Roma Tre University
Ph.D. in Computer Science and Engineering

Understanding and Detecting BGP Instabilities

Luca Cittadini

Understanding and Detecting BGP Instabilities

A thesis presented by
Luca Cittadini
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

March 2010

Committee:
Prof. Giuseppe Di Battista

Reviewers:
Prof. Lixin Gao
Prof. Olivier Bonaventure

Acknowledgments

The first person I want to thank for literally dragging me this far is my advisor Giuseppe Di Battista, to whom I owe probably everything I know about computer networks. Besides being a great teacher and a helpful advisor, his key contribution throughout my PhD training was enthusiasm for reasearch.

All the people at the Computer Networks Research Lab of Roma Tre University gave a crucial contribution to my PhD in some way. Fabrizio Frati and Patrizio Angelini were so friendly to me that they even managed to have me do some work in the research area of Graph Drawing. Tiziana Refice helped me get started, both from an academic and from a professional point of view. Bernardo Palazzi and Pier Francesco Cortese helped bringing fun in the lab and making it an ideal workplace. Massimo Rimondini, known for his tremendous typing speed and his unsurpassed technical skills, was a continuous spur to improve myself. I owe to Maurizio Pizzonia and Maurizio "Titto" Patrignani a lot of stimulating discussions, ranging from pure graph theory to applied computer networks. I especially want to thank Stefano Vissicchio for his perseverance: I owe him the intuition that gave us new fuel.

I want to thank the whole Intelligent Networks research group at Technische Universität Berlin, and especially Anja Feldmann, for hosting me during my visit and for making it so pleasant. I also need to thank all my other coauthors, which made working together a pleasant and stimulating activity: Randy Bush, Olaf Maennel, Wolfgang Mühlbauer, Steve Uhlig, and Jan Zorz.

A special thank goes to my mum and sister, who supported me when I was overworked by deadlines. This list is surely missing a lot of people, but it is impossible to thank everyone properly in this limited space. One final thank goes to all the people that develop open source software. I used that software extensively throughout my PhD, and it never betrayed my trust. Thank you guys, you are the living proof that passion can be an alternative and even a better reason for life than money.

Contents

Preamble

Communication networks have reached amazing size and complexity nowadays. The Internet, which was born as an experimental network connecting a handful of volunteer research institutes, has grown to become a huge distributed system interconnecting almost 700 millions of hosts at present [ISC09]. As soon as it was clear that computer networks would have driven the information revolution, the Internet drew a lot of interest both from academia and from industry. Moreover, the demand for features that were not envisaged when the Internet was designed grew alongside with the size and complexity of the Internet itself. Routing, that is, finding a path in a network that interconnects a given source to a given destination, also needed to evolve accordingly: as soon as the Internet got into its commercial era, there was a strong demand for routing protocols that supported policies.

Among the wide variety of routing protocols that can be found today in the Internet, the Border Gateway Protocol (BGP) is responsible for connecting large administrative domains (called Autonomous Systems, or ASes), each administering its own network. BGP configuration languages allow network administrators to define fine-grained policies to influence the selection and the dissemination of routes over the network, and is therefore classified as a *policy-based interdomain routing protocol*. BGP policies allow each AS to autonomously configure its network in order, e.g., to minimize the cost of routing traffic, or to optimize delay.

Ideally, BGP was designed to let each administrative domain choose the best route (where "best" obviously has local significance) given the alternatives proposed by neighboring ASes. Unfortunately, as it is often the case in other branches of computer science, many agents that independently pursue a local optimum do not always converge into a global optimum. In particular, it has been shown that there exist sets of BGP policies that cannot be satisfied at the same time, and trap the protocol in infinite oscillations in which

a stable routing choice is never reached. This fact spurred lots of research efforts towards techniques to characterize, discover, mitigate and eliminate BGP instabilities.

This thesis presents novel research contributions as well as related work regarding the characterization and the detection of BGP instabilities under a common framework. We cover both the necessary theoretical background, as well as practical techniques and methodologies to analyze real BGP networks. After having introduced BGP basic notions in Part I, we focus on BGP formal analysis in Part II. First, we tackle the problem of finding a suitable model for studying BGP oscillations. Chapter 2 shows that this is indeed a nontrivial task, as many of the simplifying assumptions that have often been made to ease the analysis provably make the model unable to capture certain kinds of routing instabilities. Besides allowing us to pick the model that is best fit to study oscillations, the insight provided by our study also makes us able to review related work in Chapter 3 with a deeper understanding of the interplay among many different models for BGP.

This thesis makes three main contributions. First, we show in Chapter 4 a sufficient and necessary condition for BGP safety under filtering, that is, the property of a BGP network to have guaranteed convergence under arbitrary filtering of BGP routes. To the best of our knowledge, this is the first complete characterization of safety under filtering. We exploit this finding to show a debugging technique that is able to spot the potential trouble points of a network by just analyzing two different routing states.

Second, we study the possibility of manipulating internal BGP (iBGP) attributes. Chapter 5 shows that, while in general such a practice exacerbates the BGP stability problem, adherence to simple guidelines ensures BGP stability while still providing some benefits in terms, e.g., of traffic engineering capabilities.

Third, we devise and implement an algorithm which is able to tell whether a given BGP network is stable. This algorithm is provably free from false positives, and it is able to pinpoint the trouble points of a potentially unstable network. We show in Chapter 6 that this algorithm, together with techniques to perform some preprocessing on BGP networks, can be implemented efficiently enough to deal with Internet scale BGP topologies as well as very large iBGP networks. Finally, in Chapter 7 we propose a BGP monitoring system that is able to collect BGP data in such a way to enable the analysis of what-if scenarios by applying the same techniques devised in Chapter 6.

Conclusions are drawn in Part 7.7, where proposals for further research efforts are also presented.

Part I

Background

Chapter 1

Internet Routing and BGP

1.1 Internet Architecture

The huge Internet network has been created by the progressive and incremental interconnection of a large number of smaller networks, each containing a very large number of links, end systems, and intermediate systems. In order to manage the complexity caused by such a sheer size, the Internet is partitioned into domains called *Autonomous Systems* (ASes), and each AS is under the control of a single administrative entity. ASes join the Internet for a wide variety of purposes. The vast majority of the ASes are merely interested in getting access to the network, e.g., in order to access the content that is provided over the Internet. These domains are called *stub* domains. A relatively small fraction of the ASes, instead, makes business by providing access to the Internet, that is, by ensuring that packets can traverse the network and reach their destination. In a sense, such enterprises, called Internet Service Providers (ISPs), sell the service of transiting packets across their own network infrastructure, and are therefore called *transit* ASes: usually, when a stub AS connects to an ISP, the stub AS purchases connectivity, e.g., by paying the ISP a fixed amount of money for each bit of information that is sent across the network.

Even if some ISPs are very large and geographically distributed across the world, in general a packet that travels over the Internet from a source to a destination, e.g., to request a web page, transits across multiple ISP. Transit ASes can interconnect their networks using a number of *policies*. In [Gao01], Gao classified such policies in two major types: **customer-provider** relationships and **peer-to-peer** relationships. In the customer-provider case, a customer

AS purchases transit service from a provider AS, as it happens when a stub AS connects to a transit AS. In the peer-to-peer case, instead, the two ASes simply exchange traffic free of charge. This kind of relationship is becoming increasingly popular in the Internet, as more and more ASes realize that, when traffic is approximately balanced between inbound and outbound flows, the costs induced by an accurate billing system overcome the profits that could be made by selling transit service. Also, the recent growth and pervasive presence of content providers in the Internet (e.g., YouTube, Google, Facebook, Wikipedia) has made peer-to-peer connections more attractive because it helps keep the latency low, in order to deliver a better experience to the end user.

The fact that some connections are customer-provider while some other are peer-to-peer makes finding an optimal route from a source to a destination in the Internet quite a challenging task, since each AS has its own point of view about how an "optimal" route should look like. Since different constraints are to be taken into account at each different AS, the process of finding a route from a source to a destination in the Internet is divided into two hierarchical sub-problems: routing within an AS (*intradomain* routing) and routing among ASes (*interdomain* routing).

While intradomain routing must only deal with the network topology and is usually solved by an Interior Gateway Protocol (IGP) that exploits a relatively simple algorithm (e.g., shortest path algorithms), interdomain routing needs a more complex protocol since it also has to deal with the policies that each AS autonomously specifies. For example, an AS does not want to forward traffic coming from a non-paying peer-to-peer link over a paid connection to its provider. For this reason, the set of routes that are accessible from peer-to-peer connections needs to be controlled via specific policies. The need to support interdomain routing policies is what pushed the Border Gateway Protocol (BGP) [RLH06] to become the de facto standard interdomain routing protocol since the early nineties.

1.2 BGP: a Protocol for Interdomain Routing

The first thing to understand about BGP is that not all the routers within an AS need to be involved in the protocol: in fact, since BGP is designed for interdomain routing, in principle only the routers that are connected to foreign ASes (*border routers*) are required to speak BGP. A *peering* session between two BGP speakers, called *peers*, is a TCP connection that is used to exchange *BGP messages*.

The task of BGP is essentially to disseminate information about the reachability and the location of contiguous blocks of IP addresses known as *prefixes*. The AS that owns a certain network announces the availability of the corresponding prefix by sending a BGP message to the neighboring ASes. For this reason, we call such AS the *origin* of the prefix. BGP then distributes the reachability information while preserving the policy constraints defined at each AS and embedded in the configuration of BGP speakers. BGP is a **path-vector** protocol in the sense that each BGP speaker prepends its own AS number in the BGP message before passing it to neighboring ASes. This way, each BGP message carries the ordered list of traversed ASes (that is, the *AS-path*). The presence of the AS-path makes loop detection extremely easy in BGP: each BGP router simply discards the BGP message whenever the AS-path already contains its own AS number. Traffic destined to a certain prefix is forwarded to the origin AS by simply traveling along the AS-path in reverse order, that is, traffic takes the same route as the BGP message itself, but in the opposite direction.

BGP messages are distinguished into two types: *announcements* advertise the reachability of a prefix, while *withdrawals* communicate that a prefix has become unreachable. Each BGP message (both announcements and withdrawals) contains a set of prefixes (at least one), and for each prefix a set of *attributes* associated with it. An attribute can be classified along several dimensions:

- **Well-known** or **optional**. Well-known attributes are expected to be supported by any implementation of BGP, while optional attributes may be supported only by a specific subset of them. Since BGP speakers are instructed to never tamper with an optional BGP attribute they do not understand, optional attributes are guaranteed to be modified only by BGP speakers that support them.

- **Mandatory** or **discretionary**. Mandatory attributes are required to be present in every BGP message, while discretionary attributes are not.

- **Transitive** or **non-transitive**. Transitive attributes have a global scope and must be included (possibly after manipulation) when propagating a received BGP message to a BGP router in another AS. Non-transitive attributes have only local significance, and must be dropped as soon as the BGP messages needs to be sent to a BGP router in another AS.

RFC 4271 [RLH06] defines the following BGP attributes

- `AS-path` (well-known, mandatory, transitive): it is the sequence of ASes along which the BGP message was forwarded. Because of the way BGP operates, this is also the sequence of ASes traversed by traffic destined to the prefix (remember that traffic flows in the opposite direction with respect to BGP announcements).

- `origin` (well-known, mandatory, transitive): it signals whether the prefix has been injected into BGP due to *(i)* a specific statement in the router configuration, *(ii)* redistribution from an intradomain routing protocol, or *(iii)* redistribution from the older EGP protocol. The third case is almost never encountered in the Internet.

- `next-hop` (well known, mandatory, transitive): contains the IP address of the router that should be used to forward traffic destined to the prefix.

- `multi-exit-discriminator`, also known as `MED` (optional, discretionary, non-transitive): when present, it influences the choice among multiple exit points belonging to the same AS.

- `local-preference` (well known, non-transitive): this attribute is mandatory for all BGP updates destined to peers in the same AS, while it is not allowed in BGP updates destined to foreign ASes. The `local-preference` attribute allows a BGP router to indicate the relative degree of preference that is locally associated with the route contained in the BGP update.

- `atomic aggregate` (well-known, discretionary, transitive): when present, it indicates that the route contained in the BGP update is the result of aggregating multiple contiguous prefixes that share the same attributes.

- `aggregator` (optional, discretionary, transitive): when present, it indicates the AS number and the IP address of the last BGP router that aggregated the prefix.

- `community` (optional, discretionary, transitive): this attribute does not have any defined semantics. It is just a way to associate a set of tags (each tag consists of a pair of integer values) to a route. The common way of using it is to group together sets of BGP announcements that should be assigned a similar degree of preference, but many ISPs also allow their customers to set specific community values to influence the behavior of the provider, e.g., for traffic engineering purposes.

Step	Criterion
1	Discard routes having lower `local-preference` than the highest
2	Discard routes having longer `AS-path` length than the shortest
3	Discard routes having higher `origin` than the lowest
4	Among the routes received from the same AS neighbor, discard those having higher `MED` than the lowest
5	Prefer routes learned via eBGP to those learned via iBGP
6	Prefer routes with lower IGP metric to the egress point
7	Prefer the route announced by the BGP router with the lowest `router-id` (i.e., IP address)

Table 1.1: Steps in the BGP decision process.

Every BGP speaker collects routes from its peers and stores routing information into a special table called *Routing Information Base* (RIB). The RIB is managed as follows. Whenever a BGP message is received from a peer, the route (that is, the prefix and all the associated attributes) is stored in a data structure called **Adj-RIB-In** and is then processed according to a set of *import policies*. Import policies might mandate deletion or attribute manipulation when specific criteria are met. Loop detection, which is simply accomplished by discarding any route carrying an `AS-path` attribute which already contains the locally defined AS number, is also performed at this stage. For each prefix, among all the entries in the Adj-RIB-In that have passed the import policy application, a deterministic decision process is triggered in order to select the best route, which will be the only one that is actually used to forward traffic (i.e., it will be pushed to the router's forwarding plane). Table 1.1 summarizes the decision process of BGP.

The `next-hop` attribute of the best route is then used to update the **Loc-RIB** data structure, which in turn is used to update the router's forwarding table. If, after the BGP decision process, the newly selected best route differs from the one that was previously selected, then the BGP router applies *export policies* to the best route. Similarly to import policies, export policies might mandate deletion of the route or attribute manipulation. The router then stores the modified routes in a data structure called `Adj-RIB-Out`, which collects the BGP messages that are scheduled to be sent to the peers of the router. This way, BGP supports export policies that are possibly different on a per-neighbor basis. Figure 1.1 provides a visual description of the building blocks that make up a BGP router.

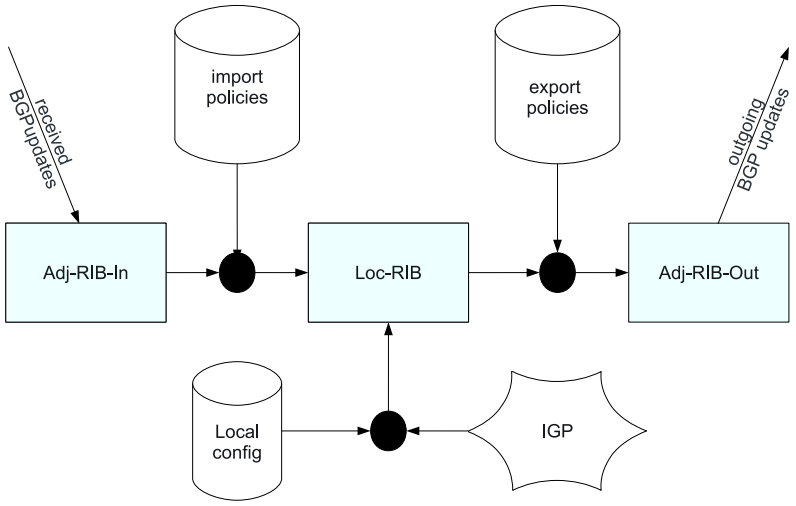


Figure 1.1: Information flow among the building blocks of a generic BGP router.

The behavior of a BGP router varies significantly depending on whether it is exchanging routing information with peers belonging to a different AS or it is talking to other BGP speakers in the same AS. These two "flavors" of BGP are denoted as *external* BGP (**eBGP**) and *internal* BGP (**iBGP**) respectively. The most interesting difference is the presence of the `local-preference` attribute, which is included in iBGP messages while it is never present for eBGP messages. Since import policies usually manipulate the `local-preference` attribute in order to influence the decision process, this makes it easier to disseminate routing information in iBGP together with degrees of preference, enabling consistent AS-wide policy enforcing. Since deep knowledge of iBGP-specific features is not necessary to understand most of the contents presented in this thesis, we defer a more precise introduction to Chapter 5, which focuses explicitly on iBGP-specific issues.

Part II

Formal Analysis of BGP Stability

Chapter 2

Modeling BGP Policies and Dynamics*

2.1 Introduction

The first step towards understanding and developing a formal analysis for a routing protocol is obviously finding a suitable model. In this thesis, we are especially interested in models that are able to capture routing instabilities. Several approaches have been proposed to model BGP and to study its dynamic properties. A pioneering work on this subject is in [VGE00], that proposed the *return graph* model and used it to show that there are collections of routing policies that together can cause BGP to diverge. Another family of models, based on routing algebras, have been shown in, e.g., [kC06, GS05, Sob05]. In this thesis we adopt the *Simple Path Vector Protocol* (SPVP) framework [GSW99], that is an abstraction that captures the underlying semantics of any policy-based path vector protocol such as BGP. This model is particularly important as it is able to effectively represent the static portion of BGP, i.e., routing policies, as well as its dynamic properties. For this reason, SPVP is the reference point of most of the scientific contribu-

*Part of the material presented in this chapter is based on the following publication

- L. Cittadini, G. Di Battista, M. Rimondini. How Stable is Stable in Interdomain Routing: Efficiently Detectable Oscillation-Free Configurations. Technical Report RT-DIA-132-2008, Dept. of Computer Science and Automation, Roma Tre University, 2008.

tions on BGP stability. The relationship between this formalism and algebraic approaches is explored in [JR05, kCGG06].

Besides defining the notation that will be used throughout the thesis, in this chapter we revisit a wide range of alternative models that have been used in prior work. Our results prove that SPVP is the most general model for policy-based path vector protocols: by applying any of the simplifications that have been previously proposed in the literature, the resulting model is provably able to capture only a strictly smaller subset of routing oscillations with respect to the original version of SPVP. For this reason, in the rest of this thesis we will only refer to the original version of SPVP whenever we need a formal model for policy-based path vector protocols.

The rest of the chapter is organized as follows. Sections 2.2 and 2.3 define the static and dynamic part of the SPVP model, respectively. Section 2.4 summarizes several variants of SPVP that have been proposed in the literature, and classifies them in a taxonomy. Finally, in Section 2.5 we formally prove that the original version of SPVP captures strictly more routing oscillations than any of the simplifications that have been proposed in the literature.

2.2 Modeling BGP Policies

In this section, we define a formal model that captures the expressiveness of BGP policy configuration. Formally, we model a set of BGP policies as an SPVP *instance*.

Let $G = (V, E)$ be an undirected graph, with vertex set $V = \{0, 1, \ldots, n\}$ and edge set E. The graph G is used to represent the Internet topology at the level of ASes. Vertices in V correspond to ASes, while edges in E correspond to adjacency relationships between ASes (i.e., BGP *peerings*). Vertex 0 is special in that it is the destination every other vertex tries to establish a path to. We denote by peers(u) the set of neighbors of vertex u, that is, the set of vertices $v|(u, v) \in E$.

Paths play an important role in this model. A *path* P in G is a sequence of $k + 1$ vertices $P = (v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$, $v_i \in V$, such that $(v_i, v_{i-1}) \in E$ for $i = 1, \ldots, k$. Vertex v_{k-1} is the *next hop* of v_k in P. The *empty path*, denoted by ϵ, represents unreachability of the destination. The *concatenation* of two nonempty paths $P = (v_k\ v_{k-1}\ \ldots\ v_i)$, $k \geq i$, and $Q = (v_i\ v_{i-1}\ \ldots\ v_0)$, $i \geq 0$, denoted as PQ, is the path $(v_k\ v_{k-1}\ \ldots\ v_i\ v_{i-1}\ \ldots\ v_0)$. We assume that $P\epsilon = \epsilon P = \epsilon$, that is, the empty path can never extend or be extended by other paths.

Since BGP manages each prefix independently, we can study the stability separately for each destination. For this reason, in an SPVP instance each vertex in $V - \{0\}$ attempts to establish a path to a single vertex 0. Each vertex $u \in V$ is assigned a set of *permitted paths* $\mathcal{P}^u$. All the paths in $\mathcal{P}^u$ are simple (i.e., without repeated vertices), start from u and end in 0, and represent the paths that u can use to reach 0. The empty path represents unreachability of 0 and is permitted at each vertex $u \neq 0$. Let $\mathcal{P}^0 = \{(0)\}$, that is, vertex 0 can reach itself only directly. Let $\mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u$.

For each vertex $u \in V$, a *ranking function* $\lambda^u : \mathcal{P}^u \to \mathbb{N}$ determines the relative level of preference $\lambda^u(P)$ assigned by u to path P. If $P_1, P_2 \in \mathcal{P}^u$ and $\lambda^u(P_2) < \lambda^u(P_1)$, then P_2 is *preferred* over P_1. Let $\Lambda = \{\lambda^u | u \in V\}$. Ranking functions in Λ are used to describe BGP routing preferences.

The following conditions hold on the paths, for each vertex $u \in V - \{0\}$:

(i) $\epsilon \in \mathcal{P}^u$ (empty path is always permitted)

(ii) $\forall P \in \mathcal{P}^u, P \neq \epsilon$: $\lambda^u(P) < \lambda^u(\epsilon)$ (empty path is the last resort);

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

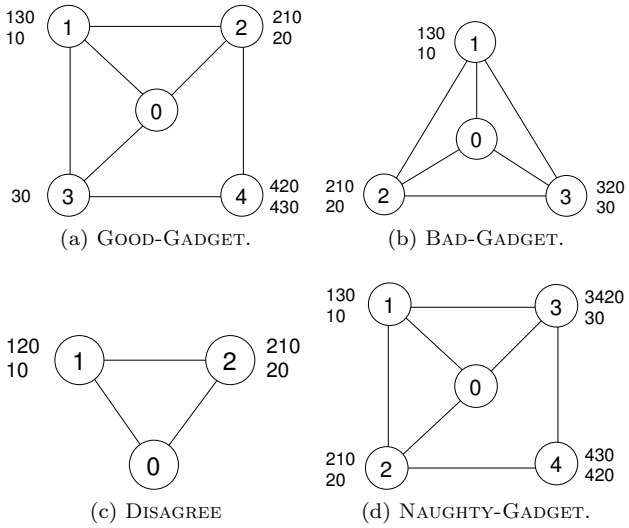
An SPVP instance S is a triple $S = (G, \mathcal{P}, \Lambda)$. See an example in Fig. 2.1a. The graphical convention adopted is the same as in [GSW02] and will be used throughout the thesis. In this convention, each vertex u is equipped with a list of paths representing $\mathcal{P}^u$, sorted by increasing values of λ^u. The empty path and $\mathcal{P}^0$ are omitted for brevity. For example, the list besides vertex 2 specifies that 2 can use paths (2 1 0) and (2 0) to reach 0 ($\mathcal{P}^2$ consists of those two paths) and prefers (2 1 0).

Since vertices and edges that are not used by any path in $\mathcal{P}$ cannot influence the stability of the protocol, we assume that the *size* of S is the size of $\mathcal{P}$.

2.3 Modeling BGP Dynamics

An SPVP instance is a simple static model for BGP policies. However, in order to understand stability properties of the protocol, we also need a dynamic model that deals with routing messages and routing table updates.

A *path assignment* π is a function that maps each vertex $u \in V$ to a permitted path $\pi(u) \in \mathcal{P}^u$. This represents the fact that vertex u is using path $\pi(u)$ to reach 0. We have that $\pi(0) = (0)$ and, if $\pi(u) = \epsilon$, then u cannot reach



(a) Good-Gadget. (b) Bad-Gadget.

(c) Disagree (d) Naughty-Gadget.

Figure 2.1: Good and bad SPVP instances. (a) an instance that has a unique guaranteed stable state. (b) an instance that has no stable states. (c) an instance that has two distinct stable states (Table 2.3) but still admits an oscillation (Table 2.2). (d) an instance that has a stable state as well as a permanent oscillation (Theorem 3.4).

vertex 0. A path assignment models a routing choice made at every vertex in the network, and as such is also referred to as the *state* of the network.

In SPVP vertices asynchronously exchange messages (*announcements*) containing paths to 0 by running the algorithm in Figure 2.2. An announcement from vertex v which advertises ϵ models unreachability of 0 from v. We assume that edges introduce a finite delay on message delivery. Each vertex u keeps in a *routing information base* $\text{rib}_t(u)$ the path it adopts at time t to reach vertex 0. If a vertex u receives from a neighbor w at time t an announcement containing a path P, first of all u checks whether $(u)P$ is permitted, namely if $(u)P \in \mathcal{P}^u$. If this is the case, u puts $(u)P$ into a data structure called $\text{rib-in}_t(u \Leftarrow w)$, which is used to store the latest path received from neighbor w. Otherwise, if $(u)P$ is not permitted (i.e, $(u)P \notin \mathcal{P}^u$), u puts ϵ in $\text{rib-in}_t(u \Leftarrow w)$. At this point, u checks whether the currently selected path, stored in $\text{rib}_{t-1}(u)$, is the currently available best path. If this is not the case, u selects the best ranked

process spvp(v)
1: **while receive** P **from** u **do**
2: rib-in$_t(v \Leftarrow u) := P$
3: rib$_t(v) :=$ best$_t(v)$
4: **if** rib$_t(v) \neq$ best$_{t-1}(v)$ **then**
5: **for all** $v \in$ peers(v) **do**
6: **send** rib$_t(v)$ **to** v
7: **end for**
8: **end if**
9: **end while**

Figure 2.2: A distributed asynchronous algorithm (SPVP) for modeling the dynamic behavior of BGP.

path among those in all its rib-in$_t$ data structures and stores it in rib$_t(u)$. We refer to this path as best$_t(u) = \underset{w|(u,w)\in E}{\arg\min}\ \lambda^u(\text{rib-in}_t(u \Leftarrow w))$. Afterwards, u announces best$_t(u)$ to all its neighbors $v \in$ peers(u).

In a real network there are a lot of factors that affect the timing of routing updates, e.g., link delays, router queues, etc. However, while modeling the exact timing might be interesting to study protocol convergence time at a fine-grained detail level, exact timings are largely irrelevant to study the behavior of policy-based path vector protocols. Instead, what is needed is a simple way to represent the order in which routing messages are processed, rather than the exact time at which they are received. In SPVP, the order in which vertices exchange messages is modeled by *activation sequences*. An activation sequence [GW00] $\sigma = (A_1\ \ldots A_i \ldots)$ is a (possibly infinite) sequence where A_t is a set representing the announcements that are received by vertices at time t. Set A_t contains an ordered pair $(u,v)|(u,v) \in E$ for each vertex v that processes a message from u at time t. We say that edge (u,v) is *activated* at time t. An activation sequence is *fair* if any edge $(u,v) \in E$ is eventually activated after u has sent a message to its neighbors. We are only interested in fair activation sequences, as we are assuming that links can only delay a message by a finite amount of time. Given an SPVP instance S, we say that an activation sequence σ on S *leads to* path assignment π_{t_2} *starting from* path assignment π_{t_1}, denoted by $\pi_{t_1} \overset{\sigma}{\rightsquigarrow} \pi_{t_2}$, if, after activating edges according to σ, S changes its state from π_{t_1} to π_{t_2}.

As an example, consider again Fig. 2.1a. Before SPVP starts working,

we assume $\text{best}_t(u) = \epsilon$ for all vertices but 0. A possible activation sequence is $\sigma = (A_1\ A_2)$ with $A_1 = \{(0,1),(0,2)\}$, and $A_2 = \{(0,3),(1,2)\}$. Namely, at time $t = 1$ vertices 1 and 2 simultaneously receive an announcement from 0, stating that vertex 0 is directly reachable. Hence, vertex 2 inserts into its rib-in path (2 0), i.e., $\text{rib-in}_1(2 \Leftarrow 0) = (2\ 0)$. Similarly, for vertex 1 $\text{rib-in}_1(1 \Leftarrow 0) = (1\ 0)$. Since vertices 1 and 2 have no other alternatives, they both select the direct path as the best route to 0, i.e., $\text{best}_1(1) = (1\ 0)$ and $\text{best}_1(2) = (2\ 0)$. Because of A_2, at time $t = 2$ vertex 3 also receives an announcement from 0, sets $\text{rib-in}_2(3 \Leftarrow 0) = (3\ 0)$, and computes its best path $\text{best}_2(3) = (3\ 0)$. At the same time, vertex 2 receives an announcement from 1, which advertises path (1 0). Hence, vertex 2 sets $\text{rib-in}_2(2 \Leftarrow 1) = (2\ 1\ 0)$, and computes a new best path $\text{best}_2(1) = (2\ 1\ 0)$.

