the choice of the dynamic evaluation model (see Section 5). Finally, we lack a complete characterization of BGP policy inconsistencies that can give rise to protocol oscillations.

Acknowledgements. We would like to thank Brenda Baker, Anindya Basu, Dave Kristol, Leonid Libkin, Suvo Mittra, Joann Ordille, Jerry Ryan, Bruce Shepherd, and Kannan Varadhan for their helpful comments on drafts of this paper.

References

[1] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizar. Routing Policy Specification Language (RPSL). RFC 2280, 1998.

[2] C. Alaettinoğlu. RAToolSet : A Routing Policy Analysis Tool Set. `http://www.isi.edu/ra/RAToolSet`.

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

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

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

[6] R. Govindan and A. Reddy. An analysis of inter-domain topology and route stability. In *INFOCOMM'97*, 1997.

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

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

[9] C. Huitema. *Routing in the Internet.* Prentice Hall, 1995.

[10] IRR. Internet Route Registy. Internet Route Registy Project, `http://www.merit.edu/radb/docs/irr.html`.

[11] D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 6(2):291–305, 1985.

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

[13] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *INFOCOM'99*, 1997.

[14] V. Paxson. End-to-end routing behavior in the internet. *Transactions on Networking*, 5(5), 1997.

[15] R. Perlman. *Interconnections: Bridges and Routers.* Addison-Wesley, 1995.

[16] Y. Rekhter and T. Li. A Border Gateway Protocol. RFC 1771 (BGP version 4), 1995.

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

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

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