As SPVP operates within the network, the routing evolves through different path assignments π_t, where $\pi_t(u) = \text{rib}_t(u)$, until a stable path assignment is reached (in this case, we say that SPVP *converges* to that path assignment). A path assignment π_t is *stable* if, for each $u \in V$, $\pi_t(u) = \text{best}_t(u)$. Intuitively, a stable path assignment satisfies all vertices in the network since none of them can switch to an alternative path that is ranked better than the one it is currently using. For this reason, once a stable path assignment is reached, no further messages are generated in the network. For this reason, any activation sequence that leads to a stable path assignment is forcedly finite.

2.4 A Taxonomy of Related Work

Besides SPVP, several variations have been proposed in the literature [GR00, VGE00, BOR+02, CGM03, FJB07]. All these SPVP variants try to simplify the original version of SPVP in order to reduce the complexity of the model. In this section, we propose a taxonomy of existing approaches, which is concisely summarized by the classification shown in Tab. 2.1.

Edge and Vertex Activation Sequences

As we said above, in the original version of SPVP an edge (u, v) is activated when vertex v receives and processes a message from its neighbor u. A common relaxation, used in [BOR+02, GSW99, GR00, FJB07], tries to abstract the semantics of message passing that is encompassed in SPVP by only considering aggregate steps where vertices (instead of edges) are activated. In this variant, when a *vertex* u is *activated to process*, u collects all the selected routes from its neighbors (namely, it collects paths $\text{rib}_t(v)$ for each neighbor v of u). Then,

u selects the best path among the alternatives it collected and updates its data structure $\text{rib}_{t+1}(u)$. Basically, u executes the algorithm in Fig. 2.2, assuming that a message is simultaneously received from every vertex v in $\text{peers}(u)$.

For the sake of completeness, it is quite natural to consider vertex activation from the opposite perspective. We say that a *vertex* u is *activated to send* at time t if u sends its current best path $\text{rib}_t(u)$ to all its neighbors, which are supposed to receive $\text{rib}_t(u)$ simultaneously. Then, for every vertex $v \in \text{peers}(u)$, a recomputation of the best path is triggered (Steps 2, 3 of the algorithm shown in Figure 2.2).

Observe that vertex activation sequences are special classes of edge activation sequences in which constraints are applied on the sequence of activated edges. An activation sequence where vertices are activated to send can be mapped to an edge activation sequence in which each vertex activation $A_i = v$ corresponds to a sequence of activations $A_{i_k} = (v, u_k)$ for each $u_k \in \text{peers}(v)$. A similar argument applies to an activation sequence where vertices are activated to process. In the latter case, pairs $A_{i_k} = (u_k, v)$ are activated for each vertex activation $A_i = v$.

Modeling Memory at Vertices

Another possible variant of the basic SPVP model is the one in which there is no rib-in$_t$ [VGE00, CGM03, FJB07]. In this case, each vertex v only stores its current best path and computes its new best paths directly referring to the best choices of its neighbors. Set $\text{choices}_t(v)$ would then be redefined in the following way: $\text{choices}_t(v) = \{(v,u)P \in \mathcal{P}^v | P = \text{best}_{t-1}(u)\}$.

Consider that, if vertices are activated to process, there is no need to consider a rib-in$_t$. In fact, every time a vertex $v \in V$ is activated, it immediately refreshes $\text{choices}_t(v)$, thus replacing any previously known path.

On the other hand, the absence of rib-in$_t$ forces a vertex to query all its neighbors for each computation of a new best path. This corresponds to activating vertices to process. As an alternative, the absence of a rib-in$_t$ can be compensated by forcing vertices to continuously send update messages, for example exploiting a timeout [CGM03].

Simultaneousness

The original version of SPVP allows activations to be simultaneous. As a further degree of freedom, we distinguish between models that admit simultaneous activations (i.e., $|A_i| \geq 1$) [BOR$^+$02, GSW99, GR00, VGE00, CGM03]

	Activations	RIB	Simult.
[GW00, GGR01, GSW02]	Edges	Yes	Yes
[FJB07]	Vertices, to process	No	No
[VGE00, CGM03]	Edges	No	Yes
[BOR$^+$02, GSW99, GR00]	Vertices, to process	Yes	Yes

Table 2.1: A taxonomy of existing models for path vector protocols.

and models that only allow a single edge (or vertex) to be activated at a time [FJB07] (i.e., $|A_i| = 1$).

2.5 Choosing a model

Although modeling is a necessary step to study BGP stability, there is little understanding about how the differences between models map on the ability to study BGP stability. Throughout the thesis, we will consider the original version of SPVP in which edges are activated, a local rib-in$_t$ is maintained by each vertex, and simultaneous activations are allowed.

In the following, we motivate such a choice by showing that the original version of SPVP cannot be simplified along any dimension without impacting the ability of the model to capture routing oscillations.

For the sake of clarity, in the following we will specify activation sequences using a tabular notation as in Tab. 2.2, where each row corresponds to an activation, the first column specifies activated vertices or edges, and the remaining columns represent the current rib-in$_t$ at each vertex, with the currently selected best path highlighted using italic face. The initial state is assumed to be $\pi_0(v) = \epsilon \ \forall v \in V - \{0\}$.

Let SPVP-ns be the variation of SPVP that does not allow simultaneous activations. The following theorem shows that relaxing SPVP by not considering simultaneous activations impacts the capability of the model to capture oscillations.

Property 2.1 SPVP *captures any oscillation captured by* SPVP*-ns. The converse does not hold.*

Proof: Trivially, non-simultaneous activation sequences can always be mapped to simultaneous edge activation sequences. On the other hand, DISAGREE (Fig. 2.1c) provides an example in which simultaneuosness is needed to trigger

t	A_t	1	2
1	$\{(0,1),(0,2)\}$	*(1 0)*	*(2 0)*
2	$\{(1,2),(2,1)\}$	*(1 2 0)* (1 0)	*(2 1 0)* (2 0)
3	$\{(1,2),(2,1)\}$	*(1 0)*	*(2 0)*

Table 2.2: An oscillating fair edge activation sequence for DISAGREE (Fig. 2.1c). The columns of the table are the time instants, the set of activated edges, and the rib-in$_t$ of each vertex, with the currently selected best path highlighted in italic face.

vertex	0	1	2
stable state 1	(0)	(1 0)	(2 1 0)
stable state 2	(0)	(1 2 0)	(2 0)

Table 2.3: Two distinct stable states for DISAGREE (Fig. 2.1c).

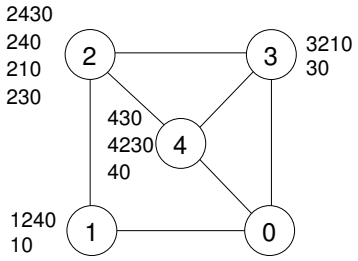


Figure 2.3: BLEEDIN-EDGE: An instance of SPVP for which a fair oscillation exists only in the edge activation model.

t	A_t	1	2	3	4
1	$\{(0,1),(0,3),(0,4)\}$	$(1\ 0)$	ϵ	$(3\ 0)$	$(4\ 0)$
2	$\{(3,2)\}$	$(1\ 0)$	$(2\ 3\ 0)$	$(3\ 0)$	$(4\ 0)$
3	$\{(2,4),(4,2)\}$	$(1\ 0)$	$(2\ 4\ 0)$ $(2\ 3\ 0)$	$(3\ 0)$	$(4\ 2\ 3\ 0)$ $(4\ 0)$
4	$\{(1,2),(2,1)\}$	$(1\ 2\ 4\ 0)$ $(1\ 0)$	$(2\ 4\ 0)$ $(2\ 1\ 0)$ $(2\ 3\ 0)$	$(3\ 0)$	$(4\ 2\ 3\ 0)$ $(4\ 0)$
5	$\{(4,2)\}$	$(1\ 2\ 4\ 0)$ $(1\ 0)$	$(2\ 1\ 0)$ $(2\ 3\ 0)$	$(3\ 0)$	$(4\ 2\ 3\ 0)$ $(4\ 0)$
6	$\{(2,3)\}$	$(1\ 2\ 4\ 0)$ $(1\ 0)$	$(2\ 1\ 0)$ $(2\ 3\ 0)$	$(3\ 2\ 1\ 0)$ $(3\ 0)$	$(4\ 2\ 3\ 0)$ $(4\ 0)$
7	$\{(3,4),(4,3)\}$	$(1\ 2\ 4\ 0)$ $(1\ 0)$	$(2\ 1\ 0)$ $(2\ 3\ 0)$	$(3\ 2\ 1\ 0)$ $(3\ 0)$	$(4\ 2\ 3\ 0)$ $(4\ 0)$
8	$\{(1,2),(3,2),(4,2)\}$	$(1\ 2\ 4\ 0)$ $(1\ 0)$	ϵ	$(3\ 2\ 1\ 0)$ $(3\ 0)$	$(4\ 2\ 3\ 0)$ $(4\ 0)$
9	$\{(2,1),(2,3),(2,4)\}$	$(1\ 0)$	ϵ	$(3\ 0)$	$(4\ 0)$

Table 2.4: An oscillating fair edge activation sequence for BLEEDIN-EDGE (Fig. 2.5).

an oscillation. Let $e \in \{(1,2),(2,1)\}$ be the first edge that is activated, at time t, between vertices 1 and 2. Note that the activation of edge $e = (u, v)$ can only be triggered by a previous activation of edge $(0, u)$. This, in turn, implies that path $(u\ 0) \in \text{choices}_t(u)$. Hence, after activating e, $(v\ u\ 0)$ enters $\text{choices}_t(v)$, that forces $\text{best}_t(v) = (v\ u\ 0)$. Since this leads to any one of the two stable states described in Tab. 2.3, any further activation has no effect. By contrast, Tab. 2.2 shows that DISAGREE admits a fair oscillation if simultaneous activations are allowed. □

We already observed in Section 2.4 that edge activation sequences are more general than vertex activation sequences, regardless of the semantic of the activation of a vertex. Theorem 2.1 shows that relaxing SPVP by not considering the activation of single edges impacts the capability of the model to capture oscillations. To prove the theorem we need the following preliminary lemmas that exploit the instance BLEEDIN-EDGE in Fig. 2.5.

Let SPVP-vp (SPVP-vs) be the variant of SPVP in which vertices are activated to process (to send).

Lemma 2.1 *Consider the* SPVP *instance* BLEEDIN-EDGE. *Independently on the activation sequence, if path* (1 0) *enters* $choices_{t'}(1)$ *at time* t', *then* $(1\ 0) \in choices_t(1)\ \forall t \geq t'$. *The same also holds for paths* (4 0) *and* (3 0).

Proof: The statement follows from $\mathcal{P}^0 = \{(0)\}$. □

Lemma 2.2 *Consider the* SPVP *instance* BLEEDIN-EDGE. *If vertices are activated to send, no vertex activation sequence and no time t exist such that* $\pi_t(4) = (4\ 2\ 3\ 0)$.

Proof: For vertex 4 to select (4 2 3 0) it is required that 2 selects (2 3 0) first, which in turn requires vertex 3 to be activated at least once. Now, once 3 is activated, vertex 4 can immediately select its best path (4 3 0), and will be unable to select (4 2 3 0) in further steps. Even if we assume that there exists a time instant t such that the rib-in$_t$ at vertex 4 already contains (4 2 3 0), we need 3 to withdraw (3 0), which is preferred at 4. Since, by Lemma 2.1, vertex 3 cannot withdraw (3 0) by announcing ϵ, then 3 must announce (3 2 1 0), which can only happen if 2 picks (2 1 0) as its best and is activated beforehand, thus actually removing (4 2 3 0) from the rib-in$_t$ of vertex 4. □

Lemma 2.3 *Consider the* SPVP *instance* BLEEDIN-EDGE. *If vertices are activated to process, no vertex activation sequence allows vertex* 4 *to select* (4 2 3 0) *at any time t.*

Proof: We prove the assertion by contradiction. Let t be the first time at which vertex 4 selects (4 2 3 0). This implies $\pi_{t-1}(2) = (2\ 3\ 0)$ and $\pi_{t-1}(3) = (3\ 2\ 1\ 0)$. In fact, by Lemma 2.1, vertex 3 can never announce ϵ after its first activation, and $\pi_{t-1}(2) = (2\ 3\ 0)$ implies that 3 was already activated before time $t-1$. Let $t' < t$ and $t'' < t$ be the instants of the last activation, before t, of vertices 2 and 3, respectively. If $t' < t''$, vertex 3 would be unable to select path (3 2 1 0), contradicting $\pi_{t-1}(3) = (3\ 2\ 1\ 0)$. On the other hand, if $t'' < t'$, vertex 2 would be unable to select path (2 3 0). Then it must be $t' = t''$, i.e., vertices 2 and 3 were activated simultaneously. This, in turn, implies $\pi_{t'-1}(2) = (2\ 1\ 0)$, $\pi_{t'-1}(1) = (1\ 2\ 4\ 0)$ and $\pi_{t'-1}(3) = (3\ 0)$. Note that, by Lemma 2.1, it cannot be $\pi_{t'-1}(1) = \epsilon$, as $\pi_{t'-1}(2) = (2\ 1\ 0)$. Moreover, since activating vertex 2 at t' will result in $\pi_{t'}(2) = (2\ 3\ 0)$, we must have $\pi_{t'-1}(4) \neq (4\ 3\ 0)$ and $\pi_{t'-1}(4) \neq (4\ 0)$. This means that $\pi_{t'-1}(4)$ can be either ϵ or (4 2 3 0). The former case contradicts Lemma 2.1, since $\pi_{t'-1}(1) = (1\ 2\ 4\ 0)$ implies that 4 was activated before $t'-1$. The latter one contradicts the hypothesis of t being the first time at which 4 selects (4 2 3 0). □

Theorem 2.1 SPVP *captures any oscillation captured by* SPVP*-vs and by* SPVP*-vp. The converse does not hold.*

Proof: We already noted that edge activation sequences are at least as powerful as vertex activation sequences. BLEEDIN-EDGE proves the strictness. In fact, Tab. 2.4 shows an edge activation sequence that triggers a fair oscillation on that instance. Observe that the states at instants $t = 1$ and $t = 9$ coincide and each edge is activated at least once in that time interval (fairness). We now prove that vertex activation sequences always converge on BLEEDIN-EDGE. Lemma 2.2 and Lemma 2.3 ensure that $\forall t\ \pi_t(4) \neq (4\ 2\ 3\ 0)$, regardless of the semantic of vertex activation. Moreover, Lemma 2.1 implies that there exists a time t_0 such that $\forall t > t_0\ \pi_t(4) \neq \epsilon$. Hence, $\forall t > t_0$, $\pi_t(4)$ must be either (4 0) or (4 3 0). Observe that both these paths are extended by vertex 2. Hence, there exists a time instant t_1 such that, for any $t > t_1$: $\pi_t(2) = (2\ 4\ 3\ 0)$ or $\pi_t(2) = (2\ 4\ 0)$. In particular, we have $\pi_t(2) \neq (2\ 1\ 0)$ which, in turn, implies the existence of a $t_2 > t_1$ such that $\pi_t(3) = (3\ 0)$ for any $t > t_2$. As a consequence, there will be a time $t_3 > t_2$ such that $\pi_t(4) = (4\ 3\ 0)$ for any $t > t_3$. This prevents 2 from selecting path (2 4 0) and therefore ultimately stabilizes 1 on $\pi_t(1) = (1\ 0)$ for any $t > t_4 > t_3$. □

We complete our discussion on the ability of SPVP variants to capture oscillations with the following theorem, that puts in evidence the importance of considering rib-in$_t$. Let SPVP-nr be the variant of SPVP that does not equip vertices with a rib-in$_t$.

Theorem 2.2 SPVP *captures any oscillation captured by* SPVP*-nr. The converse does not hold.*

Proof: As we already remarked in Section 2.4, the absence of rib-in$_t$ forces a vertex to query all its neighbors for each computation of a new best path. This corresponds to activating vertices to process, hence the statement can be proved in a way similar to Theorem 2.1. □

Chapter 3

Theoretical Literature on BGP Stability*

3.1 Introduction

Plenty of research efforts have focused on BGP stability in the last decade. In this chapter we survey the state of the art on this topic. Our goal is to present existing literature with a systematic and coherent approach, relating different pieces of work by means of a common framework.

First, we introduce a number of convergence properties that relate to BGP stability, and then classify significant network configurations according to these properties. Second, we discuss the relationships between the classical models for stability and other variants that accommodate link costs or account for the existence of commercial relationships in the routing system. Third, we study the relationship between the succinctness of the representation of routing policies and the computational complexity of the stability problem. Fourth, we discuss the stability of iBGP [RLH06], the intra-AS version of BGP. Finally, we present proposed solutions that modify BGP in such a way to guarantee stability.

Section 3.2 formalizes a set of foundational problems related to BGP sta-

*Part of the material presented in this chapter is based on the following publication

- L. Cittadini, G. Di Battista, M. Rimondini. (Un)-Stable Routing in the Internet: A Survey from the Algorithmic Perspective. In *Proc. International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008)*, Springer, 2008.

bility that will be referenced throughout the thesis. Section 3.3 shows the interplay between stability and real world constraints. In Section 3.4 we study the stability problem on networks where topology changes can happen due to either hardware or software faults, and configuration changes can happen due to policy changes. Section 3.5 discusses how the compactness of the representation of policy configurations can impact the computational complexity of the stability problem. Finally, we review some of the proposed modifications to BGP in Section 3.6.

The vast majority of the results presented in this chapter comes from existing literature and are given proper credit using citations in the statements. Other results, though never previously presented, can be easily inferred from the literature, and are indicated by citing the papers they are derived from. We also present some original contributions which allow us to better relate prior work.

3.2 Stable States and Guaranteed Convergence

Obviously, the existence of a stable path assignment is a crucial requirement for a network running BGP. For example, instance GOOD-GADGET (Fig. 2.1a) admits a stable path assignment. In fact, it is easy to check that the path assignment $\pi(1) = (1\ 3\ 0)$, $\pi(2) = (2\ 0)$, $\pi(3) = (3\ 0)$, $\pi(4) = (4\ 2\ 0)$ is stable. On the other hand, instance BAD-GADGET (Fig. 2.1b) does not admit any stable path assignment. In fact, in any stable path assignment π there must be at least one vertex among 1, 2, and 3 that picks the direct path to 0. Assume that $\pi(2) = (2\ 0)$. For π to be stable, we must have $\pi(3) = (3\ 2\ 0)$, which in turn implies $\pi(1) = (1\ 0)$. Note that now $(2\ 0)$ is not the best available path at vertex 2, hence no path assignment can have $\pi(2) = (2\ 0)$. The same argument applies symmetrically to the other vertices.

Unfortunately, deciding whether a given instance of SPVP admits a stable path assignment is a NP-complete problem [GSW02].

Problem 3.1 (Stable Paths Problem) *Given an instance S of* SPVP*, does S admit a stable path assignment?*

Theorem 3.1 *The Stable Paths Problem is NP-complete [GSW02].*

Since we have seen that there are SPVP instances admitting one or no solutions, one natural question is whether there are SPVP instances that admit more than one solution. Let Unique be the class of SPVP instances that have

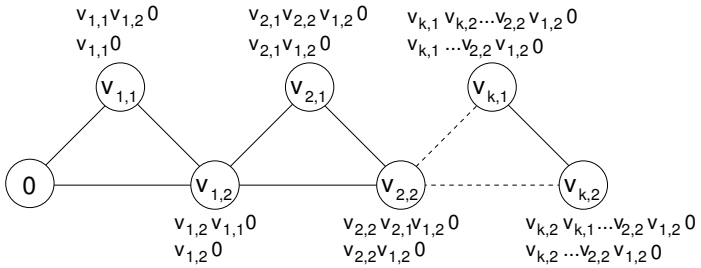


Figure 3.1: An instance of SPVP with $k + 1$ solutions.

exactly one solution. We have that GOOD-GADGET belongs to Unique. In fact, it is easy to see that any stable path assignment π for GOOD-GADGET must be such that $\pi(3) = (3\ 0)$. For π to be stable, this also implies $\pi(1) = (1\ 3\ 0)$, which in turn forces $\pi(2) = (2\ 0)$, and, ultimately, $\pi(4) = (4\ 2\ 0)$.

The following theorem shows that the inclusion between Solvable and Unique is proper.

Theorem 3.2 *Unique* $\subset$ *Solvable [GW99].*

Proof: Instance DISAGREE (Fig. 2.1c) has two solutions. It is easy to see that both $\pi(1) = (1\ 0)$, $\pi(2) = (2\ 1\ 0)$ and $\pi'(1) = (1\ 2\ 0)$, $\pi'(2) = (2\ 0)$ are stable path assignments. □

Another interesting question is whether the number of possible solutions in an SPVP instance is bounded by a polynomial function in the input size. Property 3.1 shows that SPVP instances can be constructed in such a way to admit an exponential number of distinct stable path assignments. Moreover, Property 3.2 proves that, given any integer k, it is possible to build an instance of SPVP admitting exactly k distinct stable states.

Property 3.1 *There are instances of* SPVP *with an exponential number of solutions.*

Proof: An instance that contains n distinct and independent DISAGREE structures has 2^n solutions. □

Property 3.2 *For each non-negative integer k there exists an instance of* SPVP *with k solutions.*

Proof: This is clearly true for $k = 0$ and $k = 1$. In fact, BAD-GADGET and GOOD-GADGET are examples of instances with no solutions and with a unique solution, respectively. Fig. 3.1 shows the structure of a generic SPVP instance with $k + 1$ solutions. The idea is to stack several DISAGREE gadgets, each of which adds a new stable path assignment to the SPVP instance. In particular, each triple of vertices $(v_{i,1}, v_{i,2}, v_{i-1,2})$, with $i > 0$ and $v_{0,2} = 0$, forms a DISAGREE. Each vertex $v_{i,j}$ can reach 0 by using a direct path $(v_{i-1,2}\ v_{i-2,2}\ \ldots\ 0)$. However, $v_{i,1}$ prefers the path via $v_{i,2}$, and $v_{i,2}$ prefers the path via $v_{i,1}$. With this construction, if a DISAGREE $(v_{i,1}, v_{i,2}, v_{i-1,2})$ stabilizes on $\pi_t(v_{i,1}) = (v_{i,1}\ v_{i-1,2}\ v_{i-2,2}\ \ldots\ 0)$ and $\pi_t(v_{i,2}) = (v_{i,2}\ v_{i,1})\pi_t(v_{i,1})$ at time t, then there exists a time $t' > t$ after which the vertices of any other DISAGREE $(v_{j,1}, v_{j,2}, v_{j-1,2})$, $j > i$ (but $v_{i,2}$) permanently select the empty path ϵ. On the other hand, if a DISAGREE stabilizes on $\pi_t(v_{i,1}) = (v_{i,1}\ v_{i,2})\pi_t(v_{i,2})$ and $\pi_t(v_{i,2}) = (v_{i,2}\ v_{i-1,2}\ \ldots\ 0)$, then the next DISAGREE $(v_{i+1,1}, v_{i+1,2}, v_{i,2})$ (if any) has two stable path assignments, and a similar argument can be applied.

In this way, every DISAGREE acts as a switch that can enable or disable the subsequent DISAGREE in the stack. The last DISAGREE $(v_{k,1}, v_{k,2}, v_{k-1,2})$ can arbitrarily reach one of its two stable states without influencing further gadgets. It is easy to check that this SPVP instance has exactly $k + 1$ stable states. □

Even if an instance has a stable state, it can still have oscillations [GSW99]. For example, some message orderings might lead to a stable path assignment, while others might lead to a persistent oscillation. This is especially dangerous from the point of view of a network operator: rather than knowing whether a stable path assignment exists, he is much more interested in knowing whether a given BGP configuration is *guaranteed* to converge to a stable state, regardless of any possible message orderings. Guaranteed convergence can be formalized in terms of fair activation sequences. In particular, one can ask whether all the fair activation sequences of an SPVP instance are finite. If this is the case, the instance is said to be *safe*. More formally, the *Safety* problem [GSW99, GR00] is defined as follows:

Problem 3.2 (Safety) *Given an* SPVP *instance S, does S admit only finite fair activation sequences?*

Let Safe be the class of SPVP instances admitting only finite fair activation sequences. Unfortunately, there exist instances which are not in Safe, despite the existence of a stable state.

Theorem 3.3 *Safe ⊂ Solvable [GW99].*

Proof: DISAGREE (Fig. 2.1c) is Solvable but not Safe. An infinite fair activation sequence $\sigma = (A_0 \dots A_i \dots)$ can be constructed as shown in Table 2.2. □

The following result determines the relationship between Safe and Unique.

Theorem 3.4 *Safe ⊂ Unique [SSZ09, GSW02].*

Proof: Part 1 ($\subseteq$): [SSZ09] prove the statement by analyzing the state-transition graph of an SPVP instance.

Part 2 ($\subset$): instance NAUGHTY-GADGET (Fig. 2.1d) has a unique stable state but a persistent oscillation. The unique solution is the same as in GOOD-GADGET, while the persistent oscillation works in a similar way as in BAD-GADGET (see [GSW02]). □

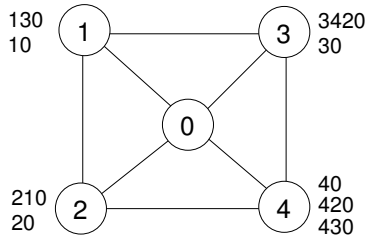
The safety of SPVP has also been studied from a game theoretic perspective and [FSS07] is a good introduction to the application of game theoretical techniques to interdomain routing problems.

[FP08] provides a model for the safety problem in the so called *convergence game*. In this game, all the vertices of the graph but 0 are players. When allowed to play, a vertex selects one of its neighbors in order to route traffic towards 0. The set of all choices defines a directed sub-graph G_s of G where each vertex has out-degree one. The payoff of vertex u is $\lambda^u(P)$ if a path $P = (u \dots 0)$ exists in G_s, or $\lambda^u(\epsilon)$ otherwise. The game has infinite rounds. SPVP safety translates to the convergence of best-reply dynamics in the convergence game. In this framework the following result has been proved:

Theorem 3.5 *The safety problem is PSPACE-complete [FP08].*

However, interpreting Theorem 3.5 from the SPVP point of view requires some care. First, [FP08] only considers activation sequences where vertices are activated one after the other. We have shown in Section 2.5 that such a model does not capture any possible routing oscillation. Hence, the computational complexity of the safety problem in the general SPVP framework is still open.

Most of the sufficient conditions [JR06, FJB07, JR04, GSW02, GSW99, RS06] to ensure safety are based on the so called dispute wheels. A *dispute wheel* [GSW99] $\Pi^k = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ of size k is a sequence of vertices $\vec{\mathcal{U}} = (u_0\ u_1 \dots u_{k-1})$ and sequences of nonempty paths $\vec{\mathcal{Q}} = (Q_0\ Q_1 \dots Q_{k-1})$ and $\vec{\mathcal{R}} = (R_0\ R_1 \dots R_{k-1})$ such that:



(e)

Figure 3.2: BAD-BACKUP: an SPVP instance that has a dispute wheel, yet is safe. Observe that this instance becomes BAD-GADGET (Figure 2.1b) if the link between vertices 4 and 0 is removed.

(i) R_i is a path from u_i to u_{i+1}

(ii) $Q_i \in \mathcal{P}^{u_i}$

(iii) $R_i Q_{i+1} \in \mathcal{P}^{u_i}$

(iv) $\lambda^{u_i}(Q_i) \geq \lambda^{u_i}(R_i Q_{i+1})$

To simplify the notation we shall omit specifying the size of the dispute wheels when it is unknown or immediately inferrable from the context.

We define No-Dispute-Wheel as the set of SPVP instances that do not have a dispute wheel. The following theorem shows that instances without a dispute wheel are safe.

Theorem 3.6 *No-Dispute-Wheel* $\subset$ *Safe [GSW02].*

Proof: Part 1 ($\subseteq$): If an SPVP instance has no dispute wheels, then it is safe [GSW02].

Part 2 ($\subset$): As shown in [GSW99], instance BAD-BACKUP (Fig. 3.2) has a dispute wheel $\Pi^3 = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$, where $\vec{\mathcal{U}} = (1\ 3\ 2)$, $\vec{\mathcal{Q}} = ((1\ 0)\ (3\ 0)\ (2\ 0))$, and $\vec{\mathcal{R}} = ((1\ 3)\ (3\ 4\ 2)\ (2\ 1)))$. However, it is easy to see that it is safe. In fact, any fair activation sequence must be such that path (0) is eventually advertised to vertex 4, which selects it as the best path. Since path (0) comes directly from the origin, it will never be withdrawn, hence vertex 4 will never fall back on alternative paths: in particular, it will never select path (4 2 0). This causes vertex 3 to eventually stabilize on path (3 0) which, in turn, implies that vertex

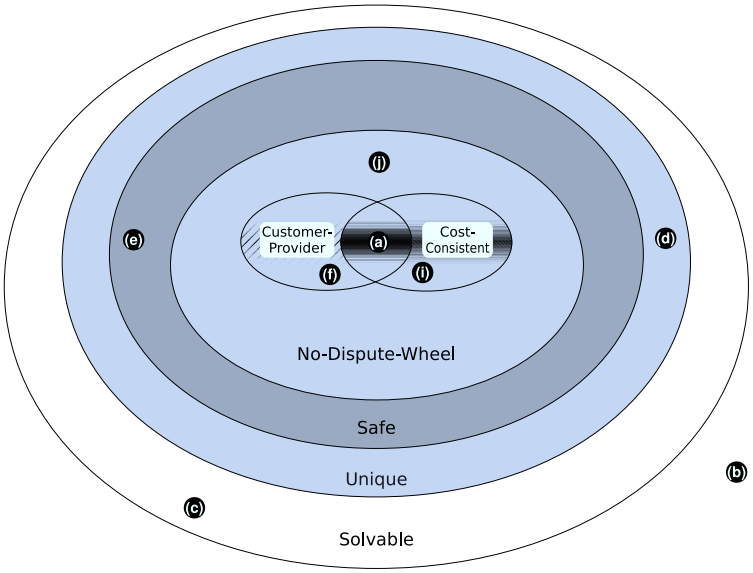


Figure 3.3: Relationships between the classes of SPVP instances. Black dots with letters represent instances from Figs. 2.1, and 3.4.

1 will permanently select its preferred path (1 3 0). As a consequence, vertex 2 will eventually stabilize on path (2 0), yielding a stable path assignment. Since we did not make any assumptions on the nature of the fair activation sequence, we conclude that BAD-BACKUP is safe. □

The absence of dispute wheels is interesting from both a theoretical and a more pragmatic point of view because it is a sufficient condition for safety that does *not* involve evaluation of the dynamics of the protocol. That is, it is a "static" condition. Such a condition allows us to prove stability properties without having to cope with the details of dynamic evaluation.

The relationships between the classes that we introduced so far are illustrated in Fig. 3.3, which can be effectively used as a guideline for reading the whole chapter.

3.3 Link Costs and Commercial Relationships

In this section we relate the stability of SPVP instances to several real-life constraints on routing policies that stem either from operational best practices and/or from economic reasons (see again Fig. 3.3 to follow the section). First, we discuss the effect of policies that rank paths according to a generalized shortest path. Second, we show perhaps the most interesting insight of this section, proved in [GR00]: it is possible to achieve provably safe path vector interdomain routing without the need to enforce constraints globally, if all ASes independently behave according to basic economic rules. This result can be viewed as one of the reasons why routing instabilities are not so frequently observed in the Internet. Although, as put in evidence in [FBR04], practical configuration can significantly deviate from such economic rules for legitimate purposes.

Cost-Consistent Instances

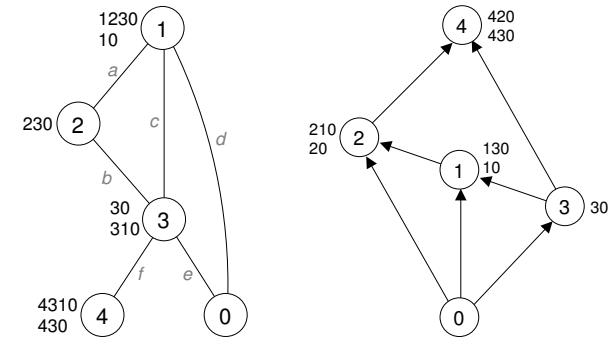
In computer networks quite often links are associated with a cost, e.g., related to bandwidth, latency, traffic, etc. Hence, it is natural to rank the paths according to their cost. Consider an SPVP instance $(G, \mathcal{P}, \Lambda)$ and suppose that the edges of G have a cost. Let the cost function $c : E \to \mathbb{Z}$ be such that no cycle exists with a non-positive cost and suppose that the paths are ranked by functions in Λ according to their cost. We say that $(G, \mathcal{P}, \Lambda)$ is *cost-consistent* with c. We define set Cost-Consistent as the set of SPVP instances that are cost-consistent with at least one cost function [GSW99]. The following theorem relates cost-consistent instances to the concept of dispute wheel.

Theorem 3.7 *Cost-Consistent* $\subset$ *No-Dispute-Wheel [GSW99].*

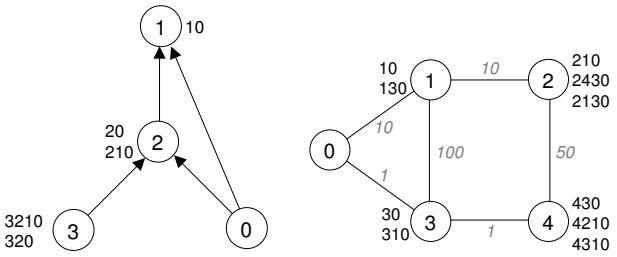
Proof: Part 1 ($\subseteq$): The absence of dispute wheels in cost-consistent instances is proved in [GSW99].

Part 2 ($\subset$): INCOHERENT (Fig. 3.4f), a simplification of an instance presented in [GSW99], is an example of instance that has no dispute wheel and is not cost-consistent with any cost function. Assign variables to edges according to Fig. 3.4f. Since path (3 2 1 0) is preferred over (3 2 0), the cost function must be such that $a + b + d < a + c$. Also, since (2 0) is preferred over (2 1 0) we have $c < b + d$, yielding a contradiction. □

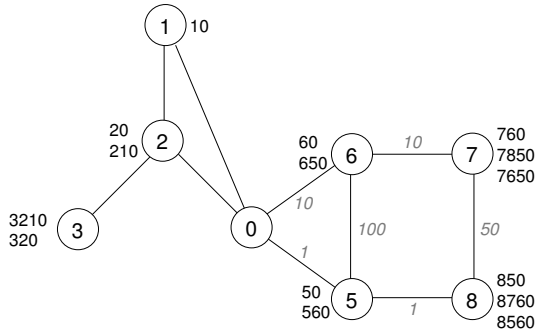
An interesting consequence of Theorem 3.7 is that the class of instances that are provably safe is strictly larger than cost-consistent instances. That



(f) Incoherent. (g) Layered Good-Gadget.



(h) Layered Incoherent. (i) Costomer.



(j) Nocust-Nolowest-omer.

Figure 3.4: SPVP instances and real world constraints.

is, we can achieve stability even without being consistent with a global cost function. This disproves a guess of [VGE00], hypothesizing that only shortest path route selection can be provably safe.

Modeling Commercial Relationships

From the economic perspective, relationships between ASes can be roughly classified as customer-provider or peer-peer. In order to implement these agreements, routing policies must obey several constraints. In [GR00] it has been observed that an SPVP instance $(G, \mathcal{P}, \Lambda)$ satisfying these constraints must be as follows.

The neighbors of each vertex of G can be partitioned into three sets: customers, providers, and peers, such that:

(i) Each path of $\mathcal{P}$ is *valley-free*: provider-customer and peer-peer edges can only be followed by provider-customer edges. A valley is considered an anomaly because it corresponds to an AS providing transit to either its peers or its providers without revenues.

(ii) Functions in Λ are such that, at each vertex in G, paths through customers are ranked better than paths through peers that, in turn, are ranked better than paths through providers (*prefer-customer* ranking). This corresponds to preferring routes with lower cost.

(iii) The customer-provider digraph $\mathcal{G}$ is acyclic. The vertices of $\mathcal{G}$ are the ASes in G, while the edges represent a customer-provider relationship between two ASes, and are directed from the customer AS to the provider AS. Cycles would correspond to unclear customer-provider roles.

Good-Gadget (Fig. 2.1a) and Incoherent (Fig. 3.4f) are examples of instances that admit an assignment of commercial relationships satisfying Conditions *(i)*, *(ii)*, and *(iii)* above. Two such assignments are shown in Figs. 3.4g and 3.4h, where edges go from customers to providers. The layering emphasizes the customer-provider hierarchy.

In [DEH$^+$07] it is shown that Condition *(i)* can be tested efficiently:

Theorem 3.8 *Given an* SPVP *instance, it takes polynomial time to test whether the neighbors of each vertex of G can be partitioned into three sets: customers, providers, and peers, such that each permitted path is valley-free [DEH$^+$07].*

Other work on the computational complexity of checking Conditions *(i)* and *(iii)* is presented in [KMT06]. An interesting open problem in this respect is whether it is possible to check in polynomial time if an SPVP instance satisfies Conditions *(i)*, *(ii)*, and *(iii)*.

The following two Theorems show that there exist cost-consistent instances that do not belong to Customer-Provider, and vice versa.

Theorem 3.9 *Customer-Provider* $\cap$ *Cost-Consistent* $\neq$ *Cost-Consistent.*

Proof: We have already shown in the proof of Theorem 3.7 that INCOHERENT (Fig. 3.4f) is not cost-consistent with any cost function. On the other hand, the policies in INCOHERENT are compatible with the customer-provider hierarchy depicted in Fig. 3.4h. □

Theorem 3.10 *Customer-Provider* $\cap$ *Cost-Consistent* $\neq$ *Customer-Provider.*

Proof: It is easy to check that COSTOMER (Fig. 3.4i) is cost-consistent with the edge cost function which is presented in the figure. We now show that COSTOMER is not compatible with any customer-provider hierarchy. Consider edge $(1, 2)$. We have three possibilities:

(i) 1 is a provider of 2. Since $\lambda^2((2\ 1\ 3\ 0)) < \lambda^2((2\ 4\ 3\ 0)) < \lambda^2((2\ 1\ 0))$, 4 must be a provider of 2 as well. Then path (4 2 1 0) contains a valley.

(ii) 1 is a peer of 2. As above, 4 is also a peer of 2, and path (4 2 1 0) contains two consecutive peer-peer edges.

(iii) 1 is a customer of 2. Applying the same argument as above, we conclude that 2 is a provider of 4. By looking at the path rankings at 4, we have that $\lambda^4((4\ 2\ 1\ 0)) < \lambda^4((4\ 3\ 1\ 0))$ implies that 3 is also a provider of 4. Then path (2 4 3 0) forms a valley.

□

The following two Theorems complete the taxonomy of the Cost-Consistent and Customer-Provider classes, with respect to the other classes of instances presented so far.

Theorem 3.11 *Customer-Provider* $\subset$ *No-Dispute-Wheel (derived from [GGR01]).*

Proof: Part 1 ($\subseteq$): Customer-provider instances cannot have a dispute wheel [GGR01].

Part 2 ($\subset$): COSTOMER (Fig. 3.4i) has no dispute wheel, since it is cost-consistent with an edge cost function. On the other hand, from the proof of Theorem 3.10 we have that COSTOMER $\notin$ Customer-Provider. □

Theorem 3.12 *Customer-Provider* $\cup$ *Cost-Consistent* $\subset$ *No-Dispute-Wheel.*

Proof: It is possible to merge COSTOMER and INCOHERENT together, obtaining NOCUST-NOLOWEST-OMER (Fig. 3.4j). Since the building blocks are completely independent and do not form a dispute wheel, then NOCUST-NOLOWEST-OMER $\in$ No-Dispute-Wheel. On the other hand, the proofs of Theorems 3.7 and 3.11 ensure that the instance is neither in Cost-Consistent nor in Customer-Provider. □

3.4 Guaranteed Convergence under Faulty Conditions

Observe that the problems we defined so far assume that both the network topology and the policies at each vertex are fixed. In a real network, topology changes can happen due to either hardware or software faults, and configuration changes can happen due to policy changes. Hence, it makes sense to study the extent to which stability properties are safe even under topology or policy changes.

Robust Instances

The most common type of topology change in a computer network is the failure of a vertex or link. From a network operator's perspective, it is interesting to determine whether a given policy configuration is safe even under arbitrary combinations of link or vertex failures (the latter being a special case of the former). More formally, we define the robustness problem as follows:

Problem 3.3 (Robustness) *Given an* SPVP *instance* $S = ((V, E), \mathcal{P}, \Lambda)$*, is instance* $S' = ((V, E'), \mathcal{P}, \Lambda)$ *safe for any* $E' \subseteq E$*?*

We define set Robust as the set of robust SPVP instances.

Property 3.3 *Robust* $\subset$ *Safe [GSW02].*

Proof: Part 1 ($\subseteq$): By definition, safety is a necessary condition for robustness.

Part 2 ($\subset$): BAD-BACKUP, Fig. 3.2, proves the strictness. From the proof of Theorem 3.6, we know that BAD-BACKUP is safe. However, by removing

edge $(4, 0)$ the instance becomes similar to BAD-GADGET (Fig. 2.1b). Since there exists a combination of link and vertex failures which leads to an instance that is not safe, BAD-BACKUP is not robust. □

Interestingly, the absence of a dispute wheel, which is a sufficient condition for safety, is also a sufficient condition for robustness. In a sense, this shows how the absence of dispute wheels is far from being necessary for safety.

Theorem 3.13 *No-Dispute-Wheel ⊂ Robust (derived from [GSW02]).*

Proof: Part 1 ($\subseteq$) [GSW02]: If an instance S on graph (V, E) has no dispute wheel, then, by Theorem 3.6, the instance is safe. As removing vertices or links cannot result in creating a dispute wheel, any instance S' on graph (V, E') such that $E' \subseteq E$ is also safe.

Part 2 ($\subset$): Consider instance RO-DW-UST in Fig. 3.5a. It is easy to see that RO-DW-UST contains only one dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ where $\vec{\mathcal{U}} = (1\ 2)$, $\vec{\mathcal{Q}} = ((1\ 3\ 0)\ (2\ 3\ 4\ 0))$, and $\vec{\mathcal{R}} = ((1\ 2)\ (2\ 1))$. We now prove that the instance is robust. Observe that RO-DW-UST is safe. In fact, vertex 4 will eventually learn about its direct route to 0, causing vertex 3 to permanently select its preferred route (3 4 0). In turn, 1 and 2 will eventually have no other options but to select paths (1 2 3 4 0) and (2 3 4 0) respectively, thus leading to a stable path assignment. Moreover, it is easy to check that removing links or vertices from RO-DW-UST destroys Π. Hence, by Theorem 3.6, S' must be safe. □

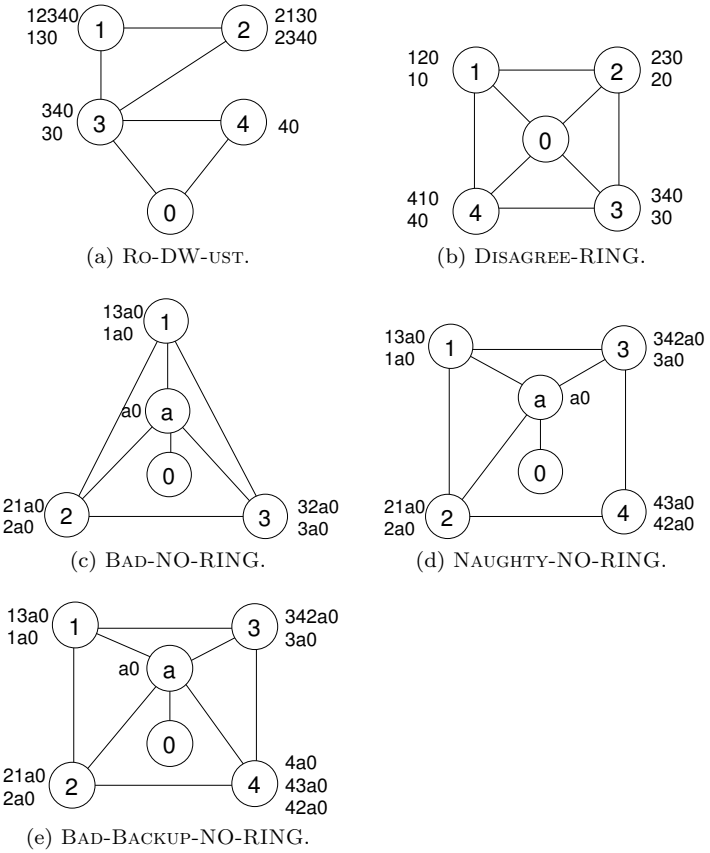
Safe under Filtering Instances

Another network event that prevents some ASes from receiving the announcement of paths is the insertion of route *filters*. Hence, it is interesting to study whether the safety of a given routing configuration is guaranteed to survive even if arbitrary filtering is applied by the ASes. More formally, the safety under filtering problem is defined as follows:

Problem 3.4 (Safety Under Filtering) *Given an* SPVP *instance* $S = (G, \mathcal{P}, \Lambda)$, *is instance* $S' = (G, \mathcal{P}', \Lambda)$ *safe for any* $\mathcal{P}' \subseteq \mathcal{P}$*?*

We define SUF as the set of SPVP instances which are safe under filtering.

Property 3.4 *SUF ⊆ Robust [FJB07].*



(a) Ro-DW-ust. (b) Disagree-RING.

(c) Bad-NO-RING. (d) Naughty-NO-RING.

(e) Bad-Backup-NO-RING.

Figure 3.5: Instances that prove strict inclusions for the No-Dispute-Ring, No-Dispute-Wheel, Robust, and SUF classes.

Proof: The failure of edge $e = (u, v)$ makes all the paths in $\mathcal{P}$ which contain e unavailable. On the other hand, the failure of vertex v is equivalent to the removal of all edges (v, u) that are incident on v. Therefore, if an SPVP instance is safe under removal of any path (i.e., safe under filtering), it is also robust. □

An argument similar to the one in Theorem 3.13 can be used to show the following.

Theorem 3.14 *No-Dispute-Wheel ⊂ SUF (derived from [FJB07]).*

Proof: Part 1 (⊆): Removing paths cannot create a dispute wheel in an instance that does not contain dispute wheels. Therefore, filtering paths from any instance without dispute wheels forcedly results in a safe instance.

Part 2 (⊂): Instance Ro-DW-ust (Fig. 3.5a) is safe but has the dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ where $\vec{\mathcal{U}} = (1\ 2)$, $\vec{\mathcal{Q}} = ((1\ 3\ 0)\ (2\ 3\ 4\ 0))$, and $\vec{\mathcal{R}} = ((1\ 2)\ (2\ 1))$. Now, the removal of any path from Ro-DW-ust breaks at least one path of Π, yielding a safe instance. □

BAD-BACKUP is also an example of a safe instance which is not in SUF.

So far, we have focused on sufficient conditions (e.g., the no dispute wheel condition guarantees robustness). An interesting result from [FJB05, FJB07] is a necessary condition for safety under filtering, which is based on a specialized class of dispute wheels which the authors call dispute rings. A *dispute ring* is a dispute wheel $\Pi^k = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ such that $k \geq 3$ and $\forall v \neq 0$ and $v \notin \vec{\mathcal{U}}$, v appears in at most one path Q_i or R_i. Essentially, a dispute ring can be drawn as a "wheel", where paths in $\vec{\mathcal{Q}}$ form the spokes of the wheel and share only vertex 0. Paths in $\vec{\mathcal{R}}$, on the other hand, form the rim of the wheel, and each path R_i has only vertex u_i in common with Q_i. We define set No-Dispute-Ring as the set of instances which do not have any dispute ring.

Theorem 3.15 *SUF ⊂ No-Dispute-Ring [FJB07].*

Proof: Part 1 (⊆): Consider an instance S which has a dispute ring $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$, and remove all paths P such that $\nexists i, A, B \mid Q_i = APB \vee R_i = APB$, that is, keep only those paths which build up the dispute ring. As explained in [FJB07], the resulting instance can oscillate by first propagating messages along the spoke paths and then over the rim paths so that vertices in $\vec{\mathcal{U}}$ are updated in reverse order.

Part 2 (⊂): It can be easily checked that instance DISAGREE in Fig. 2.1c does not have any dispute ring and, as pointed out in the proof of Theorem 3.3, is not safe, hence is not in SUF. □

We now show how No-Dispute-Ring and Robust relate to the classes we have introduced so far.

Relating No-Dispute-Ring to other Classes

As shown in Theorem 3.15, the absence of dispute rings is a necessary condition for safety under filtering. Investigating the relationships between No-Dispute-Ring and the other classes is useful for perceiving the distance between this necessary condition and safety under filtering.

We observe that, starting from an instance S that has a dispute ring and belongs to Solvable−Unique (see Fig. 3.3), we can easily create a new instance S' without dispute rings and still belonging to Solvable−Unique. We now show how to build $S' = (G' = (V', E'), \mathcal{P}', \Lambda')$ starting from $S = (G = (V, E), \mathcal{P}, \Lambda)$. Let $V' = V \cup \{a\}$, with $\mathcal{P}'^a = \{(a\ 0), \epsilon\}$, $E' = E - \{(u, 0) \in E\} \cup \{(a, 0)\} \cup \{(u, a) \mid (u, 0) \in E\}$ and, for each path $P \in \mathcal{P}$, replace (0) with $(a\ 0)$. That is, we replace vertex 0 with a vertex a that has a single path to 0 and all the paths are updated to pass through a before reaching 0. It is easy to see that any dispute ring in S disappears in S' since vertex a is repeated in all paths. In this way, we can easily prove the following:

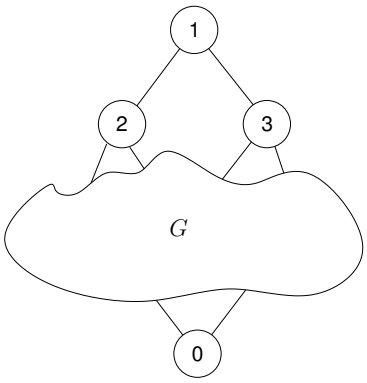
— Solvable ∩ No-Dispute-Ring ≠ No-Dispute-Ring (see instance BAD-NO-RING in Fig. 3.5c). That is, there exist unsolvable instances yet without any dispute rings.

— No-Dispute-Ring ∩ Unique ⊈ Safe (see instance NAUGHTY-NO-RING in Fig. 3.5d). That is, there exist instances that have a unique solution and no dispute rings, but are not safe.

On the other hand, we observe that instances DISAGREE and DI-SAFE-GREE do not have dispute rings just because they violate the condition of having at least 3 vertices participating in the dispute. We can easily alter them to have a dispute ring, thus proving that there exists at least one instance that is solvable, not unique (hence not safe), and has a dispute ring (see instance DISAGREE-RING in Fig. 3.5b). Solvable ∩ No-Dispute-Ring − (Safe ∪ Unique) ≠ Solvable

Relating Robust to other Classes

A fundamental problem which remains still open is to capture the difference between SUF and Robust, the two classes that have been introduced to study guaranteed convergence under faulty conditions. We know from Property 3.4




```
aut-num: AS1
import: from AS2 action pref=50;
        accept ANY
export: to AS2 announce ANY
import: from AS3 action pref=100;
        accept ANY
export: to AS3 announce ANY
```

Figure 3.6: Example of compact description of routing policies that affect an exponential number of paths.

that robustness can be viewed as a special case of safety under filtering. In Chapter 4 we shed some light on the relationship between these two classes.

3.5 Compact Routing Policies and Convergence

As we have seen throughout this chapter, routing policies are modeled in an SPVP instance $(G, \mathcal{P}, \Lambda)$ by sets $\mathcal{P}$ and Λ. Namely, $\mathcal{P}^v$ represents all the routes accepted by vertex v, while λ^v captures the relative preference assigned to the accepted routes. [FJB07] stressed the different role of these two components, distinguishing between the *filtering* and *ranking* component, respectively.

We built all the proofs and examples showed up to this point using filtering and ranking components of constant size with respect to the number of vertices in the graph. This allowed us to explicitly represent filters and rankings as ordered lists of paths. However, an SPVP instance can have an exponential number of paths with respect to the number of vertices in the graph, which makes an explicit representation of the routing policies unfeasible. We are therefore interested in considering instances that admit a compact description of both the filtering and the ranking components. Indeed, router configuration languages are designed to support a compact specification of routing policies,

so that network operators can make use of concise commands to perform actions on a number of entities (e.g., paths, network destinations, etc.) that is exponential in the network size. Consider, for example, the SPVP instance in Fig. 3.6. In this instance, vertices 2 and 3 are connected to a graph G whose density may be, in general, very high. The two vertices may therefore learn an exponential number of paths to 0. Assume vertex 1 wants to rank the paths received through 2 better than those received through 3. With an explicit representation of the routing policies, this would require enumerating all the possible paths through 2 and 3 and sorting them so that paths through 2 come before paths through 3. Instead of this verbose and computationally intractable representation, the more succinct policies in Fig. 3.6 can be adopted. The polices are described using the Routing Policy Specification Language [AVG+99], a vendor-independent formalism to describe router configurations. The policies apply different preference values (`pref=`) to the announcements on a per-neighbor basis. That is, all the paths that 1 receives from neighbor `AS2` are assigned preference `50` (higher), while all the paths that 1 receives from neighbor `AS3` are assigned preference `100` (lower). Most routing configuration languages adopt a syntax that is functionally similar to the one of [AVG+99].

In order to study succinct routing policies in an analytical manner, we now formally define compactly representable routing policies. Let $\chi_{\mathcal{P}}$ be the characteristic function of set $\mathcal{P}$, defined as

$$\chi_{\mathcal{P}}(P) = \begin{cases} 1 & \text{if } P \in \mathcal{P}, \\ 0 & \text{otherwise.} \end{cases}$$

Given an SPVP instance $S = ((V, E), \mathcal{P}, \Lambda)$, we say that the filtering (ranking) component *has a compact representation* if $\chi_{\mathcal{P}}$ (respectively, λ^v) is represented in polynomial space in $|V|$. By extension, we say that S itself is *compact* if both components have a compact representation. This section deals with policies that, although compactly representable, allow a number of paths in $\mathcal{P}$ that can be exponential in the size of the network.

Instances without Routing Filters

We say that an SPVP instance $(G = (V, E), \mathcal{P}, \Lambda)$ is *unfiltered* if, for all $v \in V - \{0\}$, $\mathcal{P}^v$ consists of all simple paths on G connecting v to 0. In the following we omit specifying $\mathcal{P}$ for unfiltered instances. Observe that the filtering component of an unfiltered instance has a (trivial) compact representation. Therefore, for

an unfiltered instance to be compact, even the ranking component must have a compact representation.

Fig. 3.6 shows a compact representation of the routing policies of a fragment of an unfiltered instance. In the figure, route filters at vertex 1 are such as to accept (`accept ANY`) and propagate (`announce ANY`) every path from/to the neighbors 2 and 3. The following theorem shows that finding a stable path assignment on unfiltered instances is not easier than solving Problem 3.1 on a generic SPVP instance.

Theorem 3.16 *The Stable Paths Problem (Problem 3.1) is NP-complete even for unfiltered instances having arbitrary rankings.*

We prove the statement by modifying the gadget presented in [GSW02] in order to account for routing policies which lack the filtering component. In the modified gadget, shown in Fig. 3.7, all the paths on the graph are permitted and a ranking is imposed only on the paths that explicitly appear in the figure, while sets irr_u (standing for "irrelevant") abbreviate all the other paths at vertex u. Therefore, the instance has exponential size with respect to the number of vertices. However, since the gadget is unfiltered and the paths in irr_u can be ranked arbitrarily after the constant number of paths that appear explicitly in the figure, the resulting instance has a compact representation and can be constructed in polynomial time with respect to the size of the original 3-SAT instance.

In order to prove the theorem, we first need the results of the following lemmas.

Lemma 3.1 *Every stable path assignment π for the instance in Fig. 3.7 is such that $\pi(u) \notin irr_u$ for every vertex u.*

Proof: Assume by contradiction that there exists a stable state π, reached at time $\bar{t}$, such that $\pi(u) \in \text{irr}_u$ for some vertex u. We have the following cases:

(i) u is either one of the X_i vertices or B_0. However, since $\pi(0) = (0)$ by definition, there exists a time $t' > \bar{t}$ such that $(u\ 0) \in \text{rib-in}_t(u)$ for any $t > t'$. This, together with the fact that $\lambda^u((u\ 0)) < \lambda^u(P)$ for all $P \in \text{irr}_u$, contradicts the assumption that π is stable.

(ii) u is one of the vertices B_i, $i = 1, 2, 3$. By the previous argument we have that $\pi(B_0) = (B_0\ 0)$ which, in turn, implies the existence of a time $t'' > \bar{t}$ such that $(B_i\ B_0\ 0) \in \text{rib-in}_t(B_i)$ for any $t > t''$. The fact

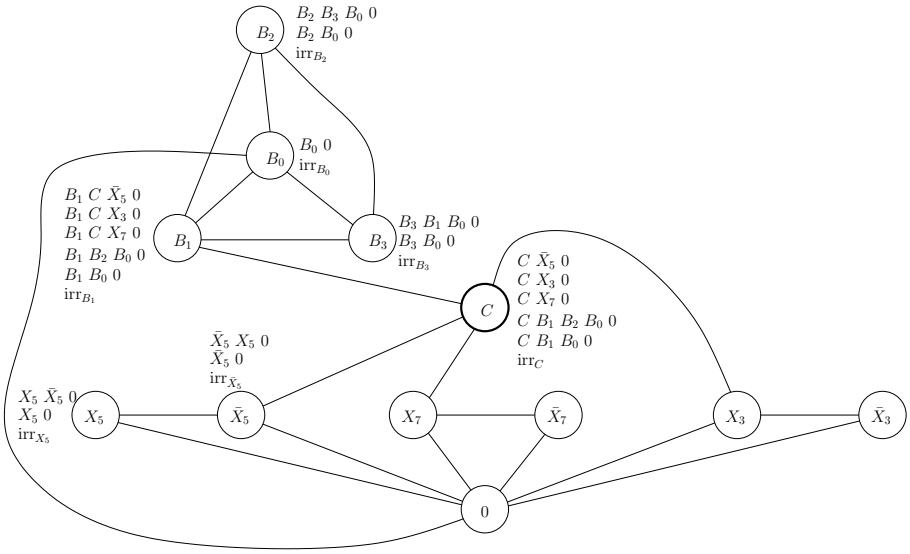


Figure 3.7: An SPVP gadget used to reduce 3-SAT to an unfiltered SPVP instance. Vertex C (with thicker stroke) models a single 3-SAT clause $C = x_3 \vee \bar{x}_5 \vee x_7$. Vertices X_i and $\bar{X}_i$ compose a DISAGREE gadget for each literal appearing in the clause. A BAD-GADGET is built using nodes B_i, $i = 0, 1, 2, 3$. For each vertex u a ranking is imposed only on explicitly represented paths, while set irr_u abbreviates all the other paths. Path rankings at $X_3, \bar{X}_3, X_7, \bar{X}_7$ are omitted for brevity.

that $\lambda^{B_i}((B_i\ B_0\ 0)) < \lambda^{B_i}(P)$ for all $P \in \text{irr}_{B_i}$ again contradicts the assumption that π is stable.

(iii) $u = C$. If this is the case, we show that either $\pi(B_1) = (B_1\ B_2\ B_0\ 0)$ or $\pi(B_1) = (B_1\ B_0\ 0)$. In fact, by the previous argument we know that $\pi(B_1) \notin \text{irr}_{B_1}$. Moreover, the only paths passing through C that B_1 can select in state π are those in irr_{B_1}: doing otherwise would imply $\pi(C) \notin \text{irr}_C$. Hence, there exists a time instant $t''' > \bar{t}$ such that $P \in \text{rib-in}_t(C)$ for any $t > t'''$, where $P \in \{(C\ B_1\ B_2\ B_0\ 0), (C\ B_1\ B_0\ 0)\}$. Since $\lambda^C(P) < \lambda^C(Q)$ for all $Q \in \text{irr}_C$, π cannot be a stable path assignment, once more yielding a contradiction.

□

Lemma 3.2 *The instance in Fig. 3.7 has no stable path assignments π such that $\pi(C) = (C\ B_1\ B_2\ B_0\ 0)$ or $\pi(C) = (C\ B_1\ B_0\ 0)$.*

Proof: Assume by contradiction that a stable state π exists such that $\pi(C) = (C\ B_1)P$, $P \in \{(B_1\ B_2\ B_0\ 0), (B_1\ B_0\ 0)\}$, which implies that $\pi(B_1) = P$. Due to the BAD-GADGET built up by vertices B_0, B_1, B_2, B_3, state π cannot be stable for at least one of the B_i vertices. □

We are now able to prove Theorem 3.16.

PROOF OF THEOREM 3.16. Because of Lemmas 3.1 and 3.2, the gadget in Fig. 3.7 admits a stable state if and only if the corresponding 3-SAT clause is satisfied. The gadget is indeed equivalent to the one used in [GSW02], in the sense that it admits the very same solutions, although it does not require route filtering. It is therefore possible to use the same construction as in [GSW02] to prove the NP-completeness of the Stable Paths Problem for unfiltered instances.

Instances with Next-hop Based Rankings

The great majority of ASes usually configure their policies based solely on the first hop in the route. Therefore, a straightforward example of a ranking function which has a practical interest is next-hop based ranking. More formally, an instance $S = ((V, E), \mathcal{P}, \Lambda)$ of SPVP is said to have *next-hop based rankings* if, for all distinct vertices u, v, and w, we have

$$\lambda^u((u\ v)P_v) < \lambda^u((u\ w)P_w) \Rightarrow \lambda^u((u\ v)P'_v) < \lambda^u((u\ w)P'_w)$$

for all $P_v, P'_v \in \mathcal{P}^v$ and $P_w, P'_w \in \mathcal{P}^w$. That is, the ranking function λ^u is based on the next hop of each path in $\mathcal{P}^u$. For example, it is easy to check that vertices 7 and 8 in COSTOMER (Fig. 3.4i) do not. In fact, considering vertex 7, the ranking is such that $\lambda^7((7\ 6\ 0)) < \lambda^7((7\ 8\ 5\ 0)) < \lambda^7((7\ 6\ 5\ 0))$, and a similar argument applies to vertex 8.

Observe that algorithm SPVP ensures that two paths from the same neighbor of vertex v are never compared by v. Hence, in this case, ranking the paths that traverse the same neighbor does not make sense, and they can all be considered as having the same value of λ^v. As a consequence, next-hop based rankings always admit a compact representation consisting of a total order of the neighbors of each vertex. Fig. 3.6 shows an example of compact representation of next-hop based rankings. Vertex 1 prefers all the paths through 2 over

all the paths through 3, and this is specified by assigning a higher preference value to all the announcements received from 2 without distinction. The statement `from AS2` indicates that the preference applies to all the paths announced by vertex 2, while the statement `accept ANY` specifies that these paths are all assigned the same preference. A similar argument applies to vertex 3.

Similarly, a *filtering* component is *next-hop based* if

$$(u\ v)P_v \in \mathcal{P}^u \Rightarrow \forall P'_v \in \mathcal{P}^v : (u\ v)P'_v \in \mathcal{P}^u$$

That is, the filtering component only considers the next hop in each path.

As it happens for next-hop based rankings, also next-hop based filters always admit a compact representation, simply consisting of the list of (un)filtered neighbors for each vertex.

Theorem 3.17 *Every unfiltered instance $S = ((V, E), \Lambda)$ with next-hop based rankings admits a stable state π. Moreover, π can be computed in polynomial time (derived from [FSS06]).*

Proof: Let $G' = (V, E' = \{(u, v), (v, u) \mid (u, v) \in E\})$ be a directed graph and let δ be a weighing function $\delta : E' \to \mathbb{N}$ such that $\delta(u, v) = k$ if v is the k^{th} vertex when sorting u's neighbors by decreasing values of λ^u. We consider a maximum-weight directed spanning tree (MDST) on G' rooted at vertex 0. Such a tree has exactly one path from 0 to every other vertex $v \in V$, and is therefore also called a maximum-weight spanning 0-*arborescence*. We first prove that an MDST on G' corresponds to a stable state of S. Let $T \subseteq E'$ be a MDST on G' and let π be the path assignment induced by T, that is, $\forall (u, v) \in E' : \pi(v) = (v\ u)\pi(u)$ and $\pi(0) = (0)$. Suppose by contradiction that π, reached at some time t, is not stable. Hence, there exists at least a vertex v for which $\pi(v) \neq \text{best}_t(v)$. Let $u_1, \ldots, u_m$ and w be the successors and the predecessor of v in T, respectively, i.e., $(w, v) \in T$ and $(v, u_i) \in T$, $i = 1, \ldots, m$. Observe that path $\text{best}_t(v)$ cannot have u_i as next hop, since $\pi(u_i) = (u_i\ v\ w)\pi(w)$, or can have w as next hop, since $\pi(v) \neq \text{best}_t(v)$. Therefore, we conclude that $\exists x \in V : (v\ x)\pi(x) = \text{best}_t(v)$, where $x \notin \{u_1, \ldots, u_m, w\}$ and $\delta(x, v) > \delta(w, v)$. Hence $T' = (T \cup \{(x, v)\}) - \{(w, v)\}$ is a spanning tree that has a strictly higher weight than T, contradicting the hypothesis of T being a MDST. The statement follows by noting that an MDST always exists and is computable in polynomial time [FSS06]. □

Observe that the practical relevance of this result is impaired by the fact that, while next-hop based rankings are very common, routing configurations that do not make use of filters are really unlikely.

We now show that, despite the fact that a stable state is guaranteed to exist when next-hop based rankings are in place, the same does not hold for next-hop based filters.

Theorem 3.18 *Consider an* SPVP *instance S having next-hop based filters and arbitrary rankings. S is not guaranteed to have a stable state. Moreover, deciding whether S admits a stable state is NP-hard (derived from [GSW02]).*

Proof: Consider again the gadget in Fig. 3.7 unmodified. In this gadget it is trivially true that every vertex that accepts a path from a neighbor v also accepts any other path from v. That is, the filtering components in the gadget in Fig. 3.7 are next-hop based. Therefore, provided that the explicitly described rankings are adopted, the same argument used in the proof of Theorem 3.16 applies. □

Theorem 3.19 *Consider an* SPVP *instance S having next-hop based rankings and arbitrary filters. S is not guaranteed to have a stable state. Moreover, deciding whether S admits a stable state is NP-hard (derived from [GSW02]).*

Proof: The same arguments as in the proof of Theorem 3.18 can be applied. □

Table 3.1 summarizes the computational complexity of the Stable Paths Problem under different assumptions on the ranking and filtering components. Observe that determining the complexity for the case in which both the ranking and the filtering components are next-hop based is still an open problem.

A natural generalization of next-hop based preferences are *class based* routing policies [JR04]. The ranking (filtering) component of a routing policy is class based if function λ^v (respectively, $\chi^v_{\mathcal{P}}$) at vertex v ranks (filters) paths according to a total order defined on a partition of v's neighbors into classes. That is, vertex v prefers paths announced by neighbors in class C_j to paths announced by neighbors in class C_k, $k > j$, and does not accept paths from neighbors in specific classes. The set of available classes is a network-wide constant. Observe that customer-provider instances are a special case of class based instances where each vertex partitions its neighbors into 3 classes: customers, peers, and providers. In [JR04] a polynomial time algorithm is shown that decides whether an SPVP instance having a dispute wheel can be built given an input specification of class based routing policies.

		Rankings	
		Arbitrary	**Next-hop based**
Filters	**Arbitrary**	NP-complete (Theorem 3.1)	NP-hard (Theorem 3.19)
	Unfiltered	NP-complete (Theorem 3.16)	constant (Theorem 3.17)
	Next-hop based	NP-hard (Theorem 3.18)	?

Table 3.1: Computational complexity of the Stable Paths Problem for different classes of routing policies. The question mark represents an open problem.

3.6 Solving or Detecting Routing Oscillations

Immediately after the existence of intrinsically unstable BGP configurations had been proved, lots of research efforts started focusing on modifications to the BGP protocol that guarantee the absence of routing oscillations. Unfortunately, the widespread use of BGP, together with the large basis of legacy equipment, make deploying changes to the protocol an extremely hard task. Hence, none of the proposed modifications to BGP has ever seen substantial deployment out of network research labs. In this section, we briefly present several solutions that have been proposed in the literature, and highlight the trade-offs between advantages and costs.

SPVP-3

We have shown in Section 2.3 that SPVP is an abstract representation of BGP. In [GW00], extensions to the basic SPVP protocol are proposed in order to detect (SPVP-2) and automatically solve (SPVP-3) possible policy-based protocol oscillations. The idea behind SPVP-2 is the following: every time a vertex v is about to advertise a path P, it adds to the announcement some information regarding the sequence of events that led v to select P. The SPVP protocol is therefore extended with a new piece of information in each announcement. More formally, we define a *path change event* to be a pair $e = (s, P)$ where $s \in \{+, -\}$ is the *sign* of e, and P is a path. The path change event $e = (+, P)$, where $P = (u \dots 0)$, indicates that vertex u has switched to path P because it received an announcement advertising path P, and u ranks P better than its previous best path. On the other hand, $e = (-, P)$ indicates that u was forced to select P because the previously selected best path was made unavailable. A *path history* h is either the empty history $\Diamond$, or a sequence $h = (e_k\ e_{k-1} \dots e_1)$ of path change events e_i, where event e_i is subsequent to event e_{i-1}. A history

h is said to contain a cycle if there exist two distinct events e_i and e_j that contain the same path, i.e., $e_i = (s_i, P)$ and $e_j = (s_j, P)$.

SPVP-2 is simply an extension of SPVP which is able to dynamically compute and store histories in a distributed manner. Each message of the protocol consists of a pair (P, h) where P is a path and h is an history. Upon receiving a message (P, h) from neighbor v, vertex u performs the same actions as in SPVP (see Section 2.3). Additionally, if the currently selected path is no longer the best path, then u builds a new history by prepending a new path change event $e = (s, P)$ to h. Then, u propagates a message (Q, h') to its neighbors, where Q is the new best path, and h' is the concatenation of event e and the old history h.

Intuitively, each time there is a policy-based oscillation in SPVP-2, there will be a cycle in the history h. We can formalize this intuition by the following theorems. The first one shows that SPVP-2 is *correct*, while the second proves the *completeness*.

Theorem 3.20 *The detection of a cycle in a path history means that a policy-based oscillation has been dynamically realized. Furthermore, the paths involved in the cycle describe the policy conflicts that generate the oscillation [GW00].*

Theorem 3.21 *If a policy-based protocol oscillation persists, then all oscillating nodes will eventually detect cycles in path histories [GW00].*

So far, it is clear that SPVP-2 is only able to detect the persistent oscillations – it does not provide mechanisms to solve them. However, it is pretty straightforward to devise a further extension of the protocol that suppresses the paths involved in a cycle in some path history. More precisely, SPVP-3 is an extension of SPVP-2 with a simple additional step: after computing the new best path P and the new path history h, vertex u checks whether h contains a cycle. If not, the execution proceeds as in SPVP-2. Otherwise, path P is removed from $\mathcal{P}^u$, and a new computation of the best path is triggered. This modification forces the first vertex which detects a cycle to permanently filter the path causing the cycle, hence fixing the resulting oscillation. Theorem 3.22 proves that SPVP-3 is free from oscillations. However, we argue that, since SPVP-3 is allowed to modify $\mathcal{P}$, this does not correspond to solving Problem 3.1 on the original instance $S = (G, \mathcal{P}, \Lambda)$ of SPVP: instead, the filtering step of SPVP-3 builds a new instance $S' = (G, \mathcal{P}', \Lambda)$ on which it computes a stable path assignment.

Theorem 3.22 *Given an* SPVP *instance* $S = (G, \mathcal{P}, \Lambda)$*,* SPVP*-3 builds a new instance* $S' = (G, \mathcal{P}', \Lambda)$ *with* $\mathcal{P}' \subseteq \mathcal{P}$ *such that* S' *always converges to a stable path assignment [GW00].*

Global Precedence

The major drawback of SPVP-3 is that routing histories tend to be large in size, thus causing lots of overhead in protocol communication. [ERC+07] try to tackle this drawback by proposing an extension of SPVP which only carries additional information of limited size, that can be represented by a natural number). This additional attribute is referred to as the *global precedence* metric. The distributed algorithm is modified so that the best path is computed considering only those announcements that have a minimal precedence value. More precisely, a message $m = (P, \alpha)$ consists of a path P and a precedence α. Define $\text{pr}_t(u, \alpha)$ as the set of announcements in $\text{rib-in}_t(u \Leftarrow w)$, for all the neighbors w of u, that have exactly α as the precedence metric. Let α_1 be the smallest precedence value such that $\text{pr}_t(u, \alpha_1)$ is not empty. Finally, let $\text{usable}_t(u) = \text{pr}_t(u, \alpha_1)$. The selection step is then redefined such that $\text{best}_t(u) = \arg\min \lambda^u(\text{usable}_t(u))$. Upon receiving a message (P, α), vertex u performs the usual steps of SPVP, though with the modified selection step introduced above. After selecting the new best path Q, u computes the *local precedence* of Q, $\text{lp}^u(Q)$, as follows: first, u builds the set of received announcements $\mathcal{A} = \bigcup_w \text{rib-in}_t(u \Leftarrow w)$, for all w neighbors of u; then, $\text{lp}^u(Q)$ is given by the position of Q in $\mathcal{A}$, sorted by increasing values of λ^u. Finally, u propagates the message $(Q, \alpha + \text{lp}^u(Q))$ to every neighbor. Intuitively, u is using the precedence metric to inform its neighbors about the ranking u assigns to the announced path Q.

It is interesting to observe that this mechanism forces a route that is most preferred at all ASes along its path to be tagged with global precedence 0 at all its hops. Under these circumstances, the protocol behaves exactly like standard SPVP. Conversely, whenever a message (P, α) is received and $\alpha > 0$, there must exist some node along path P which is not selecting its most preferred route. It is possible to prove that, given a dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$, the global precedence metric will eventually force one vertex u_i to advertise path Q_i with a non-zero precedence value. The dispute wheel Π will then be prevented from oscillating as path $R_{i-1}Q_i$ will have a higher precedence than path Q_{i-1}. Hence, no dispute wheel will be able to generate a persistent oscillation, yielding the following results:

Theorem 3.23 *Enforcement of the global precedence metric prevents oscillations in* SPVP *[ERC+07].*

Theorem 3.24 *If a policy-based protocol oscillation persists, then some* SPVP *updates will forcedly advertise a non-zero precedence value [ERC+07].*

The Cost of Guaranteed Convergence

The two modifications to SPVP we discussed in this section work by either removing (SPVP-3) or re-ranking (global precedence) some specific paths. This corresponds to sacrificing strict enforcement of the locally defined routing policy for the sake of guaranteed convergence. In other words, some ASes will lose part of the autonomy that allowed them to independently specify their own routing policies. Quantifying such a loss of autonomy is intrinsically a hard task. However, both solutions adopt a lazy approach: the fix is automatically applied only when an oscillation is actually detected. Under normal circumstances, no restrictions are applied and the standard SPVP protocol is used.

The possibility of influencing the choice at a neighboring AS by forging a crafted BGP message allows misbehaving ASes to maliciously affect the selection of particular paths at a victim AS. To some extent, this scenario is partially already realizable in SPVP by advertising spoofed paths. However, since ASes can autonomously rank paths according to their own needs, the possibility of announcing forged paths does not directly cause a loss of autonomy. Unfortunately, both SPVP-3 and global precedence exacerbate this problem: the protocol has no way to prevent misbehavior, and a misbehaving AS can force a neighbor to filter (SPVP-3) or rerank (global precedence) a route. Moreover, detecting misbehaving ASes is, in general, a very hard task. This is one of the major reasons driving the study and experimentation of BGP variants that aim at adding some degree of security to the protocol [KLMS00].

One important difference between the two solutions is the way transient oscillations are dealt with. Recall that SPVP-3 filters a path as soon as it detects a cycle in the corresponding path history. This implies that even transient oscillations, i.e. oscillations that are dependent on particular message timings and thus have extremely small probability of lasting, will be fixed by automatically installing permanent filters at some ASes. In comparison, the global precedence approach allows much more flexibility: transient oscillations will be handled by forcing a local reranking via the global precedence attribute, however, in the steady state, the global precedence value will influence rankings only if there ex-

	SPVP-3	**Global Precedence**
message overhead	large	small
loss of autonomy	automatic filters	automatic rankings
transient oscillations	permanent filtering	temporary reranking
misbehavior	no prevention, no detection	no prevention, limited detection [1]

Table 3.2: A qualitative comparison between two oscillation-free SPVP variants.

ists a persistent oscillation which is being prevented (Theorem 3.24). Table 3.2 summarizes the comparison between the two SPVP variants we introduced in this section.

Proposing variations of BGP that are guaranteed to converge is continuously attracting the attention of researchers. General guidelines for the design of policy-based path vector protocols are given in [GJR03]. A variation of BGP that allows a router to customize the route selection on behalf of each neighbor has been presented in [WSR09]. Modifications to iBGP are proposed in [CGM03, MC04a, KCM04].

In [HW08] is described a variant of the SPVP model where vertices assign fractional weights to paths instead of simply selecting them. [HW08] shows that every instance of this model admits a solution, and [Kin08] describes a distributed protocol to compute a stable solution where rankings are within a factor ϵ from the optimum.

Many other contributions have been proposed in the literature to modify BGP in such a way to improve its convergence properties [AKS06, BBAS03, LXHL02, PZW+02, ZAL04]. However, a detailed discussion of those approaches is out of the scope of this thesis.

[1] The detection process involves further modifications to the protocol, and only works for special kinds of misbehavior.

Chapter 4

Characterization of eBGP Safety Under Filtering*

4.1 Introduction and Related Work

BGP provides Autonomous Systems (ASes) with the *autonomy* to set routing policies independent of each other, and with the *expressiveness* to specify extremely complex configurations. However, autonomy and expressiveness come at the expense of guaranteed convergence. In particular, a BGP configuration could never reach a stable routing, either because a stable state for that configuration does not exist at all, or because the protocol gets "trapped" into bad event timings. This is highly undesirable, since it has been observed that interdomain routing changes can cause performance degradation and packet loss [WMW+06], and continuously changing routing can severely affect the availability of services [KKK07]. The need to avoid such disadvantages has spurred significant research efforts on BGP stability.

Varadhan et al. [VGE00] showed that autonomy in configuring routing policies can lead to persistent routing oscillations, and proposed constraints to be applied to routing policies in order to achieve *safety*, i.e., stability under any timings of routing events. A number of fundamental contributions on this topic

*Part of the material presented in this chapter is based on the following publication

- L. Cittadini, G. Di Battista, M. Rimondini, S. Vissicchio. Wheel + Ring = Reel: the Impact of Route Filtering on the Stability of Policy Routing. In *Proc. International Conference on Network Protocols (ICNP 2009)*, IEEE, 2009.

are due to Griffin et al. [GW99, GW00, GSW99, GSW02]. Among the results they presented, those works showed how the dynamic behavior of BGP can be related to characteristics of the BGP configuration that can be statically analyzed. In particular, in [GSW02] it is shown that the absence of a *dispute wheel* (DW), a cyclic pattern of routing preferences, is sufficient to guarantee safety.

The "no DW" condition is a cornerstone in the literature on BGP stability. As an example, Gao et al. [GR00, GGR01] used the absence of DWs to prove that, if routing policies are specified consistently with the commercial relationships between ASes, safety is guaranteed.

In [kC06] Chau took into account the general case in which non-strict path rankings can be expressed. Even in this case, the absence of DWs is fundamental to guarantee safety.

Feamster et al. [FJB07] explored the impact of autonomy and expressiveness on the stability of the BGP protocol by distinguishing the roles of the *ranking* and *filtering* components of routing policies. Ranking allows an AS to specify preferences over multiple candidate routes to the same destination, while filtering allows an AS to selectively advertise specific routes to specific neighbors. A crucial question is posed in [FJB07]: "provided that each AS retains complete autonomy and complete filtering expressiveness, how expressive can rankings be while guaranteeing stable routing?". This question is formalized by the concept of *safety under filtering* (Problem 3.4). A configuration is safe under filtering if it is safe under any combination of route filters. A necessary condition for safety under filtering is the absence of a particular subclass of DWs, called *dispute rings* [FJB07].

In this chapter, we make three main contributions. First, we show a necessary and sufficient condition for safety under filtering, filling the large gap between previously known necessary and sufficient conditions. To the best of our knowledge, this is the first characterization of stability in policy routing under realistic assumptions about the autonomy of ASes. Our result is based on the presence of a structure called *dispute reel* (DR), which is both a special case of a DW *and* a generalization of a dispute ring. Dispute reels inherit from DWs the interesting property of depending on routing policies alone. Hence, checking for the presence of a DR does not require to delve into the details of BGP dynamics.

Second, we show that, in a network admitting multiple stable routing states, safety under filtering is provably compromised. In particular, whenever the existence of multiple stable states is detected, we provide a way for network operators to pinpoint the portions of the BGP configuration which define a

DR (thus making the configuration not safe under filtering). Observe that this implies that the so called BGP wedgies [TG05] are an hallmark for unsafety under filtering.

Third, we show that robustness (Problem 3.3) does not necessarily imply safety under filtering. *Robustness* is the property of a configuration to be safe under any combination of link/node failures [GSW02]. It is known that safety under filtering implies robustness (see Property 3.4). We explore the relationship between those two properties by showing that the opposite does not hold. In a sense, this proves that the autonomy of adding (possibly misconfigured) filters can do more harm than network faults.

The popularity of DWs in the literature on the stability of policy-based protocols is mostly due to the fact that the "no DW" condition implies the existence of a unique stable routing state [GSW02], safety [GSW02], robustness [GSW02], and safety under filtering [FJB07]. As a side effect of our work, we show that DRs can replace DWs, giving raise to less constraining sufficient conditions for all those properties.

The chapter is structured as follows. Section 4.2 defines the concept of dispute reel. Section 4.3 (4.4) proves that the absence of a dispute reel is a necessary (sufficient) condition for safety under filtering. In Section 4.5 we discuss the relationship between safety under filtering and robustness. Conclusions are drawn in Section 4.6.

4.2 Wheel + Ring = Reel

It is shown in [FJB07] that safety under filtering can be studied by analyzing structural properties of the policy configuration, without the need to deal with the details of dynamic evaluation. The main known structural properties that are related to safety under filtering are based on the absence of cyclic dependencies among routing preferences, which are called dispute wheels and dispute rings. We briefly define these two concepts using the SPVP model (see Chapter 2).

As we discussed in Section 3.2, a *dispute wheel* (DW) [GSW02] $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ is a triple consisting of a sequence of nodes $\vec{\mathcal{U}} = (u_0\ u_1 \dots u_{k-1})$ and two sequences of nonempty paths $\vec{\mathcal{Q}} = (Q_0\ Q_1 \dots Q_{k-1})$ and $\vec{\mathcal{R}} = (R_0\ R_1 \dots R_{k-1})$ such that for each $i = 0, \dots, k-1$ we have:

(i) R_i is a path from u_i to u_{i+1}

(ii) $Q_i \in \mathcal{P}^{u_i}$

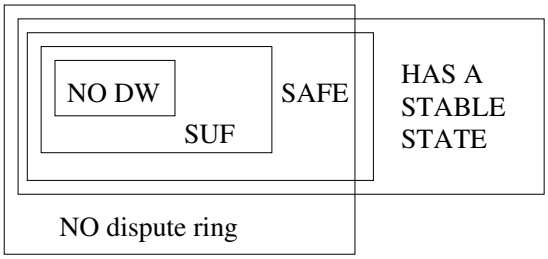


Figure 4.1: The absence of a dispute ring (wheel) is a necessary (sufficient) condition for safety under filtering [FJB07].

(iii) $R_i Q_{i+1} \in \mathcal{P}^{u_i}$

(iv) $\lambda^{u_i}(R_i Q_{i+1}) \leq \lambda^{u_i}(Q_i)$

We call vertices u_i *pivot* vertices, paths Q_i *spoke* paths, and paths R_i *rim* paths. Throughout the chapter, we intend subscripts of vertices and paths in a dispute wheel to be interpreted modulo k where $k = |\vec{\mathcal{U}}|$. The absence of a dispute wheel implies safety under filtering (see Theorem 3.14).

Feamster et al. show in [FJB07] that the absence of a particular class of dispute wheels, called *dispute rings*, is a necessary condition for safety under filtering. A dispute ring is a dispute wheel having at least three pivot vertices, and such that each vertex appears only once in the wheel We refer to Section 3.4 for a more formal definition of dispute ring. Figure 4.1 shows how the "no dispute wheel" and "no dispute ring" conditions relate to the properties of an SPVP instance. We stress that there is a large gap between the two conditions, as the absence of a dispute ring does not guarantee safety, and does not even imply that the SPVP instance admits a stable path assignment.

We now define a dispute reel as a special case of dispute wheel. Intuitively, a reel is a dispute wheel such that the spoke paths form a tree T and each rim path R_i contains no vertex in T except u_i and u_{i+1}. In order to formally define the dispute reel, we use the notation $P[v]$ to denote the sub-path of P starting at vertex v, that is, $P = (u \dots v)P[v]$. This implies $P[0] = (0)$ for any P.

Definition 4.1 *A* dispute reel *(DR) is a dispute wheel which satisfies the following conditions:*

(i) *(*Pivot vertices appear in exactly three paths*) – for each $u_i \in \vec{\mathcal{U}}$, u_i only appears in paths Q_i, R_i and R_{i-1}.*

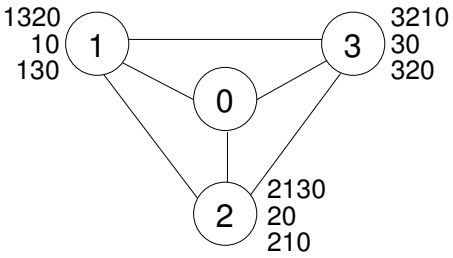


Figure 4.2: An SPVP instance, showed in [FJB07], which is safe under filtering but contains DWs. However, none of these DWs is a DR.

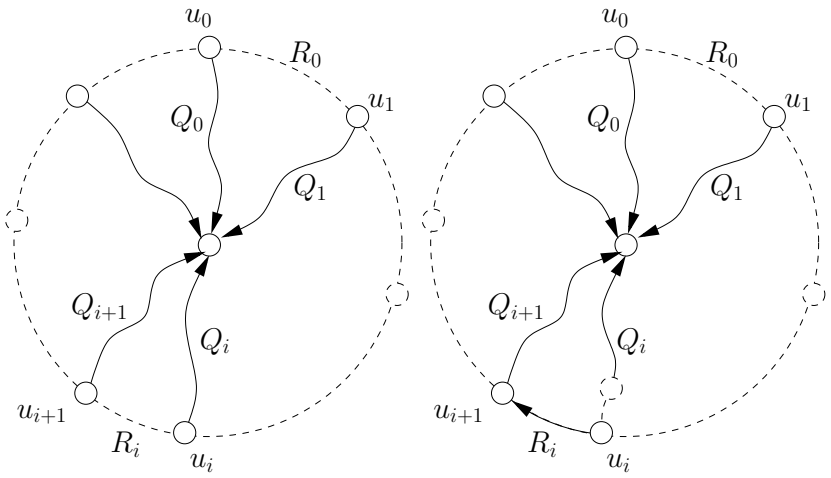
(ii) *(*Spoke and rim paths do not intersect*) – for each* $u \notin \vec{\mathcal{U}}$*, if* $u \in Q_i$ *for some* i*, then no* j *exists such that* $u \in R_j$*.*

(iii) *(*Spoke paths form a tree*) – for each distinct* $Q_i, Q_j \in \vec{\mathcal{Q}}$*, if* $v \in Q_i \cap Q_j$*, then* $Q_i[v] = Q_j[v]$*.*

We stress that the existence of a DR does not depend at all on the protocol dynamics, i.e., it is a structural property of the policy configuration that can be statically checked. It is easy to check that DISAGREE (Figure 2.1c) is an example of a DR. Conversely, the instance in Figure 4.2, first used in [FJB07] to show that the presence of a DW does not prevent an instance from being safe under filtering, does not contain any DRs. As an example, a DW Π exists in Figure 4.2 where pivot vertices are $\vec{\mathcal{U}} = (1\ 2\ 3)$, spoke paths are $\vec{\mathcal{Q}} = ((1\ 0)\ (2\ 0)\ (3\ 0))$, and rim paths are $\vec{\mathcal{R}} = ((1\ 3\ 2)\ (2\ 1\ 3)\ (3\ 2\ 1))$. However, pivot vertex 1 appears in all rim paths, thus violating Condition *(i)* of Definition 4.1. On the other hand, the instance in Figure 4.2 also contains the DW Π' where pivot vertices are $u_0 = 1$ and $u_1 = 2$, spoke paths are $Q_0 = (1\ 3\ 0)$ and $Q_1 = (2\ 0)$, and rim paths are $R_0 = (1\ 3\ 2)$ and $R_1 = (2\ 1)$. Π' too is not a DR because Condition *(ii)* is not satisfied, as vertex 3 appears both in Q_0 and in R_0. Similar arguments can be applied to the other DWs in the instance in Figure 4.2.

An even simpler dispute wheel is the *dispute duo*.

Definition 4.2 *A* dispute duo *is a dispute reel such that* $|\vec{\mathcal{U}}| = 2$ *and* $R_0 \cap R_1 = \{u_0, u_1\}$*.*



(a) All-spoke path assignment $\bar{\pi}$. (b) One-rim path assignment $\bar{\pi}^i$.

Figure 4.3: Two special path assignments of a dispute reel. The selected paths are highlighted using solid stroke. Note that in $\bar{\pi}^i$, u_i is the only vertex in Q_i which is not selecting a sub-path of Q_i.

The simple structure of DRs allows us to identify two classes of activation sequences leading to two "natural" classes of path assignments. Given an SPVP instance S containing a DW Π, the *supporting instance* $S[\Pi]$ *of* Π is the minimal SPVP instance which contains the vertices, edges and paths of Π. Intuitively, $S[\Pi]$ can be obtained from S by filtering all paths but those used in the DW. Observe that, if Π is a DR, then in $S[\Pi]$ pivot vertices have exactly two permitted paths, and vertices along the spoke paths (except pivots) have exactly one permitted path.

Let S be an SPVP instance containing a DR Π and let $S[\Pi]$ be the supporting instance of Π. The *all-spoke* path assignment (see Figure 4.3a) is a path assignment $\bar{\pi}$ such that $\bar{\pi}(u) = Q_i[u]$ if $u \in Q_i$, $\bar{\pi}(u) = \epsilon$ otherwise. Since spoke paths form a tree, by activating the edges of each spoke path Q_i in reverse order (starting from 0) it is easy to construct an activation sequence σ_{spoke} leading to an all-spoke path assignment.

Similarly, we define the *one-rim* path assignment *for pivot* u_i (see Fig-

ure 4.3b) as a path assignment $\bar{\pi}^i$ such that:

$$\bar{\pi}^i(u) = \begin{cases} Q_j[u] & \text{if } u \in Q_j, u \neq u_i \\ R_i[u]Q_{i+1} & \text{if } u \in R_i \\ \epsilon & \text{otherwise.} \end{cases}$$

In order to build an activation sequence that leads to $\bar{\pi}^i$, we can extend σ_{spoke} by activating the edges of R_i in reverse order (starting from u_{i+1}). This is always possible because rim paths never intersect spoke paths and for each non-pivot vertex along R_i, $\bar{\pi}(v) = \epsilon$.

4.3 Safety Under Filtering implies no DR

In this section we show that the absence of DRs is a necessary condition for safety under filtering. We do this by showing that the presence of a DR in an SPVP instance S makes S not safe under filtering. The proof consists of three parts. First, we show that if S contains a dispute duo, then S is not SUF (Lemma 4.1). Second, we generalize this result by stating that if S contains a DR consisting of two pivot vertices, then S is not SUF (Lemma 4.2). Last, we show that if an instance S contains a DR Π, then an oscillation can always be constructed, either by cycling through one-rim path assignments on Π (Lemma 4.3), or by exploiting a different DR consisting of two pivot vertices (Lemma 4.4). Thus, S is not safe under filtering.

Dispute Reels with 2 Pivots

We start by showing that the presence of a dispute reel having 2 pivot vertices makes an SPVP instance not safe under filtering. First, we generalize the routing oscillation showed in Table 2.2 for DISAGREE to the broader class of dispute duos.

Lemma 4.1 *An* SPVP *instance that contains a dispute duo is not safe under filtering.*

Proof: Let S be an SPVP instance containing a dispute duo $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ and consider $S[\Pi]$. We now construct a fair activation sequence that induces an oscillation on $S[\Pi]$. The main idea is that vertices u_0 and u_1 can simultaneously select paths $\pi(u_0) = R_0Q_1$ and $\pi(u_1) = R_1Q_0$. Path assignment π is clearly not stable as u_0 and u_1 are both convinced that the other vertex is offering a

feasible path to 0. For this reason, the two pivot vertices will eventually fall back on their spoke paths Q_0 and Q_1. By iterating this argument, we are able to show an infinite fair activation sequence.

First of all, since Π is a DR, we can construct on $S[\Pi]$ an activation sequence that leads to the all-spoke path assignment π_{t_1} at some time t_1. We now propagate the announcement of path Q_1 (respectively, Q_0) by activating the edges along R_0 (R_1) in reverse order. Since R_0 and R_1 have no shared vertices other than u_0 and u_1, the two announcements cannot interfere with each other. We halt one hop before the announcement of Q_1 (Q_0) reaches u_0 (u_1). Formally, let $R_0 = (v_0\ v_1 \dots v_k)$, where $v_0 = u_0$ and $v_k = u_1$. We activate edges in R_0 in reverse order until we hit v_1, that is,

$$\sigma_{R_0} = (\{(v_k, v_{k-1})\}\ \{(v_{k-1}, v_{k-2})\} \dots \{(v_2, v_1)\}).$$

Symmetrically, let $R_1 = (w_0\ w_1 \dots w_j)$, and consider the sequence

$$\sigma_{R_1} = (\{(w_j, w_{j-1})\}\ \{(w_{j-1}, w_{j-2})\} \dots \{(w_2, w_1)\}).$$

We activate edges according to σ_{R_0}, and then according to σ_{R_1}. Then, we simultaneously activate edges (v_1, v_0) and (w_1, w_0). Observe that the simultaneous activation of edges (v_1, v_0) and (w_1, w_0) makes path R_0Q_1 available at u_0, and path R_1Q_0 available at u_1. It is easy to check that these activations lead to a path assignment π_{t_2} such that, for $i \in \{0, 1\}$:

$$\pi_{t_2}(u) = \begin{cases} Q_i[u] & \text{if } u \in Q_i, u \neq u_i \\ R_i[u]Q_{i+1} & \text{if } u \in R_i \end{cases}$$

We now activate edges in R_0 (R_1) in reverse order, again halting at v_1 (w_1), and then we simultaneously activate edges (v_1, v_0) and (w_1, w_0). By doing so, vertex u_0 (u_1) withdraws the availability of path Q_0 (Q_1). Since R_0 and R_1 do not have vertices in common other than u_0 and u_1, the withdrawal will eventually reach vertex u_1 (u_0). Vertex u_1 (u_0) will then fall back on path Q_1 (Q_0). Observe that we have now reached the all-spoke path assignment π_{t_3}, which implies $\pi_{t_3}(u) = \pi_{t_1}(u)$ for every vertex u. Since we can iterate this argument, it is clear that there exists an infinite activation sequence. Moreover, no announcement is delayed indefinitely, i.e., the activation sequence is also fair on $S[\Pi]$. The proof is completed by noting that $S[\Pi]$ can be obtained by S through path filtering, hence we conclude that S is not SUF. □

Lemma 4.1 can be generalized, as DRs having two pivot vertices always imply the existence of a dispute duo. As an example, consider the instance in

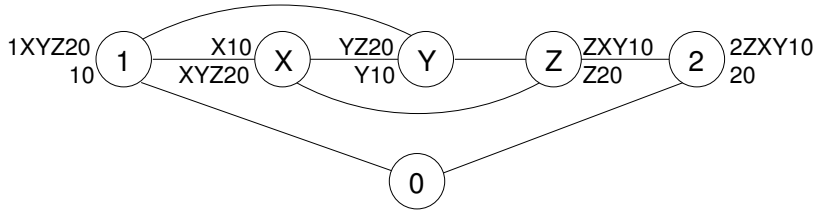


Figure 4.4: An SPVP instance containing a DR consisting of two pivot vertices (1 and 2) and whose rim paths intersect at vertices X, Y, and Z.

Figure 4.4. Clearly, this instance contains a DR having $u_0 = 1$ and $u_1 = 2$ as pivot vertices, $Q_0 = (1\ 0)$ and $Q_1 = (2\ 0)$ as spoke paths, and $R_0 = (1\ X\ Y\ Z\ 2)$ and $R_1 = (2\ Z\ X\ Y\ 1)$ as rim paths. Notice that both rim paths traverse vertices X, Y, and Z. We now search for a dispute duo. Walk along R_1 and stop at the last vertex which is in $R_1 \cap R_0$, that is, Y. By analyzing λ^Y, it is easy to see that there exists another DR having Y and 2 as pivot vertices, $(Y\ 1\ 0)$ and $(2\ 0)$ as spoke paths, and $(Y\ Z\ 2)$ and $(2\ Z\ X\ Y)$ as rim paths. Note that the rim paths of this DR do not intersect at vertex X. We now repeat the process on the new DR, considering vertex Z. It is easy to see that there exists a dispute duo having Z and Y as pivot vertices. The following lemma generalizes the approach we just showed to any DR having two pivot vertices.

Lemma 4.2 *An* SPVP *instance that contains a dispute reel having exactly 2 pivot vertices is not safe under filtering.*

Proof: Let S be an SPVP instance containing a dispute reel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$, with $|\vec{\mathcal{U}}| = 2$. First, we show that the presence of Π implies that S contains a dispute duo Π', then we use Lemma 4.1 to argue that S is not SUF.

If R_0 and R_1 do not share any vertices except u_0 and u_1, then Π is a dispute duo and the statement directly follows from Lemma 4.1. Otherwise, let $\{v_1, \ldots v_k\}$ be the vertices in $R_0 \cap R_1 - \{u_0, u_1\}$, in the same order as they appear in R_0. That is, $R_0 = (u_0 \ldots v_1 \ldots v_k \ldots u_1)$, where $\forall i\ v_i \in R_1$. Let v_j be the "rightmost" vertex in R_1 among vertices $\{v_1, \ldots, v_k\}$, and let $P = R_1[v_j]$. More formally, v_j is such that $v_i \notin P\ \forall i \neq j$. We now show that either there exists a dispute duo Π' having u_0 and v_j as pivot vertices, or there exists a DR Π'' consisting of two pivot vertices v_j and u_1 and having strictly less intersections between its rim paths than Π.

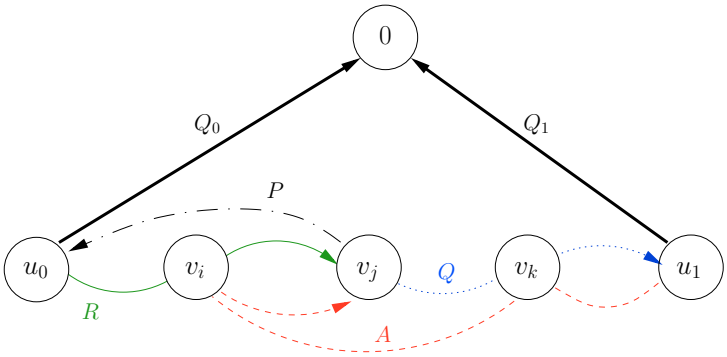


Figure 4.5: A dispute reel having 2 pivot vertices. Rim paths $R_0 = RQ$ and $R_1 = AP$ are split as explained in the proof of Lemma 4.2. Different paths are represented using different strokes. In particular, spoke paths Q_0 and Q_1 are in thicker stroke.

Refer to Figure 4.5. Split R_1 and R_2 such that $R_1 = A(v_j)P$ and $R_0 = R(v_j)Q$.

Since we are considering $S[\Pi]$ and $v_j \in R_0 \cap R_1$, $\mathcal{P}^{v_j} = \{PQ_0, QQ_1\}$. Depending on the ranking at vertex v_j and since (by construction) we cannot have $\lambda^{v_j}(PQ_0) = \lambda^{v_j}(QQ_1)$, we have two possible cases.

(i) $\lambda^{v_j}(PQ_0) < \lambda^{v_j}(QQ_1)$. We now show that $\Pi' = ((u_0\ v_j), (Q_0\ QQ_1), (R\ P))$ is a dispute duo. By construction, Π' has only two pivot vertices, and $P \cap R = \{u_0, v_j\}$. Observe that u_0 appears only in Q_0, R and P, while v_j appears only in QQ_1, R, and P. Therefore, Condition *(i)* of Definition 4.1 is satisfied. Condition *(ii)* is also satisfied, since $Q_0 \cap R = Q_0 \cap P = \{u_0\}$ and $Q_1 \cap R = Q_1 \cap P = \oslash$ are guaranteed by the fact that Π is a DR. Moreover, by construction, $Q \cap R = Q \cap P = \{v_j\}$. Finally, Condition *(iii)* holds for paths Q_0 and Q_1 since Π is a DR, and $Q \cap Q_0 = \oslash$.

(ii) $\lambda^{v_j}(PQ_0) > \lambda^{v_j}(QQ_1)$. We now show that $\Pi'' = ((v_j\ u_1), (PQ_0\ Q_1), (Q\ A))$ is a dispute reel. Since $v_j \neq u_0$ by construction, Π'' has strictly less intersections between rim paths than Π. Observe that v_j appears only in PQ_0, Q, and A, while u_1 appears only in Q_1, Q, and A. Hence, Condition *(i)* of Definition 4.1 is satisfied. Condition *(ii)* is also satisfied, since $Q_0 \cap Q = Q_0 \cap A = \oslash$ and $Q_1 \cap Q = Q_1 \cap A = \{u_1\}$ are guaranteed by the fact that Π is a dispute reel. By construction, $P \cap Q = P \cap A = \{v_j\}$.

Finally, Condition *(iii)* holds for paths Q_0 and Q_1 since Π is a DR, and $P \cap Q_1 = \emptyset$.

Hence, in the first case we find a dispute duo Π'. In the second case, we find another dispute reel Π'' having two pivot vertices and having strictly less intersections between rim paths than Π. By iterating this argument, we eventually end up finding a dispute duo. We then use the result from Lemma 4.1 to prove that an instance containing a DR with two pivot vertices is not safe under filtering. □

Dispute Reels with more than 2 Pivots

The next step is to show that the presence of a dispute reel having more than two pivot vertices makes an SPVP instance not safe under filtering. We prove that in two parts. First, we introduce the concept of a "rim-by-rim" dispute reel, that is, a DR for which it is easy to construct a routing oscillation. Second, we show that the presence of a dispute reel which is not rim-by-rim implies the existence of a dispute reel having only two pivot vertices.

Given a DR $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$, with $|\vec{\mathcal{U}}| = k > 2$, we say that Π is *rim-by-rim* if $\forall i \in \{0, \ldots, k-1\}$ there exists an activation sequence σ_i on $S[\Pi]$ such that $\bar{\pi}^i \overset{\sigma_i}{\rightsquigarrow} \bar{\pi}^{i+1}$. That is, starting from the one-rim path assignment for any pivot u_i, σ_i leads to the one-rim path assignment for pivot u_{i+1}. The following property is a straightforward consequence of the definition of rim-by-rim DR.

Property 4.1 *σ_i activates all the edges in R_{i+1} at least once.*

Observe that the well known instance Bad-Gadget in Figure 2.1b is a trivial rim-by-rim DR. More generally, any dispute ring can be viewed as a special case of rim-by-rim DR. Feamster et al. show in [FJB07] that it is particularly easy to find an oscillation on a dispute ring. We are now able to generalize that result to the broader class of rim-by-rim DRs.

Lemma 4.3 *An* SPVP *instance containing a rim-by-rim dispute reel is not safe under filtering.*

Proof: Let S be an SPVP instance containing a rim-by-rim dispute reel Π. Using the fact that Π is rim-by-rim, we build an infinite fair activation sequence in the supporting instance $S[\Pi]$ that cycles indefinitely among one-rim path assignments.

As we have already seen, since Π is a dispute reel there exists an activation sequence on $S[\Pi]$ that induces a one-rim path assignment $\bar{\pi}^i$ for an arbitrary pivot u_i.

Since Π is rim-by-rim, there exist activation sequences σ_j such that $\bar{\pi}^i \overset{\sigma_i}{\rightsquigarrow} \bar{\pi}^{i+1} \overset{\sigma_{i+1}}{\rightsquigarrow} \dots \overset{\sigma_{i-1}}{\rightsquigarrow} \bar{\pi}^i$. Note that the initial and final path assignments are the same, thus we can iterate the same set of activations in order to create an infinite activation sequence σ on $S[\Pi]$. By Property 4.1, edges traversed by rim paths are activated at least once per iteration. To ensure fairness, at the end of each iteration we activate edges according to σ_{spoke} without altering the current path assignment. This implies that there exists an infinite fair activation sequence on $S[\Pi]$, hence S is not safe under filtering. □

Now consider the instance in Figure 4.6. Clearly, this instance contains a DR Π where pivot vertices are $u_0 = 1$, $u_1 = 2$, and $u_2 = 3$; spoke paths are $Q_0 =$ (1 0), $Q_1 =$ (2 0), and $Q_2 =$ (3 0); and rim paths are $R_0 =$ (1 X Y W Z 2), $R_1 =$ (2 Z W X Y 3), and $R_2 =$ (3 1). Π is not rim-by-rim: in particular, no activation sequence exists that, starting from the one-rim path assignment for pivot u_0 ($\bar{\pi}^0$), makes path R_1Q_2 available at vertex 2. In fact, assume that the instance is in state $\bar{\pi}^0$, that is, vertices 2 and 3 select their spoke paths, while vertices on R_0 select a sub-path of R_0Q_1. In particular, vertex 1 selects path (1 X Y W Z 2 0). We now explore how far the announcement of path (3 0) can be propagated along rim path R_1. Suppose that vertex 3 announces path (3 0) to Y. Since path (Y 3 0) is preferred, Y selects the new path and propagates the announcement to X. Observe that, even if X does not prefer path (X Y 3 0), Y's announcement withdraws the availability of the previously selected path (X Y W Z 2 0). Hence, X propagates the announcement further to W. Now, W does not change its choice, since path (W X Y 3 0) is less preferred. It is easy to see that there is no way to propagate the announcement further than vertex W. Nevertheless, the rankings at vertex W are such that there exists a DR having W and 2 as pivot vertices. The following lemma shows that the presence of a DR having two pivot vertices is actually a general property of any DR which is not rim-by-rim. By using Lemma 4.2, we are then able to show an oscillation even on DRs that are not rim-by-rim.

Lemma 4.4 *An* SPVP *instance containing a dispute reel which is not rim-by-rim is not safe under filtering.*

Proof: Let S be an SPVP instance containing a dispute reel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ which is not rim-by-rim. If $|\vec{\mathcal{U}}| = 2$, the statement follows from Lemma 4.2. Otherwise, consider $S[\Pi]$. Since Π is not rim-by-rim by hypothesis, there are at

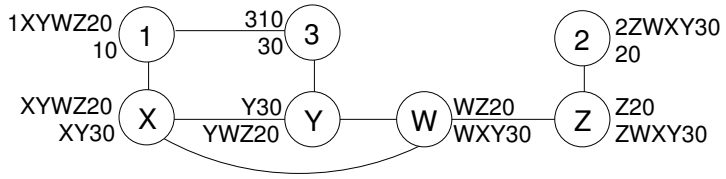


Figure 4.6: A DR which is not rim-by-rim. Vertex 0 is omitted for brevity.

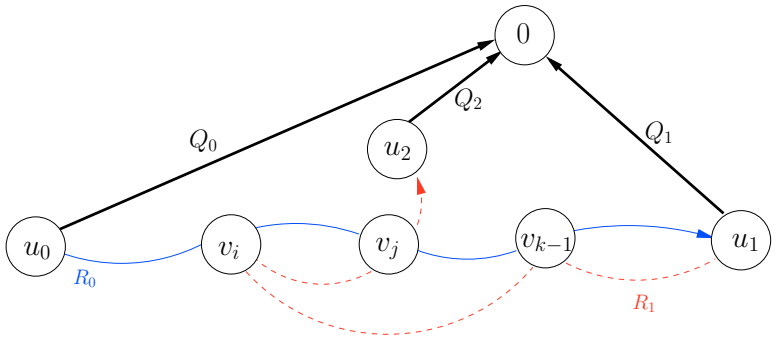


Figure 4.7: A portion of a dispute reel which is not rim-by-rim, used in the proof of Lemma 4.4. Different paths are represented using different strokes. Spoke paths Q_0, Q_1, and Q_2 are in thicker stroke.

least $\bar{\pi}^i$ and $\bar{\pi}^{i+1}$ such that $\nexists\sigma : \bar{\pi}^i \overset{\sigma}{\rightsquigarrow} \bar{\pi}^{i+1}$. Assume, without loss of generality, that $i = 0$.

Let $\{v_1, \ldots v_k\}$ be the vertices of $R_0 \cap R_1$, in the same order as they appear in R_0, that is, $R_0 = (u_0 \ldots v_1 \ldots v_k)$, where $v_k = u_1$, as showed in Figure 4.7.

Let Σ be the set of all the activation sequences that, starting from the one-rim path assignment $\bar{\pi}^0$, make path Q_2 available in the set of choices of some vertex v_m. More formally, $\forall\sigma \in \Sigma$, $\bar{\pi}^0 \overset{\sigma}{\rightsquigarrow} \pi_t$, where $R_1[v_m]Q_2 \in \text{choices}_t(v_m)$ for some m and t. Note that Σ contains at least the activation sequence obtained by activating the edges of R_1 in reverse order, which would lead to $R_1[v_j]Q_2 \in \text{choices}_t(v_j)$, where v_j is the common vertex that is the "rightmost" in R_1, that is, $\forall i \neq j$, $v_i \notin R_1[v_j]$. Consider the activation sequence $\sigma' \in \Sigma$ such that v_m has the highest index. We now show that, if the announcement of path Q_2 reaches vertex u_1, then we have a contradiction. In

fact, if $v_m = u_1$, we would have $\bar{\pi}^0 \overset{\sigma'}{\rightsquigarrow} \pi_t$, where $\pi_t(u_1) = R_1 Q_2$. This enable us to activate the edges in R_0 in reverse order, withdrawing the availability of path Q_1 on all the vertices along R_0, and eventually reaching state $\bar{\pi}^1$. This contradicts the hypothesis that $\nexists \sigma : \bar{\pi}^0 \overset{\sigma}{\rightsquigarrow} \bar{\pi}^1$.

Hence, $v_m \neq u_1$. We now prove that, if the announcement of path Q_2 cannot be propagated further than v_m, then we have a dispute reel having two pivot vertices. Consider the path ranking at vertex v_m. We have two cases:

(i) $\lambda^{v_m}(R_1[v_m]Q_2) \leq \lambda^{v_m}(R_0[v_m]Q_1)$. We now show that there exists an activation sequence $\bar{\sigma} \in \Sigma$ that makes path Q_2 available in the set of choices of $v_{m'}$, with $m' > m$, hence a contradiction. Intuitively, v_m can announce path $R_1[v_m]Q_2$ to withdraw the availability of path $R_0[v_m]Q_1$ to the vertices on R_0. This allows the announcement of path Q_2 to be propagated beyond vertex v_m. Observe that, since path $R_1[v_m]Q_2$ is in $\text{choices}_t(v_m)$ and it is preferred, we must have $\pi_t(v_m) = R_1[v_m]Q_2$ after activation sequence σ'. Let σ_1 consist of the activations of all the edges in R_0 in reverse order, starting from v_m. Let π_{t_1} be the path assignment after σ_1, that is, $\pi_t \overset{\sigma_1}{\rightsquigarrow} \pi_{t_1}$. Note that π_{t_1} is such that path $R_0[v_h]Q_1$ has been withdrawn at each v_h, $h < m$. We now construct σ_2 by activating the edges along R_1 in reverse order. In this way, v_m propagates the announcement of path $R_1[v_m]Q_2$. Clearly, if a vertex v_h, $h < m$, receives the announcement, it will select path $R_1[v_h]Q_2$, since the set of choices at v_h is currently empty. Hence, the announcement will be propagated further. This implies that the message will eventually reach vertex $v_{m'}$, $m' > m$.

(ii) $\lambda^{v_m}(R_1[v_m]Q_2) > \lambda^{v_m}(R_0[v_m]Q_1)$. We now show that there exists a dispute reel having v_m and u_1 as pivot vertices. Let $\bar{R}$ be the sub-path of R_1 from u_1 to v_m, that is, $R_1 = \bar{R}R_1[v_m]$. Now consider the dispute wheel $\Pi' = ((v_m\ u_1), (R_1[v_m]Q_2\ Q_1), (R_0[v_m]\ \bar{R}))$. We now show that Π' is a DR. Being Π a DR, Condition *(i)* of Definition 4.1 holds since $v_m \notin Q_1$ and $u_1 \notin R_1[v_m]Q_2$. Condition *(ii)* is trivially satisfied by vertices on paths Q_1 and Q_2, because both are spoke paths in Π. By definition, $\bar{R} \cap R_1[v_m] = \{v_m\}$. Moreover, $R_1[v_m] \cap R_0[v_m] = \{v_m\}$, since, by definition of v_m, $v_j \notin R_1[v_m]$ if $j > m$, and $v_j \notin R_0[v_m]$ if $j < m$. Again, being Π a DR, Condition *(iii)* holds for paths Q_1 and Q_2, and we have $R_1[v_m] \cap Q_1 = \oslash$.

We then conclude that if Π is not rim-by-rim, then it contains a dispute reel having two pivot vertices. By Lemma 4.2, instance S is not safe under

filtering. □

By combining Lemmas 4.2, 4.3, and 4.4, we can state the following theorem.

Theorem 4.1 *An* SPVP *instance containing a dispute reel is not safe under filtering.*

Multiple Solutions and Safety Under Filtering

We now exploit Theorem 4.1 to show that networks admitting multiple stable states are not safe under filtering. Since multiple stable states happen in practice (see, e.g., BGP wedgies [TG05]), this is especially interesting from an operational perspective.

Theorem 4.2 *If an* SPVP *instance S admits two stable states, then S is not safe under filtering.*

Proof: Theorem V.4 in [GSW02] proves that S must contain a dispute wheel Π. Π is derived by merging two stable path assignments π_1 and π_2. Let T_1 and T_2 be the routing trees induced by π_1 and π_2, and let $T = T_1 \cap T_2$. Each spoke path in Π is composed by a path along T plus a final edge which does not connect two vertices in T. Hence, spoke paths form a tree (Condition *(iii)* of Definition 4.1). Rim paths are built up by vertices which are not in the intersection of π_1 and π_2, thus Condition *(ii)* is also satisfied. Each pivot vertex u_i can only appear in Q_i, R_i, and R_{i-1} (Condition *(i)*), since the dispute wheel is built using only $\pi_1(u_i)$ and $\pi_2(u_i)$. Therefore, Π is a dispute reel. By Theorem 4.1, the presence of a dispute reel in S is enough to conclude that S is not SUF. □

An important consequence of Theorem 4.2 is that observing multiple different stable routing states in a network indicates that its stability may be definitively compromised by the application of route filters. Therefore, the existence of multiple stable states in a network constitutes an important alert to consider for a network operator. As a final remark, we stress that the construction presented in Theorem V.4 of [GSW02] can be exploited to identify a portion of the network which can potentially lead to oscillations under filtering. Moreover, given a set of stable routing states, implementing that construction is straightforward and can be done efficiently. Network operators can use the technique in [GSW02] to disclose a policy dispute in the routing configuration. Our results prove that the presence of such a policy dispute makes the network not SUF.

4.4 No DR implies Safety Under Filtering

We now show that the absence of a dispute reel is a sufficient condition for safety under filtering. Combined with the result from the previous section, we can conclude that the presence of a DR characterizes safety under filtering. We prove the sufficient condition by showing that if an SPVP instance is not SUF, then it contains a DR. First, we use the same technique as in [GSW02] to show that a routing oscillation implies the existence of a particular kind of dispute wheel, which satisfies a slightly different set of conditions than those in Definition 4.1. Then, we show that the presence of such a dispute wheel implies the existence of a dispute reel.

Lemma 4.5 *Consider an* SPVP *instance S. If S is not safe under filtering, then there exists a dispute wheel Π which satisfies the following conditions:*

(i) *Conditions* (ii) *and* (iii) *of Definition 4.1.*

(ii) *For all $u_i \in \vec{\mathcal{U}}$, u_i cannot appear in Q_j, $j \neq i$.*

(iii) *If $u_i \in R_j$, then $R_j[u_i]Q_{j+1}$ is preferred to Q_i.*

Proof: Since S is not SUF, there exists a combination of filters inducing an instance S' such that S' is not safe. We can then apply the technique described in Theorem V.9 of [GSW02] to show that S' contains a dispute wheel Π satisfying the above conditions. The statement follows by noting that Π must also be present in S. □

Observe that the dispute wheel of Lemma 4.5 is not a DR. In particular, it could be the case that a pivot vertex u_i appears in a rim path R_m with $m \notin \{i-1, i\}$. The following lemma shows that such a DW implies the existence of a DR.

Lemma 4.6 *Given an instance S, suppose it contains a dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ satisfying the conditions in the statement of Lemma 4.5. Then, S contains a dispute reel.*

Proof: If Π is already a DR, the statement trivially holds. Otherwise, for Π not to be a reel, there must exist at least a pivot vertex u_i such that $u_i \in R_m$ with $m \notin \{i-1, i\}$. Let $R_{i_1}, \ldots, R_{i_k}$ be the rim paths traversing u_i, where $i_j \notin \{i-1, i\}$. Without loss of generality, assume that $i_k < i$ is the closest index to i in the order induced by $\vec{\mathcal{U}}$, see Figure 4.8. Condition *(iii)* of Lemma 4.5

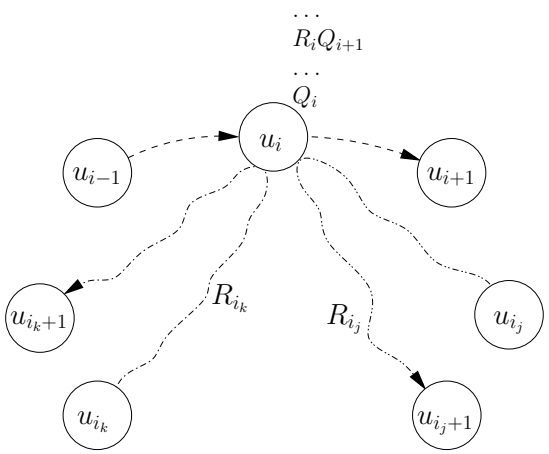


Figure 4.8: A dispute wheel where pivot vertex u_i appears in rim paths other than R_i and R_{i-1}. By Lemma 4.6, another dispute wheel can be constructed such that u_i appears in exactly 3 paths.

ensures that u_i prefers path $R_{i_k}[u_i]Q_{i_k+1}$ to Q_i. Now consider the dispute wheel $\Pi' = (\vec{\mathcal{U}}', \vec{\mathcal{Q}}', \vec{\mathcal{R}}')$, where $\vec{\mathcal{U}}' = (u_i\ u_{i_k+1} \dots u_{i-1})$, $\vec{\mathcal{Q}}' = (Q_i\ Q_{i_k+1} \dots Q_{i-1})$, and $\vec{\mathcal{R}}' = (R_{i_k}[u_i]\ R_{i_k+1} \dots R_{i-1})$. Intuitively, Π' is obtained by "chopping" Π, using path $R_{i_k}[u_i]$ as the new rim path associated with vertex u_i. Observe that every spoke path in Π' is a also spoke path in Π. Moreover, every rim path in Π' except $R_{i_k}[u_i]$ is also a rim path in Π, and $R_{i_k}[u_i]$ is a sub-path of R_{i_k}. Therefore, Π' trivially satisfies all the conditions of Lemma 4.5. Moreover, by the definition of index i_k, we know that Π' is such that u_i only appears in Q_i, $R_{i_k}[u_i]$ and R_{i-1}. By applying this construction, we force one pivot vertex at a time to satisfy Condition *(i)* of Definition 4.1, even if R_{i_k} contains other pivot vertices than u_i. Hence, after iterating the construction at most $|\vec{\mathcal{U}}|$ times, we eventually end up with a dispute reel. □

We stress that Condition *(iii)* of Lemma 4.5 is strictly necessary to apply the construction in Lemma 4.6. As a counterexample, consider again the instance in Figure 4.2. In this instance, a DW $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ exists where $\vec{\mathcal{U}} = (1\ 2\ 3)$, $\vec{\mathcal{Q}} = ((1\ 0)\ (2\ 0)\ (3\ 0))$, and $\vec{\mathcal{R}} = ((1\ 3\ 2)\ (2\ 1\ 3)\ (3\ 2\ 1))$. Observe that Π only violates Condition *(iii)* of Lemma 4.5. In fact, rim path (1 3 2) traverses pivot

vertex 3, but $\lambda^3((3\ 2\ 0)) > \lambda^3((3\ 0))$. It is easy to check that, in this case, no DR can be constructed starting from the DW.

Theorem 4.3 *If an* SPVP *instance S is not safe under filtering, then it contains a dispute reel.*

Proof: Lemma 4.5 ensures that S contains a dispute wheel satisfying some particular constraints. We can then apply Lemma 4.6 to find a dispute reel in S. □

By combining Theorems 4.1 and 4.3, we conclude that **the absence of a dispute reel is a sufficient and necessary condition for safety under filtering**.

Researchers have deemed the dispute wheel concept important because it only depends on the routing policies. As such, it allows us to prove fundamental properties of the SPVP protocol using just static analysis, i.e., without having to cope with the details of routing dynamics. In fact, the absence of a dispute wheel implies that an SPVP instance is safe under filtering (Corollary 1 of [FJB07]) and has a unique stable state (Theorem V.4 of [GSW02]). Obviously, as safety and robustness can be viewed as special cases of safety under filtering, the absence of a dispute wheel also implies that an SPVP instance is safe and robust. Figure 4.1 is a Venn diagram that effectively displays those implications.

As a side effect of our findings, we show that a "no DR" condition can replace the well known "no DW" one in all the above results: in fact, "no DR" is a strictly less constraining condition to show that an SPVP instance is safe, robust, SUF, and has a unique stable state. Moreover, this condition still depends only on the structure of routing policies.

Corollary 4.1 *The absence of a DR in an* SPVP *instance S implies that S has a unique stable state, is safe, and is robust.*

Proof: Theorem 4.2 proves that S has a unique stable state. Since safety and robustness are special cases of safety under filtering, Theorem 4.3 proves the rest of the statement. □

4.5 Safety Under Filtering and Robustness

Safety under filtering is an extremely useful concept to study the impact of route filters on routing stability. An interesting related problem is the impact

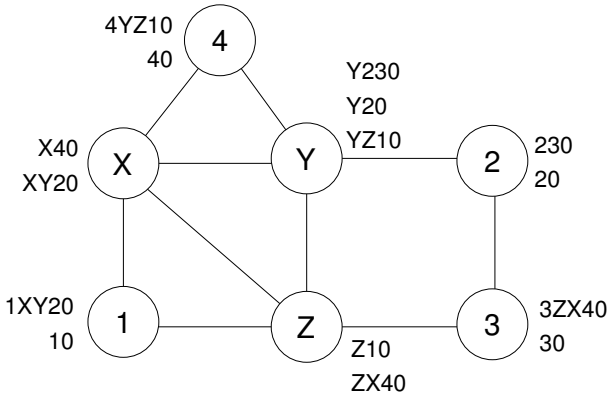


Figure 4.9: FILTHY-GADGET: an instance which is robust but not safe under filtering. Vertex 0 is omitted for brevity.

of link and/or router failures on the safety of BGP. The property of being safe after removing any subset of the vertices or edges from an SPVP instance is referred to as *robustness* (Problem 3.3). Without loss of generality, in the following we only consider link failures.

As pointed out in Property 3.4, an instance that is SUF is also robust. Following the findings of Section 4.4, we now show that the class of robust SPVP instances is strictly larger than the class of instances that are SUF. Consider the instance FILTHY-GADGET in Figure 4.9. This instance is clearly not SUF since it contains a DR $\bar{\Pi} = (\bar{\mathcal{U}}, \bar{\mathcal{Q}}, \bar{\mathcal{R}})$, where $\bar{\mathcal{U}} = (1\ 2\ 3\ 4)$, $\bar{\mathcal{Q}} = ((1\ 0)\ (2\ 0)\ (3\ 0)\ (4\ 0))$, and $\bar{\mathcal{R}} = ((1\ X\ Y\ 2)\ (2\ 3)\ (3\ Z\ X\ 4)\ (4\ Y\ Z\ 1))$. Yet, FILTHY-GADGET is robust. We prove the latter statement in two parts: first, we show that FILTHY-GADGET is safe; second, we show that any combination of link failures produces a safe instance.

To prove the first part, we need the following definition. A vertex v is said to be *prevented from selecting path* P if, for every fair activation sequence, there exists a time t' such that v does not select P (i.e., $\pi_t(v) \neq P$) for any $t > t'$.

Lemma 4.7 *Instance* FILTHY-GADGET *is safe.*

Proof: Let σ be any fair activation sequence. Given that $\pi_t(0) = (0)$ for all t, by the fairness of σ each neighbor of 0 is prevented from selecting path ϵ. In particular, after some time vertex 2 can only use paths (2 3 0) or (2 0).

Since Y accepts both paths from vertex 2, Y is prevented from selecting path (Y Z 1 0), which is less preferred. Vertex 4 is therefore prevented from selecting path (4 Y Z 1 0). Since 4 is a neighbor of 0, it is also prevented from selecting ϵ. Hence, by the fairness of σ, vertex 4 will end up selecting path (4 0) permanently, in turn forcing vertex X to permanently choose path (X 4 0). Since path (X Y 2 0) will not be advertised by X, vertex 1 is prevented from selecting path (1 X Y 2 0). Also, being 1 a neighbor of 0, it will end up selecting path (1 0) permanently. Vertex Z, in turn, will be forced to select path (Z 1 0), preventing vertex 3 from selecting (3 Z X 4 0). By applying the same argument as above, we conclude that vertex 3 will permanently select path (3 0). Hence, vertex 2 will select path (2 3 0), in turn forcing vertex Y to select (Y 2 3 0). It is easy to check that the path assignment induced by σ is stable. Since we did not make any hypothesis on σ, we conclude that FILTHY-GADGET is guaranteed to reach this stable path assignment for any fair activation sequences, that is, FILTHY-GADGET is safe. □

Lemma 4.8 *Instance* FILTHY-GADGET *is robust.*

Proof: By the previous lemma, we know that FILTHY-GADGET is safe. We now show that any instance S' obtained by removing one or more links from FILTHY-GADGET contains no DR, hence it is safe. Recall that FILTHY-GADGET contains the DR $\bar{\Pi}$ we described above. It is easy to see that its supporting instance $S[\bar{\Pi}]$ is built on the same graph as FILTHY-GADGET. Hence, removing one or more links forcedly creates an instance where $\bar{\Pi}$ does not exist anymore. In order to complete the proof, we need to demonstrate that $\bar{\Pi}$ is the only DR in FILTHY-GADGET. Observe that this is trivially true if vertices X, Y and Z are not pivot vertices. We now show that no DR $\Pi' = (\vec{\mathcal{U}}', \vec{\mathcal{Q}}', \vec{\mathcal{R}}')$ exists having X, Y, or Z as a pivot vertex.

(i) Assume that X is a pivot vertex of Π'. Without loss of generality, we say $X = u'_0$. Then $Q'_0 = (X\ Y\ 2\ 0)$ and $R'_0 = (X\ 4)$, which implies $u'_1 = 4$. Since (Z 1 0) is the best ranked path at vertex Z, we have either $u'_2 = Y$ or $u'_2 = 1$. The former case results in a dispute wheel where spoke path Q'_0 contains a pivot node $u'_2 = Y$. The latter case results in a DW where spoke path Q'_0 shares vertex Y with rim path R'_1. In both cases, Π' cannot be a DR.

(ii) We can apply a symmetric argument to vertex Z. Assume that Z is a pivot vertex of Π', $Z = u'_0$. Then $Q'_0 = (Z\ X\ 4\ 0)$ and $R'_0 = (Z\ 1)$, which

implies $u'_1 = 1$. As above, if $u'_2 = X$ or $u'_2 = 2$, we find that Π' cannot be a DR. The only other possibility is $u'_2 = Y$, i.e., Y is also a pivot vertex. This case is discussed in the following.

(iii) Assume that Y is a pivot vertex of Π'. Without loss of generality, we say $Y = u'_i$. We have two cases, namely $Q'_i = (Y\ Z\ 1\ 0)$ or $Q'_i = (Y\ 2\ 0)$.

- if $Q'_i = (Y\ Z\ 1\ 0)$, then $u'_{i-1} = 4$. We now have either $u'_{i-2} = X$ or $u'_{i-2} = 3$. The former case implies that Q'_{i-2} contains pivot vertex Y. The latter case implies that R'_{i-2} intersects Q'_i at vertex Z. Hence, Π' cannot be a DR.
- if $Q'_i = (Y\ 2\ 0)$, then $u'_{i-1} = 1$. We now have either $u'_{i-2} = Z$ or $u'_{i-2} = 4$. The former case implies that Q'_{i-2} and R'_{i-1} share vertex X. The latter case implies that pivot vertex Y also appears in R'_{i-2}. In both cases, Π' cannot be a DR.

We conclude that $\bar{\Pi}$ is the only DR in FILTHY-GADGET, hence the instance is robust. □

4.6 Conclusions

Under the realistic assumption that ASes are allowed to filter routes arbitrarily, the safety of policy-based routing is intrinsically incompatible with unrestricted route rankings. This chapter characterizes safety under filtering, determining the amount of autonomy that rankings must sacrifice in order to guarantee stable policy routing. The significance of this result is twofold: on one hand, we fill the large gap that separates currently known necessary and sufficient conditions; on the other hand, we bind safety under filtering to the presence of a particular structure of routing preferences, called dispute reel, which can be statically detected.

An interesting consequence of our results is that a network admitting multiple stable routing states (e.g., BGP wedgies [TG05]) is not safe under filtering. In this case, we can also pinpoint the problematic portions of the policy configuration, even in the case where we do not know the policies of all ASes: in fact, the technique only takes the stable routing states as its inputs.

We finally show that a robust instance may not be safe under filtering. In a sense, this proves that the autonomy of adding (possibly misconfigured) filters can be more harmful than network faults. Finally, as a side effect of our work,

we show that the less constraining "no dispute reel" condition can replace the "no dispute wheel" one in a lot of results in the field of policy routing stability.

Chapter 5

The Impact of Changing iBGP Attributes on Routing Stability*

5.1 Introduction and Related Work

BGP configuration languages allow border routers to change iBGP attributes that are relevant to the route selection process. However, both theoretical [GW02a] and practical [FRBS08] research contributions neglected this peculiar feature of iBGP, assuming that those iBGP attributes which are relevant to the BGP decision process (e.g., the `local-preference` attribute) are not changed as the BGP message is passed to iBGP peers.

In this chapter we investigate the possibility of changing iBGP attributes, trying to answer the following questions:

(i) What are the pros and cons of changing iBGP attributes? Why should an ISP (not) configure its routers to modify iBGP data en route?

(ii) Do ISPs actually change iBGP attributes?

(iii) How does this possibility relate to iBGP stability?

*Part of the material presented in this chapter is based on the following publication

- L. Cittadini, G. Di Battista, S. Vissicchio. Doing Don'ts: Modifying BGP Attributes within an Autonomous System. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*, IEEE, 2010.

(iv) Can we profitably change some attributes in iBGP to enforce traffic engineering policies while preserving iBGP stability?

First, we discuss possible advantages of changing iBGP attributes and related caveats. Second, by analyzing BGP update traces collected at multiple vantage points in the Internet, we estimate the number of ISPs that are actually changing iBGP attributes: our data show that this practice is adopted by few ISPs. Third, we revisit a well-known theoretical model to analyze iBGP stability [GW02a], extending it to support iBGP attributes that change within an ISP. We use this extended model to prove that changing iBGP attributes makes iBGP prone to new types of oscillations. Fourth, given that state-of-the-art algorithms to detect oscillations [FRBS08] assume that iBGP messages are left untouched, we show a technique that does not rely on this assumption. We exploit this technique to build a tool that is able to statically check an iBGP configuration for stability. Results with a prototype implementation show promising performance, hence we conclude that changing iBGP attributes does not intrinsically prevent a network operator from debugging its routing policies using advanced configuration analyses. Finally, we state configuration guidelines to change iBGP attributes in a rational and systematic way. Our guidelines are easy to configure on routers, guarantee iBGP stability even under faulty conditions, and ensure that reasonable traffic engineering policies are enforced, regardless of the behavior of other ISPs.

The rest of the chapter is organized as follows. Section 5.2 covers background notions about iBGP. We outline the main pros and cons of changing iBGP attributes within an AS in Section 5.3, and we estimate the extent to which iBGP attributes are actually changed by ISPs in Section 5.4. Section 5.5 analyzes the impact of changing iBGP attributes on routing stability. In Section 5.6, we devise guidelines to modify iBGP attributes while preserving stability. Conclusions are drawn in Section 5.7.

5.2 Background

BGP configuration languages allow operators to modify the attributes carried by a message in order to influence the best route selection process (see Table 1.1) and therefore control outbound traffic. Some commands can even force a BGP speaker to skip some steps of the BGP decision process (see, e.g., Cisco `bgp bestpath AS-path ignore` command).

Internal BGP (*iBGP*) is used by an ISP in an Autonomous System (AS) to distribute the routes that are learned from external ASes among its border

Route learned from	Distribute to clients	Distribute to non-clients
eBGP neighbor	yes	yes
client	yes	yes
non-client	yes	no

Table 5.1: Route propagation rules for an iBGP speaker.

routers. We refer to the manipulation of an attribute in an iBGP message as iBGP attribute changing (IAC). Observe that IAC implicitly takes into account the possibility to skip some BGP decision steps. As an example, skipping Step 2 has the same effect of overwriting each `AS-path` with a constant string.

The original design of BGP mandated a full mesh of iBGP peerings within an AS in order to distribute the routes received from external ASes. However, the scaling issues of such a solution spurred the search for alternatives. The most common and widespread alternative to fully meshed iBGP is route reflection [BCC06]. Route reflection organizes BGP routers within an AS in a hierarchy of *clusters*. The iBGP neighbors of each router are split into two sets: *clients* and *non-clients*. In a fully meshed iBGP network, all iBGP routers are non-clients. A router that has one or more clients acts as a *route reflector*, i.e., it relays routing information to its clients. An iBGP speaker propagates its best route according to the rules depicted in Table 5.1: if the best route is learned from a non-client iBGP peer, then it is relayed only to clients, otherwise it is propagated to all iBGP neighbors. Each cluster has (at least) one route reflector. In order to ensure that routes are correctly distributed within the AS, there must be a full mesh of iBGP peerings at the top of the route reflection hierarchy.

We now define the concept of *valid signaling path*, which models route dissemination across the route reflection hierarchy. This concept is needed to define iBGP topology connectivity. Intuitively, the rules in Table 5.1 ensure that iBGP route distribution follows the topology of the route reflection hierarchy. Formally, let $G = (V, E)$ be the topology of the route reflection hierarchy. Namely, each node $u \in V$ represents an iBGP router, and each edge $e \in E$ represents an iBGP peering. The set of edges is partitioned into two sets **over** and **up-down**. An edge $(u, v) \in$ **over** represents the fact that v is a non-client of u and u is a non-client of v. Hence, an **over** edge indicates a vanilla iBGP peering between routers u and v. An edge $(u, v) \in$ **up-down** represents the fact that either v is a non-client of u while u is a client of v, or vice versa. Hence,

an **up-down** edge (u, v) indicates an iBGP peering between routers u and v where v (u) acts as a route reflector for u (v). We say that an **up-down** edge is **up** when it is traversed from the client to its route reflector, **down** otherwise. A *valid signaling path* is any path on G that can be used to disseminate a route within the AS, according to the rules in Table 5.1. Any valid signaling path P consists of: *(i)* a (possibly empty) sequence of **up** edges, followed by *(ii)* a (possibly missing) **over** edge, followed by *(iii)* a (possibly empty) sequence of **down** edges [GW02a].

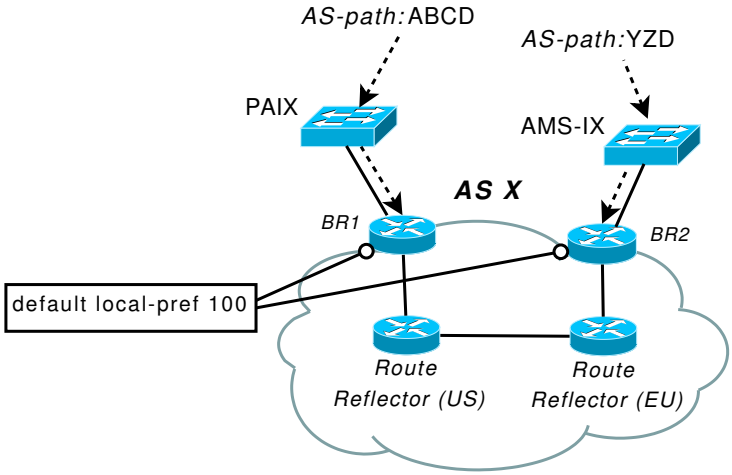
An iBGP topology is *connected* if there exists a valid signaling path between every pair of iBGP speakers. Intuitively, a connected iBGP topology ensures that routing information can be propagated by any iBGP speaker to any other. Throughout the thesis, we only consider connected iBGP topologies.

5.3 Why or Why Not?

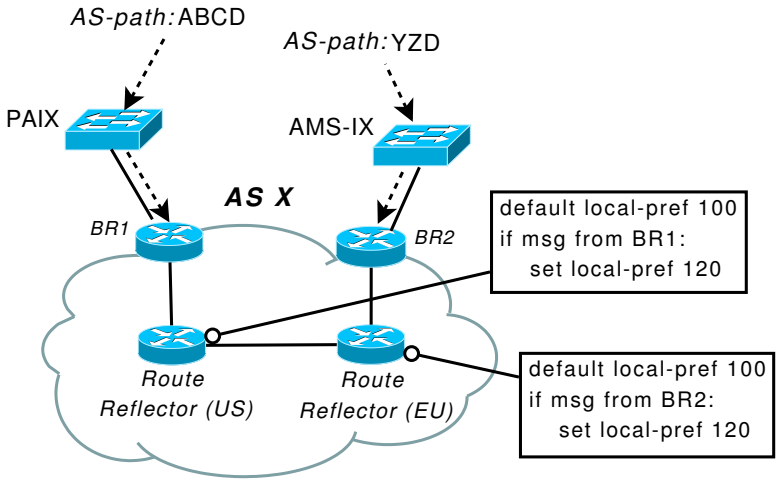
This section presents the possibilities opened by changing iBGP attributes and the drawbacks this practice can incur. We will assume the viewpoint of a single ISP managing its AS.

The main reason why a network operator might think about modifying iBGP attributes within his AS is the extended flexibility this practice allows. Operators can exploit this flexibility for implementing policies which are otherwise impossible to enforce. Figure 5.1a provides a simple example where AS X spans over North America and Europe, and has public peerings at Internet exchange points (IXPs) in Palo Alto (PAIX) and Amsterdam (AMS-IX). Configurations described in figures are expressed in an intuitive vendor-independent pseudo-language and are trivial to translate to any vendor-specific language. Since AS X has multiple border routers in geographically distributed locations, it employs route reflectors in order to scale its iBGP configuration. For the purpose of this example, we assume that AS X has, among others, a route reflector somewhere in the US and another one in Europe, and that route reflectors are connected in a full-mesh of iBGP peerings. Being a large ISP, X is likely to exhibit high route diversity [MFM+06], that is, multiple routes for the same destination prefix p are likely available at multiple border routers. Suppose that X receives two BGP routes for prefix p: *(i)* a BGP route advertising path $ABCD$ from a peer at PAIX, and *(ii)* another BGP route advertising path YZD from a peer at AMS-IX.

Assuming that X assigns `local-preference` values according to business relationships [GR00, CR05], the received routes are assigned the same value



(a)



(b)

Figure 5.1: (a) Default BGP configuration causes sub-optimal traffic forwarding in AS X: outbound traffic is routed through AMS-IX, due to the length of the `AS-path` attribute. (b) By changing iBGP attributes, AS X is able to exploit both AMS-IX and PAIX as traffic egress points, achieving better load balancing.

since they both come from a peer. For this reason, the two routes are equally ranked from the first step of the BGP decision process. The next step of the process evaluates the length of the `AS-path` attribute: since the path received at AMS-IX is shorter than the path received at PAIX, every BGP router will prefer the former, which implies that all the traffic directed to p will be forwarded to Amsterdam.

Observe that AS X does not get any revenue from traffic transiting over IXPs, so its best strategy would be to minimize the cost of traffic forwarding. Since routers in the US must forward traffic towards Europe while they could simply send traffic to Palo Alto, the high-level business objective of minimizing costs seems to be not well implemented by the BGP configuration described above. Such an objective would be better accomplished if X was able to send traffic from US out of Palo Alto and from Europe out of Amsterdam, reducing the usage of cables connecting US and Europe.

Unfortunately, this simple requirement cannot be implemented (within the standard BGP decision process) unless X splits its network into multiple AS domains. On the other hand, if X performs IAC, it is fairly simple to force the route reflector in America to prefer American routes, and the route reflector in Europe to prefer European routes, as shown in Figure 5.1b. By conditionally changing the value of the `local-preference` attribute (e.g., via `route-map`s), this configuration enforces the high-level objective regardless of what `AS-path`s are announced by X's neighbors.

We analyzed the BGP updates received from the border routers of a medium-sized Italian ISP and we inferred that more than 135 thousands IP prefixes (almost half routing table) were load-balanced across exit points just because of equal `AS-path` lengths. Should the `AS-path` length vary on one of the available routes (e.g., because of new connectivity or because the AS that originates the prefix is performing inbound traffic engineering activities via `AS-path` prepending), the traffic balance would be immediately compromised. People that operate that ISP were not aware that at least 20% of their traffic is actually load balanced this way.

To better understand how a traffic shift would look like, recall the example in Figure 5.1a, and now suppose that the European peer of AS X started advertising an `AS-path` of length 5 or more. As soon as this new route is propagated within AS X, the American route is preferred, and traffic destined to prefix p is completely forwarded via Palo Alto.

After showing that there exist benefits in manipulating iBGP attributes, we turn to study the drawbacks and caveats of IAC. It is a common practice not to touch iBGP attributes (see Section 5.4), to keep the configuration as

simple and easy to understand as possible. Typically, a policy is only applied when a BGP route enters or exits the AS and iBGP is just used to distribute routes within the AS. This ensures consistent AS-wide BGP decisions, and significantly simplifies the task of translating business objectives into BGP configurations.

Another important drawback of changing iBGP attributes is that it exacerbates the iBGP stability problem, as the added flexibility can translate into the ability to create routing oscillations which would be impossible otherwise. Due to its impact, this disadvantage is discussed in depth in Section 5.5.

5.4 Changing iBGP Attributes in the Internet

Given that changing iBGP attributes provides some advantages to ISPs, as we described in the previous section, one might ask whether this practice is common in the Internet, and to what extent. Unfortunately, an exact answer to this question would require access to router configuration files, which most ISPs refuse to grant as they do not want to disclose their routing policies. However, in this section, we give a method to roughly estimate the popularity of *IAC* using public data.

In [FR07] it is shown that applying policies only to routes announced by eBGP peers implies that only routes that are equally good up through the first three steps of the BGP decision process (see Table 1.1) can be selected by iBGP speakers as best routes in the steady state. The main intuition behind our measurement approach is then to search for two BGP routers in the same AS that are selecting distinct routes which are not equally good up through to the first three decision steps. In such a case, assuming a connected iBGP topology, we conclude that *IAC* is performed within the AS.

Figure 5.2 shows a real-world example of the list of BGP routes available for destination prefix 189.90.12.0/24 in the Global Crossing network (AS 3549), as reported by a publicly available route server on August, 31^{st} 2009, at $14:36$ UTC. Each entry in the list (delimited by a box in the figure) represents a BGP route. The first line of each entry represents the `AS-path` attribute, then other attributes follow, e.g., `local-preference`, `origin`, etc. Note that all routes were received from iBGP peers, as they include iBGP-only attributes like `cluster-list`. This implies that each route was selected as best by the corresponding iBGP peer. Observe that the first and the third entries have different `AS-path` lengths (see the highlighted text in Figure 5.2), so they are not equally good up through Step 3 of the BGP decision process. Since the

```
ROUTE-SERVER.PHX1>SH IP BGP 189.90.12.0/24
BGP ROUTING TABLE ENTRY FOR 189.90.12.0/24
PATHS: (4 AVAILABLE, BEST #1)
  NOT ADVERTISED TO ANY PEER
  13878 15180 28189
     67.17.64.89 FROM 67.17.80.210 (67.17.80.210)
        ORIGIN IGP, METRIC 0, LOCALPREF 300, BEST
        COMMUNITY: 3549:4471 3549:30840
        ORIGINATOR: 67.17.81.221,
        CLUSTER LIST: 0.0.0.92
  13878 15180 28189
     67.17.64.89 FROM 67.17.82.130 (67.17.82.130)
        ORIGIN IGP, METRIC 0, LOCALPREF 300
        COMMUNITY: 3549:4471 3549:30840
        ORIGINATOR: 67.17.81.221,
        CLUSTER LIST: 0.0.0.92
  28189 28189 28189 28189 28189 28189 28189
     67.17.64.89 FROM 67.17.82.40 (67.17.82.40)
        ORIGIN IGP, METRIC 0, LOCALPREF 300
        COMMUNITY: 3549:4950 3549:34076
        ORIGINATOR: 200.186.0.67,
        CLUSTER LIST: 0.0.2.109, 0.0.5.2
  28189 28189 28189 28189 28189 28189 28189
     67.17.64.89 FROM 67.17.82.41 (67.17.82.41)
        ORIGIN IGP, METRIC 0, LOCALPREF 300
        COMMUNITY: 3549:4950 3549:34076
        ORIGINATOR: 200.186.0.67,
        CLUSTER LIST: 0.0.2.109, 0.0.5.2
```

Entry #1

Entry #2

Entry #3

Entry #4

Figure 5.2: A set of BGP routes that are simultaneously active within AS 3549.

routes are simultaneously active at two distinct iBGP routers, we conclude that the ISP performs IAC. Of course, another possible explanation is that the iBGP topology of the ISP is not connected. However, this sharply contrasts with the objective iBGP is designed for.

For a quantitative analysis of how many ASes show this behavior in the Internet, we used the technique described in [DRCD09] for computing the sets of BGP routes for the same destination prefix which are simultaneously active in the same AS, taking as input BGP routing tables and update traces provided by RIS [RIP] and Routeviews [Ore] through May 2009. Then, when we found routes having different `AS-path` length among those that are simultaneously active at AS A, we inferred that AS A was changing iBGP attributes within its network. Our analysis estimated that $1,838$ ASes out of $32,066$ (0.17%) change iBGP attributes.

Note that our estimate is actually a lower bound with respect to the real number of ASes that change iBGP attributes in the Internet. First of all, since we only have some hundreds of publicly available BGP monitors, our data do not reliably represent the full route diversity that is available in the Internet. Secondly, we only focused on the `AS-path` length, disregarding other attributes that are involved in later steps of the BGP decision process. Nevertheless, our estimate confirms that the majority of ASes apply policies only to eBGP sessions and then rely on the iBGP topology just to distribute routing information within the network. However, adopting the classification of the ASes given in [DD08], we found that many of the $1,838$ ASes are transit providers. This could be explained by the fact that provider ASes have traffic engineering needs that are more complex to fulfill than those of customers.

5.5 More Flexibility implies More Instability

Policy-based path vector protocols such as BGP are renowned to be prone to oscillations [GSW02] and, unfortunately, iBGP makes no exception [GW02a]. In this section, we use the SPVP model (see Chapter 2) to study how IAC can improve or degrade the stability of the protocol. For the sake of simplicity, we exclude from our analysis the `MED` attribute. In fact, our analysis is easy to extend to deal with `MED` adopting techniques similar to those in [GW02b].

We now show how to construct an instance $S(X, t, p)$ of SPVP which models a given iBGP configuration for AS X at time t, with respect to a given destination prefix p, assuming that iBGP attributes can be changed within the AS. The set of nodes consists of node 0 and one node for each iBGP speaker

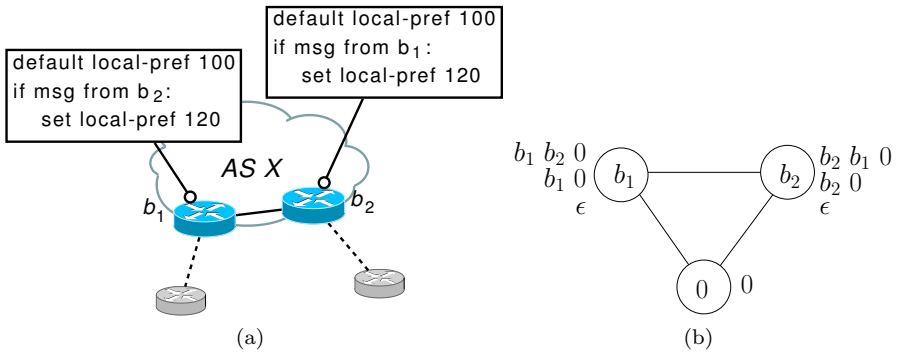


Figure 5.3: (a) Configuration of two border routers that modify iBGP attributes. (b) The corresponding translation to SPVP.

in X. Observe that some of these iBGP speakers are border routers while some others are route reflectors. There is an edge (u, v) for each iBGP peering between iBGP speakers u and v. Moreover, there exists an edge $(u, 0)$ for each border router u that has an eBGP path to prefix p at time t. At node $u \neq 0$, the set of permitted paths consists of the empty path ϵ and all paths $(u \ \ldots \ v \ 0)$ where $(v, 0)$ is an edge and $(u \ \ldots \ v)$ is a valid signaling path (see Section 5.2) from u to v. If border router u has multiple eBGP paths to prefix p at time t, permitted path $(u \ 0)$ represents the best among them, according to the standard BGP decision process. Permitted paths at node u are ranked according to the iBGP configuration of router u and the BGP decision process. Since Step 6 of the BGP decision process evaluates IGP metrics, we assume that these metrics are known.

Observe that our construction is more general than the one proposed in Section 5.1 of [GW02a], where rankings are determined by only relying on IGP metrics, since iBGP attributes are supposed to be the same at every node.

Figure 5.3a depicts a simple iBGP configuration, while Figure 5.3b shows the corresponding translation to SPVP, where each node u is equipped with a list of paths representing $\mathcal{P}^u$, sorted according to λ^u (better paths are positioned higher in the list). For example, the list besides node b_1 specifies that b_1 can use paths $(b_1 \ b_2 \ 0)$ and $(b_1 \ 0)$ to reach 0, and prefers $(b_1 \ b_2 \ 0)$. The opposite happens at vertex b_2. The following theorem shows that *IAC* exacerbates the iBGP routing stability problem.

Theorem 5.1 *BGP configurations that allow iBGP attribute changing can generate a larger set of oscillations than BGP configurations where iBGP attributes are not modified.*

Proof: Trivially, the former family of BGP configurations strictly includes the latter, and as such it can generate at least the same set of routing oscillations. We now show that the iBGP topology in Figure 5.3a cannot oscillate if iBGP attributes are not allowed to be changed within the AS. Let P_i be the best eBGP route received by b_i. We now walk through the BGP decision process at routers b_1 and b_2, examining all possible cases.

- P_1 and P_2 have different `local-preference` values. In this case, the one with the highest value is eventually selected at both routers.
- P_1 and P_2 have different `AS-path` lengths. Assuming a tie in the first decision step (otherwise, we fall in the previous case), the route with the shortest length is eventually selected at both routers.
- P_1 and P_2 have different `origin` values. Again, assuming a tie in the previous decision steps, the route with the lowest origin is eventually selected at both routers.
- P_1 and P_2 have the same `origin` value. In this case, Step 5 of the BGP decision process implies that router b_i eventually selects P_i, $i \in \{1, 2\}$.

In every case, no oscillations can be generated. On the other hand, consider the iBGP configuration of Figure 5.3a. The corresponding SPVP instance in Figure 5.3b is the well known DISAGREE gadget (compare with Figure 2.1c), which is renowned to possibly exhibit oscillations if messages are exchanged simultaneously between routers b_1 and b_2 (see Table 2.2). □

5.6 Profitable iBGP Attribute Modification

Sections 5.3 and 5.5 suggest that an ISP willing to change iBGP attributes within its own network essentially faces a trade-off between flexibility and stability. In this section, we define policy configuration guidelines that safely exploit the flexibility of modifying iBGP attributes. The main concern here is to obtain benefits in terms of traffic load balancing (see, e.g., Figure 5.1b), while ensuring routing stability and keeping the complexity of BGP configuration manageable.

Our guidelines are meant to fulfill two main high level requirements: *(i)* Routes should be ranked according to revenues and costs; and *(ii)* Internal transit cost, i.e., the cost of forwarding traffic within the ISP network, should be minimized.

We assume that the neighbors of an ISP can be broadly classified, according to commercial relationships among ISPs, into customers, peers, and providers [GR00]. Selecting a route announced by a customer means forwarding traffic to that customer, which pays for it. Similarly, selecting a route announced by a peer implies that traffic is exchanged free of charge between the two ISPs. Selecting a route announced by a provider, instead, involves paying a cost. We then implement requirement *(i)* by mandating that customer routes have an higher `local-preference` than peer routes that, in turn, have an higher `local-preference` than provider routes. Moreover, to avoid offering transit service for free, routes learned from a peer or a provider are not exported to other peers or providers. This is one of the most typical way of expressing routing policies in BGP [CR05] and it provides the additional benefit of ensuring global interdomain routing stability [GR00]. Requirement *(ii)* is implemented by forcing each route reflector to prefer routes learned from its own clients, assuming that the cost of sending traffic from a route reflector to a client is less than the one of sending traffic to a non-client. This is very frequently the case, as route reflection topology design should be congruent with the network topology [BCC06].

Guideline A *Every iBGP speaker assigns a local preference value LP_{cust} to the routes announced by customer ASes, LP_{peer} to the routes announced by peer ASes, and LP_{prov} to the routes announced by provider ASes, in such a way that $LP_{cust} > LP_{peer} > LP_{prov}$.*

Guideline B *Route reflectors modify the local preference value with LP_{mod} when receiving a route R from one of their clients, in such a way that*

- *if R is from a customer AS, $LP_{mod} > LP_{cust}$*
- *if R is from a peer AS, $LP_{cust} > LP_{mod} > LP_{peer}$*
- *if R is from a provider AS, $LP_{peer} > LP_{mod} > LP_{prov}$*

Figure 5.4 shows a simple implementation of our guidelines. First, the `community` attribute is used to tag routes according to our requirements. Then, the `local-preference` attribute is modified according to the tags. Since a very similar technique is commonly used by ISPs to manage traffic from eBGP

Configuration for Border Routers

(i) Tag routes according to commercial relationships

```
if msg from customer
        add community comm_cust
if msg from peer
        add community comm_peer
if msg from provider
        add community comm_prov
```

(ii) Prefer customers to peers, and peers to providers

```
if comm_cust in community
        set local-pref 200
if comm_peer in community
        set local-pref 100
if comm_prov in community
        set local-pref 50
```

Configuration for Route Reflectors

(i) Tag routes announced by clients

```
del community comm_client
if msg from client
        add community comm_client
```

(ii) Prefer customers to peers, and peers to providers
Prefer clients to non-clients

```
if comm_cust in community
        set local-pref 200
if comm_cust and comm_client in community
        set local-pref 220
if comm_peer in community
        set local-pref 100
if comm_peer and comm_client in community
        set local-pref 120
if comm_prov in community
        set local-pref 50
if comm_prov and comm_client in community
        set local-pref 70
```

Figure 5.4: A simple configuration complying with Guidelines A and B.

neighboring ASes [CR05], we argue that our guidelines do not add significant configuration complexity.

We now prove that our guidelines guarantee iBGP stability.

Lemma 5.1 *If the configurations of all iBGP speakers of an AS comply with Guidelines A and B, then eventually either:* (i) *all iBGP speakers select routes learned from customer ASes,* (ii) *all iBGP speakers select routes learned from peer ASes, or* (iii) *all iBGP speakers select routes learned from provider ASes.*

Proof: Consider an AS in the steady state, and let W be the set of BGP routes to a given destination prefix that are selected as best by at least one iBGP speaker. Let C_1 be the set (*class*) of customer ASes, C_2 be the class of peer ASes, and C_3 be the class of provider ASes.

The statement is trivially true if $|W| = 1$ or if all routes in W are learned from neighboring ASes belonging to the same class. Then, assume by contradiction that there exist at least two routes r_1 and r_2 in W such that r_1 (r_2) is learned from a neighboring AS belonging to class C_i ($C_j \neq C_i$). Without loss of generality, let $i < j$. Since each iBGP speaker only propagates its best route, there must exist a border router u which selects r_1 and a border router v which selects r_2.

Let P be a valid signaling path between u and v (P must exist, see Section 5.2). Because of the iBGP propagation rules in Table 5.1, there must exist two speakers x and y in P such that x selects r_1, y selects r_2, and there is an iBGP peering between x and y. We have the following cases:

- x acts as a route reflector for y (or vice versa). Then, according to iBGP route propagation rules in Table 5.1, x eventually announces r_1 to y.
- x and y are peers. In this case, we have that x learned route r_1 either from an eBGP neighbor or from a client. In both cases, iBGP route propagation rules in Table 5.1 ensure that x eventually announces r_1 to y.

Hence, y is aware of r_1 in the steady state. Guidelines A and B imply that y eventually selects route r_1 because it has a higher `local-preference` than r_2 (a contradiction). □

Theorem 5.2 *If every router configuration complies with Guidelines A and B, then the resulting iBGP configuration is free from routing oscillations under arbitrary link failures.*

Proof: Consider the translation to an SPVP instance S computed as described in Section 5.5. By Lemma 5.1, we know that we can restrict our attention to routes announced by the same class of neighboring ASes. We now direct some of the edges in S and then show that the resulting instance satisfies the sufficient conditions for robustness (i.e., stability under arbitrary link failures) described in [GR00]. We refer to Section 3.3 for an accurate description of how these conditions relate to eBGP stability.

Take each edge in S and direct it from a client to its route reflector. Namely, if router u is a client of v, then we have edge (u, v). We say that v is a *parent* of u, and, similarly, u is a *child* of v. Under this convention, edges are oriented from a child to its parent. According to the conditions in [GR00], a partially oriented instance is free from routing oscillations if all the following conditions hold.

valley-free Permitted paths can be written as an "uphill" part, i.e., a (possibly empty) sequence of child-to-parent edges, optionally followed by a "step", i.e., an undirected edge, and terminated by a "downhill" part, i.e., a (possibly empty) sequence of parent-to-child edges.

prefer-child Each node prefers routes announced by its children to the routes announced by other neighbors.

no-directed-cycle There are no directed cycles in the graph.

The *valley-free* condition holds since the set of permitted paths $\mathcal{P}^u$ at each node u consists only of valid signaling paths, according to the translation to SPVP described in Section 5.5. Recall from Section 5.2 that any valid signaling path can be written as a (possibly empty) sequence of **up** edges, an optional **over** edge, and a (possibly empty) sequence of **down** edges, and that we oriented each edge from a client to its route reflector. The *prefer-child* condition is ensured by Guideline B. The *no-directed-cycle* condition follows from the fact that route reflectors are organized in a hierarchy (Section 5.2), hence the orientation we defined cannot result in a directed cycle.

The statement hence follows by Theorem 5.1 of [GR00]. □

Observe that Guidelines A and B act on the `local-preference` attribute. Since this attribute is evaluated at the first step in the decision process, the ISP's policy takes the highest precedence and the selected routes are guaranteed to be compliant with the policy no matter what the value of other BGP attributes. In particular, attributes like `AS-path` and `origin`, which can be manipulated by external ASes for their own traffic engineering purposes, are

only considered as tie breakers. As a side effect, the forwarding plane is no longer affected by changes to the `AS-path` or `origin` attribute, which makes BGP-induced traffic shifts across the network much less likely to occur.

5.7 Conclusions

BGP configuration languages offer the possibility to change iBGP attributes en route, but there is little understanding on the extent to which routing could be affected. This chapter discusses the potential benefits and drawbacks, and proposes a systematic way to mitigate the risks of this practice. We stress that our results should not be taken as an argument supporting (nor discouraging) modification of iBGP attributes.

We show a simple scenario where changing iBGP attributes yields better traffic engineering, however we also prove that changing iBGP attributes can result in creating routing oscillations that would not be possible otherwise.

By analyzing BGP update traces collected at multiple vantage points, we estimate that at least $1,800$ ASes in the Internet exhibit a set of selected routes which cannot be explained if iBGP attributes are left untouched.

Since neither known theoretical models [GW02a] nor practical techniques for oscillation detection [FRBS08] allow iBGP attributes to be changed, we define a way to translate an iBGP configuration to an instance of a well-known model for policy-based path vector protocols like BGP. We use this translation to formally prove stability properties.

Finally, we propose configuration guidelines to change iBGP attributes in a profitable way. Compliance to our guidelines guarantees stability under faulty conditions and enforces reasonable traffic engineering policies, not depending on BGP attributes that could be modified by other ASes.

A natural question that arises is how hard it is to translate complex traffic engineering requirements into BGP configurations with iBGP attribute changing. This chapter gives a preliminary answer for the case where policies follow the customer-provider pattern. However, this is far from a complete methodology tackling this issue.

Part III

Detecting BGP Instabilities

Chapter 6

Finding Potential Instabilities by Static Analysis*

6.1 Introduction and Related Work

Despite the literature provides many important theoretical contributions on the stability of policy-based path vector protocols, in practice network operators have a limited set of weapons to fight BGP oscillations. Since theoretical problems about stability seem to have untractable complexity [GSW02, FP08], network administrators must rely on configuration guidelines that prevent routing oscillations. A number of such guidelines have been proposed to tackle both eBGP [GR00, GGR01] and iBGP [GW02a, GW02b, RS06] routing instabilities. Unfortunately, while guidelines can be useful in building from scratch a network which is oscillation-free by design, they do not help an operator in

*Part of the material presented in this chapter is based on the following publications

- L. Cittadini, G. Di Battista, S. Vissicchio. Doing Don'ts: Modifying BGP Attributes within an Autonomous System. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*, IEEE, 2010.
- L. Cittadini, M. Rimondini, M. Corea, G. Di Battista. On the Feasibility of Static Analysis for BGP Convergence. In *Proc. International Symposium on Integrated Network Management (IM 2009)*, IEEE, 2009.
- L. Cittadini, G. Di Battista, M. Rimondini. How Stable is Stable in Interdomain Routing: Efficiently Detectable Oscillation-Free Configurations. Technical Report RT-DIA-132-2008, Dept. of Computer Science and Automation, Roma Tre University, 2008.

deciding whether his running configuration is stable or not.

On the other hand, modifications have been proposed to both eBGP [GW00, ERC+07] and iBGP [CGM03, KCM04, MC04b, MC04a, FR09] that ensure convergence of the protocol to a stable routing choice for each router. However, given the deployment constraints, none of these modified versions of BGP has ever seen substantial deployment.

In this chapter, we tackle the problem of BGP stability from the perspective of a network operator that wants to know whether an already deployed BGP configuration is stable or not. First, we describe a heuristic algorithm that statically detects potential oscillations in an SPVP instance. SPVP is an abstract model for BGP (see Chapter 2 for a detailed description of SPVP). We prove that our algorithm has several highly desirable properties: *(i)* it exceeds state of the art algorithms in that it is able to correctly report more configurations as stable, *(ii)* it can be implemented efficiently enough to enable static analysis of Internet scale eBGP configurations, *(iii)* it is free from false negatives, meaning that configurations are only reported as stable if there are no potential oscillations, and *(iv)* it can help in spotting the troublesome points in a detected oscillation.

Then, we describe a technique to translate BGP configurations to SPVP instances. We show smart optimizations that allow us to efficiently translate large iBGP configurations into SPVP instances.

We also describe and evaluate the architecture of a modular tool that exploits our algorithms to process native router configurations and return information about the potential presence of oscillations. Our approach is complementary to other existing BGP configuration checkers such as [QN04, FB05] and other existing policy checkers in that we explicitly focus on convergence, which requires analyzing configuration semantics rather than executing syntactic checks and batch tests. We also overleap simulators [QU05], in that we are able to point out the converging portion of networks that could permanently oscillate. For these reasons, we believe such a tool can effectively integrate existing checkers to further assist operators in verifying configurations. Our approach is also more general than [FRBS08] because we support arbitrarily complex route reflection hierarchies, as well as iBGP attributes that are changed en route (see Chapter 5). We validate the architecture using a prototype implementation and show that both the translation of policies to SPVP and the convergence check algorithm itself can be implemented efficiently enough to statically analyze BGP configurations in practice.

The rest of the chapter is organized as follows. Our algorithm to check for convergence is presented in Section 6.2, together with a formal proof of

correctness. However, since our algorithm works on an abstract model of BGP (see Chapter 2), we also need to translate a real network to an instance of such a model to apply the algorithm in practice. While the translation takes exponential time in the worst case, our experimental results show that smart optimizations allow us to handle both eBGP (Section 6.3) and iBGP (Section 6.4) networks with a reasonable amount of resources. We then conclude in Section 6.5

6.2 A Greedy Algorithm for SPVP Instances

In this section we first briefly recall a greedy algorithm (we call it GREEDY) that has been proposed in [GSW02] to find a stable path assignment (see Section 2.3) in an SPVP instance. Second, we propose a new greedy algorithm, called GREEDY$^+$. Finally, we compare GREEDY and GREEDY$^+$.

Algorithm GREEDY attempts to grow a solution by iteratively building a stable path assignment. If the algorithm terminates successfully, the output is the only stable path assignment admitted by the input SPVP instance. Otherwise, the greedy algorithm is only able to identify a stable path assignment for a subset of the vertices.

The algorithm maintains a *stable set* of vertices for which convergence is guaranteed. The stable set at iteration i of the algorithm is denoted by V_i. Vertex 0 is always in the stable set, therefore we set $V_0 = \{0\}$. As the stable set grows, a path assignment π defined on the vertices in V_i is iteratively built.

We say that a path P is *compatible with a path assignment* π if $P = P'(u\ v)\pi(v)$, where P' does not contain vertices in V_i, $(u, v) \in E$, and $v \in V_i$.

Algorithm GREEDY is as follows. At iteration i, let P_v be the path with minimum $\lambda^v(P)$ among the paths at v compatible with π. If such a path does not exist, let $P_v = \epsilon$. If there exists a vertex $v \notin V_{i-1}$ such that P_v has a next hop in V_{i-1}, then construct V_i by adding v to V_{i-1} and set $\pi(v) = P_v$. If such a vertex v does not exist, then stop.

Intuitively, at each iteration, vertex v is stabilized because its best compatible path directly reaches an already stabilized vertex. Observe that the algorithm terminates after at most $|V|$ iterations. A solution to the SPVP instance exists if, after k iterations, GREEDY ends with $V_k = V$. The *solution* is given by the stable path assignment π.

Note that the description of GREEDY we propose here slightly differs from the one in [GSW02], in that we require that only a single vertex enters the stable set at each iteration. We will explain in the following that this modified version

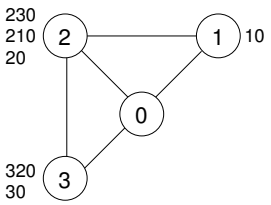


Figure 6.1: DI-SAFE-GREE: An SPVP instance for which algorithm GREEDY fails to find a solution.

is indeed equivalent to the original algorithm. We choose to describe GREEDY with this slight modification in order to better introduce the improvements that allow us to overcome some shortcomings of the original algorithm.

GREEDY can fail to find a solution even if the SPVP instance has guaranteed convergence. Consider, for example, the instance DI-SAFE-GREE in Fig. 6.1. It can be easily verified that any fair activation sequence of SPVP on this instance is finite. In fact, any fair activation sequence is such that vertices 1, 2, and 3 learn about the direct path to 0. After that, pair $(1, 2)$ is eventually activated, and 2 learns about (2 1 0). Henceforth, vertex 2 will permanently be unable to select (2 0), in turn preventing vertex 3 from choosing (3 2 0). Finally, after pair $(3, 2)$ is activated, 2 switches to its best path (2 3 0) and SPVP terminates, as no other message is further generated. Therefore any fair activation sequence is forcedly finite, and this implies that DI-SAFE-GREE is safe.

We will now walk through the execution of GREEDY on DI-SAFE-GREE. At the first iteration, vertex 1 enters the stable set V_1, and $\pi(1) = (1\ 0)$. At the second iteration, the algorithm forcedly stops. In fact, path (2 3 0) is compatible with π because $2, 3 \notin V_1$, $0 \in V_1$, and $(3\ 0) \in E$. However, even if (2 3 0) is the best compatible path at vertex 2, its next hop is not in V_1. A similar argument applies to path (3 2 0). Therefore, no new vertex can be added to the stable set and the algorithm stops without finding a solution, since $V_1 \neq V$.

We now describe a variant of this algorithm, which we call GREEDY$^+$. This variant is able to solve DI-SAFE-GREE.

We say that a path P belonging to a set $\mathcal{S}$ of paths is *consistent with* $\mathcal{S}$ if either $P = \epsilon$, $P = (0)$, or $P = (v\ u)P'$ where $(v, u) \in E$ and P' is consistent with $\mathcal{S}$. For example, let $\mathcal{S} = \{(0), (1\ 0), (2\ 1\ 3\ 0)\}$: it is easy to check that (0)

and (1 0) are consistent with $\mathcal{S}$, while (2 1 3 0) is not. Further, for each vertex v we define a set $\bar{\mathcal{P}}^v$ of paths called *useful set*. The useful set $\bar{\mathcal{P}}^v$ is initialized with the paths in $\mathcal{P}^v$ that are consistent with $\mathcal{P}$. Let $\bar{\mathcal{P}} = \bigcup_{v \in V} \bar{\mathcal{P}}^v$.

GREEDY$^+$ differs from GREEDY in that it exploits the useful set in order to prune paths that, starting from a certain iteration, become permanently unavailable. Hence, GREEDY$^+$ needs to keep the useful set up to date at each iteration.

What follows is a description of GREEDY$^+$. Let $V_0 = \{0\}$. At iteration i, GREEDY$^+$ performs the following steps:

(i) Exploit the current stable set in order to prune all those paths that cannot be selected because of the presence of a better ranked path offered by a neighbor in the stable set. For each vertex $v \in V - V_{i-1}$ such that v has a neighbor $u \in V_{i-1}$ and there exists a path $P = (v\ u)P'$ such that $\{P'\} = \bar{\mathcal{P}}^u$, remove from $\bar{\mathcal{P}}^v$ all the paths Q such that $\lambda^v(Q) > \lambda^v(P)$. Intuitively, this step is performed because P will be always available at v.

(ii) Enforce consistency on all the paths. For each vertex $v \notin V_{i-1}$, remove from $\bar{\mathcal{P}}^v$ all the paths that are not consistent with $\bar{\mathcal{P}}$.

(iii) Grow the stable set, or stop. Let $C_i \subset V - V_{i-1}$ be the set of *candidate* vertices v such that the path $P \in \bar{\mathcal{P}}^v$ with minimum $\lambda^v(P)$ either has a next hop in V_{i-1}, or $P = \epsilon$. If $C_i = \oslash$, then set $V_i = V_{i-1}$ and stop. Otherwise, if $C_i \neq \oslash$, then pick a vertex $u \in C_i$, construct V_i by adding u to V_{i-1}, and set $\bar{\mathcal{P}}^u = \{P\}$.

If GREEDY$^+$ stops after k iterations, its *output* consists of a stable set V_k and sets $\bar{\mathcal{P}}^v\ \forall v \in V$, with $|\bar{\mathcal{P}}^v| = 1\ \forall v \in V_k$. If $V_k = V$, GREEDY$^+$ computes a stable path assignment π for the input instance such that $\bar{\mathcal{P}}^v = \{\pi(v)\}\ \forall v \in V$.

An example of a successful execution of GREEDY$^+$ on DI-SAFE-GREE is shown in Table 6.1. Note that at iteration 1 path (2 0) is evicted from $\bar{\mathcal{P}}^2$ because (2 1 0) is preferred and permanently available (Step *(i)*). This action puts in evidence the difference between GREEDY$^+$ and GREEDY: as we have seen, GREEDY forcedly stops at iteration 1. Step *(ii)* then removes (3 2 0) from $\bar{\mathcal{P}}^3$ since it is inconsistent with $\bar{\mathcal{P}}$. This allows vertex 3 to enter the stable set.

Theorem 6.1 *Let n be the size of an* SPVP *instance S.* GREEDY$^+$ *can be implemented to terminate on S in time that is polynomial in n.*

i	V_i	C_i	$\bar{\mathcal{P}}^1$	$\bar{\mathcal{P}}^2$	$\bar{\mathcal{P}}^3$
0	$\{0\}$	$\{1\}$	(1 0)	(2 3 0) (2 1 0) (2 0)	(3 2 0) (3 0)
1	$\{0,1\}$	$\{3\}$	(1 0)	(2 3 0) (2 1 0)	(3 0)
2	$\{0,1,3\}$	$\{2\}$	(1 0)	(2 3 0)	(3 0)
3	V	$\oslash$	(1 0)	(2 3 0)	(3 0)

Table 6.1: A successful execution of GREEDY$^+$ on DI-SAFE-GREE (Fig. 6.1). The table shows sets V_i, C_i, $\bar{\mathcal{P}}^v$ at iteration i of GREEDY$^+$.

Proof: A trivial bound follows.

Step *(i)* of GREEDY$^+$ applies to those vertices v which extend a path P offered by some neighbor u in the stable set. This step can be implemented by evaluating λ^v for all the paths in each $\bar{\mathcal{P}}^v$ and comparing its value with $\lambda^v((v\ u)P)$. This takes $O(n^3)$ time, since the length of a path is $O(n)$.

Step *(ii)* of GREEDY$^+$ enforces consistency. This can be accomplished by comparing each path in $\bar{\mathcal{P}}$ with all the others, which takes $O(n^3)$.

Finally, at Step *(iii)* of GREEDY$^+$ candidate vertices can be found in $O(n^3)$ time.

Since GREEDY$^+$ executes at most $|V|$ iterations and an instance of SPVP can have $O(n)$ vertices, GREEDY$^+$ can be implemented to run in $O(n^4)$. □

The following properties and Lemma 6.1 show that GREEDY$^+$ is deterministic in the sense that, at any time where multiple choices are possible, performing any of them does not alter the output.

Property 6.1 *If* GREEDY$^+$ *terminates after k iterations, its output is completely defined by sets V_k and $\bar{\mathcal{P}}^v$ $\forall v \in V_k$.*

Proof: The missing portion of the output, $\bar{\mathcal{P}}^v$ $\forall v \in V - V_k$, can be uniquely constructed starting from V_k and $\bar{\mathcal{P}}^v$ $\forall v \in V_k$. Consider a new SPVP instance $S' = (G', \mathcal{P}', \Lambda')$ with $G' = G$, $\Lambda' = \Lambda$, and, for any $v \in V$:

$$\mathcal{P}'^v = \begin{cases} \bar{\mathcal{P}}^v & \text{if } v \in V_k \\ \mathcal{P}^v & \text{if } v \notin V_k \end{cases} .$$

Now, initialize the stable set V_0 to V_k and execute Steps *(i)* and *(ii)* of GREEDY$^+$ on S'. We now show that, after doing so, $\bar{\mathcal{P}}'^v = \bar{\mathcal{P}}^v$, $\forall v \in V$. This is trivially

true for vertices $u \in V_k$, as no path is ever removed from $\bar{\mathcal{P}}'^u$. Observe that the outcome of Step *(i)* of GREEDY$^+$ only depends on the topology of the graph G', the ranking functions Λ', and the sets of useful paths $\bar{\mathcal{P}}'^v$, with $v \in V_k$. Because of the way S' has been defined, we know that, at Step *(i)*, a path is removed from $\bar{\mathcal{P}}^v$ iff it is removed from $\bar{\mathcal{P}}'^v$. Hence, any possible difference must be due to Step *(ii)*.

We prove by contradiction that the output coincides also for vertices in $V - V_k$. Suppose that this is not the case, i.e., there exists some vertex $v \in V - V_k$ such that $\bar{\mathcal{P}}'^v \neq \bar{\mathcal{P}}^v$. Then, there exists a path P such that either $P \notin \bar{\mathcal{P}}'^v \wedge P \in \bar{\mathcal{P}}^v$ or $P \in \bar{\mathcal{P}}'^v \wedge P \notin \bar{\mathcal{P}}^v$. In the first case, the execution of Step *(ii)* on S' has removed from $\bar{\mathcal{P}}'^v$ a path that the execution of GREEDY$^+$ on S regarded as consistent. But this is impossible, since $\forall v \in V$, $\bar{\mathcal{P}}^v \subseteq \mathcal{P}'^v$, so there can be no path that is consistent with $\bar{\mathcal{P}}$ and is not consistent with $\mathcal{P}'$. In the second case, the execution on S has removed from $\bar{\mathcal{P}}^v$ a path P that the execution on S' considered as consistent. Since it cannot be $P \notin \mathcal{P}^v$, then for P to be inconsistent with $\bar{\mathcal{P}}$, it may only be the case that $P = (v \ \ldots \ u)P_u$, where $P_u \notin \bar{\mathcal{P}}^u$ and $P_u \in \bar{\mathcal{P}}'^u$. In turn, this is only possible if there exists a path P_w such that $P_u = (u \ \ldots \ w)P_w$, with $P_w \notin \bar{\mathcal{P}}^w$ and $P_w \in \bar{\mathcal{P}}'^w$. By proceeding this way, we must eventually end up on a vertex x in V_k, possibly 0. By recalling that $\bar{\mathcal{P}}'^v = \bar{\mathcal{P}}^v \ \forall v \in V_k$ by construction, we have a contradiction in that it should be $P_x \notin \bar{\mathcal{P}}^x$ and $P_x \in \bar{\mathcal{P}}'^x$. □

Property 6.2 *Consider a path P that is inconsistent with $\bar{\mathcal{P}}$ at iteration i of* GREEDY$^+$*. Then, P is inconsistent at any iteration $j > i$.*

Proof: The property follows by observing that GREEDY$^+$ never adds new paths to $\bar{\mathcal{P}}$. □

Property 6.3 *At any iteration i of* GREEDY$^+$*, $C_i \cap V_i = V_i - V_{i-1}$.*

Proof: By construction, $C_i \cap V_{i-1} = \oslash$. Now, at iteration i a vertex is picked from C_i and added to V_{i-1} to construct V_i. Therefore, the property follows. □

The following property states the fact that, once a vertex enters the candidate set, it stays there until it is eventually moved to the stable set.

Property 6.4 *Consider an arbitrary iteration i of* GREEDY$^+$ *and a vertex $v \in C_i$. Then there exists an iteration $j > i$ such that $v \in C_h$ for all $i \leq h \leq j$ and $v \in V_k$ for all $k \geq j$.*

Proof: Let $v \in C_i$ be a vertex such that the path $P \in \bar{\mathcal{P}}^v$ with minimum $\lambda^v(P)$ at iteration i either has a next hop in V_{i-1}, or $P = \epsilon$. Since no better path can enter $\bar{\mathcal{P}}^v$ during the execution of GREEDY$^+$ (Property 6.2) and P has the minimum value of λ^v among the paths in $\bar{\mathcal{P}}^v$ that are consistent with $\bar{\mathcal{P}}$, P can never be removed from $\bar{\mathcal{P}}^v$ at Step *(i)* of GREEDY$^+$. Moreover, if $P = \epsilon$, by definition P is a consistent path. Otherwise, if $P = (v\ u)Q$, $u \in V_{i-1}$, $\{Q\} = \bar{\mathcal{P}}^u$, then P will remain consistent with $\bar{\mathcal{P}}$ because its next hop is $u \in V_{i-1}$, so $\bar{\mathcal{P}}^u$ will not be updated after iteration i. Thus, P cannot be removed from $\bar{\mathcal{P}}^v$ at Step *(ii)*. Overall, starting from iteration i, path P will always be available in $\bar{\mathcal{P}}^v$ and will always have the minimum value of λ^v. In other words, v satisfies the conditions of Step *(iii)* at any iteration $k \geq i$, i.e., $v \in C_k \cup V_k$.

Since $\forall k \geq i\ v \in C_k \cup V_k$, and GREEDY$^+$ only terminates when the candidate set is empty, by Property 6.3 there must be an iteration j at which v is picked from C_j and added to V_{j-1} to construct V_j. The statement follows by recalling that vertices are never removed from the stable set. □

We now show that, if multiple candidates exist at Step *(iii)*, the output of GREEDY$^+$ is not affected by the vertex that actually enters the stable set.

Lemma 6.1 *Consider an arbitrary iteration j of* GREEDY$^+$ *and a set C_j of vertices satisfying the criteria of Step* (iii) *at iteration j. The output of* GREEDY$^+$ *does not change, regardless of the choice of vertex $v \in C_j$ performed at iteration j.*

Proof: Assume that GREEDY$^+$ terminates at iteration k. First of all consider that, by Property 6.1, it is sufficient to prove the assertion for sets V_k and $\bar{\mathcal{P}}^v$ with $v \in V_k$. Consider an arbitrary vertex $u \in C_j$. By Property 6.4, we know that $u \in C_h$ for any iteration $h \geq j$, until u eventually enters the stable set. Also, as shown in the proof of Property 6.4, the best path $(u\ v)P$, $v \in V_h$ is always in $\bar{\mathcal{P}}^u$. Therefore, regardless of the iteration at which u is actually selected, the set $\bar{\mathcal{P}}^u$ is always updated with path $(u\ v)P$. Moreover, the set of paths that become inconsistent with $\bar{\mathcal{P}}$ after setting $\bar{\mathcal{P}}^u = \{(u\ v)P\}$ does not depend on the iteration either.

Thus, a vertex $u \in C_h$ can be picked by Step *(iii)* at any iteration h of GREEDY$^+$ without affecting neither V_k nor $\bar{\mathcal{P}}^v\ \forall v \in V_k$. Since this is true for any vertex $u \in C_h$, GREEDY$^+$ can select an arbitrary candidate vertex at each iteration h without affecting the output. □

Note that algorithm GREEDY$^+$ essentially differs from GREEDY because of the presence of Step *(i)*. In fact, if we skip Step *(i)*, at each iteration i both the

algorithms select the best path among the consistent ones having a next hop in V_i. This can be easily verified by observing that, when Step *(i)* is removed, the set $\bar{\mathcal{P}}$ is only used to filter out inconsistent paths.

Therefore, it is easy to check that the validity of Lemma 6.1 can be extended to GREEDY by considering the path assignment π as its output and skipping any consideration about Step *(i)* in the proof of the lemma. This further confirms that the description of GREEDY given in this section and the original description given in [GSW02] are indeed equivalent.

We now show that GREEDY$^+$ is more powerful than GREEDY in that it is able to compute a guaranteed stable state for a strictly larger set of SPVP instances.

Lemma 6.2 *Let S be an* SPVP *instance. If* GREEDY *terminates on S finding a path assignment π^*, then* GREEDY$^+$ *also terminates on S finding π^*.*

Proof: By Lemma 6.1 we know that, when multiple vertices can enter the stable set at a given iteration, the solution computed by GREEDY$^+$ is independent on the order in which these vertices are considered. Therefore, we prove the assertion by showing that GREEDY$^+$ can find π^* by selecting vertices to put in the stable set in the very same order as GREEDY does. We show it by mapping each iteration of GREEDY to one iteration of GREEDY$^+$. In the following, we will refer to GREEDY's stable set as V_j, and to GREEDY$^+$'s stable set as V_j^+, and we will indicate with π the path assignment defined by GREEDY at a given iteration. The proof proceeds by induction on the iteration j. It is trivially true that, at $j = 0$, $V_j = V_j^+ = \{0\}$. Assume that $V_{j-1} = V_{j-1}^+$ and, without loss of generality, that the stable sets have been constructed by adding vertices in the very same order by the two algorithms. Consider vertex u that GREEDY selects at iteration j. This implies that $(u\ v)\pi(v)$ is the path with minimum λ^u among those compatible with π, for some $v \in V_{j-1}$. By the induction hypothesis, $\bar{\mathcal{P}}^v = \{\pi(v)\}$, therefore path $(u\ v)\pi(v)$ is consistent with $\bar{\mathcal{P}}$. We show that path $(u\ v)\pi(v)$ must still be in $\bar{\mathcal{P}}^u$ at iteration j. Property 6.2 ensures that Step *(ii)* did not remove path $(u\ v)\pi(v)$ from $\bar{\mathcal{P}}^u$. That is, since path $(u\ v)\pi(v)$ is consistent with $\bar{\mathcal{P}}$ at iteration j, it was always consistent during the previous iterations.

By the induction hypothesis, $\forall w \in V_{j-1}\ \bar{\mathcal{P}}^w = \{\pi(w)\}$, therefore all the paths that are regarded as consistent by GREEDY$^+$ are necessarily compatible with π. Hence, since $(u\ v)\pi(v)$ is a consistent with $\bar{\mathcal{P}}$ and has minimum λ^u among the paths compatible with π, it must also have minimum λ^u among the paths consistent with $\bar{\mathcal{P}}$. Therefore path $(u\ v)\pi(v)$ cannot be deleted at

Step *(i)* of GREEDY$^+$. Thus, vertex u is a candidate to be inserted in the stable set by GREEDY$^+$.

Since, by Lemma 6.1, the output of GREEDY$^+$ is unaffected by the order in which vertices enter the stable set, we can assume without loss of generality that GREEDY$^+$ too selects vertex u at iteration j. This in turn implies that GREEDY$^+$ finds the same path assignment π^*. □

Theorem 6.2 *The set of instances that* GREEDY$^+$ *can successfully solve is strictly larger than the set of instances that* GREEDY *is able to solve.*

Proof: Lemma 6.2 proves the inclusion. The strictness is supported by DISAFE-GREE, since, as we discussed above, GREEDY is not able to find a stable path assignment, while GREEDY$^+$ is (see Table 6.1). □

We now formally prove that, if GREEDY$^+$ terminates successfully on an SPVP instance S, then S is safe.

Theorem 6.3 *Consider an* SPVP *instance S and run* GREEDY$^+$ *on S. Let $P \in \mathcal{P}^v$ be a path that* GREEDY$^+$ *deletes at iteration j. Then, for any fair activation sequence σ of* SPVP *on S, there exists a time t' such that $\forall t > t'$, $\pi_t(v) \neq P$.*

Proof: The statement asserts that GREEDY$^+$ deletes only those paths that will be discarded by any fair activation sequence of SPVP. The proof is by induction on the iteration j of GREEDY$^+$. At iteration $j = 1$, since $\bar{\mathcal{P}}^u = \mathcal{P}^u$ for all $u \in V$, GREEDY$^+$ deletes a path P from $\bar{\mathcal{P}}^v$ at either Step *(i)* or Step *(ii)* according to the following conditions.

Deletion at Step (i): Since $V_0 = \{0\}$, the deletion takes place if $\lambda^v((v\ 0)) < \lambda^v(P)$. By the fairness of σ, there must exist a time t' such that $(v, 0)$ is activated at t': this prevents v from selecting P after t'.

Deletion at Step (ii): It takes place if P is inconsistent with $\mathcal{P}$, i.e., $P = Q(w)R$ and $R \notin \mathcal{P}^w$. In this case, the statement trivially follows since $\pi_t(w) \neq R\ \forall t$.

Assume, by induction, that the assertion holds for a given iteration $j - 1$ of GREEDY$^+$. We now prove that the same property is true for the paths that are deleted during iteration j. Again, during iteration j, GREEDY$^+$ deletes a path P from $\bar{\mathcal{P}}^v$ at either Step *(i)* or Step *(ii)*.

Deletion at Step (i): It takes place if there exists $u \in V_{j-1}$ such that $(v, u) \in E$ and $\lambda^v((v\ u)P') < \lambda^v(P)$, where $\{P'\} = \bar{\mathcal{P}}^u$. Observe that the induction hypothesis assures that previously deleted paths are eventually discarded after

time t'. Then, by the fairness of σ, there must exist a time $t'' > t'$ such that (u, v) is activated at t'' and $(v\ u)P'$ is made available at $v\ \forall t > t''$. This prevents v from selecting path P, i.e., $\pi_t(v) \neq P\ \forall t > t''$.

Deletion at Step (ii): It takes place if P is inconsistent, i.e., $P = Q(w)R$ and $R \notin \bar{\mathcal{P}}^w$. By the induction hypothesis, there exists t' such that $\forall t > t'$ $\pi_t(w) \neq R$. Then, by the fairness of σ, v must receive a message that withdraws the availability of R at a time $t'' > t'$. Therefore, $\pi_t(v) \neq P\ \forall t > t''$. □

Corollary 6.1 *If* GREEDY$^+$ *terminates successfully after k iterations on an* SPVP *instance S, then S is safe and has π_k as its unique stable path assignment.*

Observe that Corollary 6.1 actually states that GREEDY$^+$ can be used as a static, centralized, and deterministic algorithm to efficiently emulate the behavior of the SPVP protocol in the long term, thus dealing with the nondeterminism that SPVP features. In our opinion, this property can be effectively exploited, e.g., by a network administrator that wants to analyze how BGP will behave in his/her own network.

We would like to stress the fact that GREEDY$^+$ is able to verify the safety even of instances for which known sufficient conditions for safety do not hold (e.g., BAD BACKUP, shown in Figure 3.2, which contains a dispute reel but is safe).

Using GREEDY$^+$ also brings another important advantage to network operators. The following theorem shows that, when GREEDY$^+$ terminates with $V_k \neq V$, it actually pinpoints a dispute reel in the network. Intuitively, this means that, even when GREEDY$^+$ is unable to prove the safety of a given SPVP instance, it can help pinpoint the trouble points of the network.

Theorem 6.4 *Consider an* SPVP *instance S and run* GREEDY$^+$ *on S. If, after k iterations,* GREEDY$^+$ *terminates with $V_k \neq V$, then S contains a dispute reel.*

Proof: Let u_0 be any node in $V - V_k$ such that $(u_0\ v_0)\pi_k(v_0) \in \bar{\mathcal{P}}^u$, and let $Q_0 = (u_0\ v_0)\pi_k(v_0)$. Since GREEDY$^+$ prunes out inconsistent paths, u_0 must exist. Also, since $u_0 \notin V_k$, there must be a path $P_0 \in \bar{\mathcal{P}}^u$ which has higher rank than Q_0, that is, $\lambda^u(P_0) < \lambda^u(Q_0)$. Since P_0 must be consistent with V_k, it has the form $P_0 = R_0(u_1\ v_1)\pi_k(v_1)$, where R_0 is a path from u_0 to u_1 in $V - V_k$, $v_1 \in V_k$, and $(u_1, v_1) \in E$. Let $Q_1 = (u_1\ v_1)\pi_k(v_1)$. Note that we can repeat the same argument with u_1. If we continue in this manner we will eventually form a dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$.

Π clearly satisfies Conditions *(iii)* and *(ii)* of Definition 4.1, since each spoke path Q_i has the form $Q_i = (u_i\ v_i)\pi_k(v_i)$ and, by the way GREEDY$^+$ operates, each vertex in V_k only has a single consistent path, so that spoke paths form a tree and no intersections with other paths are possible. For the same reason, for all $u_i \in \vec{\mathcal{U}}$, u_i cannot appear in Q_j if $j \neq i$. Moreover, since path Q_i is always available at u_i, any path having a lower rank than Q_i has been pruned out by the deletion step of GREEDY$^+$, which means that, if $u_i \in R_j$, then $R_j[u_i]Q_{j+1}$ is preferred to Q_i. This allows us to conclude that Π satisfies all the conditions of Lemma 4.5, hence, by Lemma 4.6, S contains a dispute reel. □

6.3 From eBGP Networks to SPVP Instances

As we have seen in the previous section, GREEDY$^+$ exhibits interesting properties, most notably it is efficient, free from false positives, and provides partial results even when it cannot determine the stability of the input SPVP instance. However, before being able to analyze a real BGP network, we must first model it as an SPVP instance. We perform this in three separate steps:

topology we translate the network topology to a graph G,

dissemination we enumerate all possible BGP paths from the origin AS to every other AS, building a set of paths $\mathcal{P}^u$ for each vertex u, and

ranking we run the standard BGP decision process at every vertex u and extract the ranking λ^u.

Observe that, while the first step is trivial, the remaining two steps separate the generation of routing paths from the actual best route selection operated by BGP, similarly to [FB05].

For the *dissemination* phase, we essentially simulate the behavior of BGP as if it had no best path selection. Namely, we represent a BGP *announcement* with a pair $(P, \mathcal{A})$, where P is a path and $\mathcal{A}$ is a set of BGP attributes. Before a received announcement is processed by a router u, an *import filter* $F_{u \Leftarrow v}((P, \mathcal{A}))$ is applied to the announcement; similarly, before a router sends an announcement, an *export filter* $F_{u \Rightarrow v}((P, \mathcal{A}))$ is applied. The specification of a filter contains a predicate and a sequence of actions. The predicate is a boolean condition which can match BGP announcements based on the path and the other attributes they carry. If the predicate evaluates to true, the actions

process dissemination(v)

1: **while receive** $(P, \mathcal{A})$ **from** w **do**
2: $(P', \mathcal{A}') = F_{v \Leftarrow w}((P, \mathcal{A}))$
3: **if** $(P', \mathcal{A}') \neq (\epsilon, \oslash)$ **then**
4: rib-in$_t(v \Leftarrow w)$:= rib-in$_{t-1}(v \Leftarrow w) \cup \{(P', \mathcal{A}')\}$
5: **if** rib-in$_t(v \Leftarrow w) \neq$ rib-in$_{t-1}(u \Leftarrow w)$ **then**
6: **for all** $u \mid (u, v) \in E$ **do**
7: **send** $F_{v \Rightarrow u}(((v)P', \mathcal{A}'))$ **to** u
8: **end for**
9: **end if**
10: **end if**
11: **end while**

Figure 6.2: Modified version of SPVP used to disseminate routing paths.

are undertaken. Possible actions include further propagating the announcement or dropping it, as well as altering, adding, or dropping the attributes carried by the announcement itself. The application of a filter returns a BGP announcement with the pertinent attribute modifications applied, or $(\epsilon, \oslash)$ if the BGP announcement is discarded.

We run on G the modified version of SPVP described in Fig. 6.2. In this algorithm, vertex 0 first starts announcing $((0), \oslash)$. Vertices of G exchange routing messages containing the full set of attributes (including, e.g., `as_path`, `next_hop`, `community`, etc.) and apply all the configured filters, but no decision process is performed on the received messages. Instead, every time a new, not previously observed announcement is received by a vertex v, it is propagated over to v's neighbors. The purpose of this step is to enumerate all possible paths that comply with the import and export filters. We recall that an explicit representation of the paths is required by the SPVP model. It is easy to verify that the algorithm in Fig. 6.2 terminates, as it is not affected by the stability problems of SPVP. A set of permitted paths $\mathcal{P}^v$ for each $v \in V$ can then be constructed starting from the rib-in sets.

To perform the *ranking* phase, for each $v \in V$ we apply the BGP decision process to the announcements that v has collected in its rib-in during the dissemination, and we define the ranking functions λ^v.

Optimizing for Scalability

In principle, mapping vendor specific configurations to a set of explicitly permitted paths is a step that requires exponential time. On the other hand, hardcoding filter applications in the path generation process allows us to avoid generating a large number of paths.

We performed several dissemination experiments using as input the AS-level topologies from CAIDA [CAI]. While CAIDA datasets are unavoidably biased by the underlying inference algorithms by which they have been computed, we believe they are still a valuable data source of large scale policy labeled interdomain topologies, which is exactly what we need to verify the scalability of our approach. We extracted from the CAIDA dataset collected on Nov 19th, 2007 a set of smaller topologies by pruning vertices with degree lower than a threshold. We picked the thresholds in the following set of values (the corresponding number of vertices and edges is reported between parentheses): 1000 (7, 21), 500 (14, 70), 250 (33, 319), 100 (85, 1030), 50 (181, 1981), 35 (279, 2815), 25 (379, 3564), 10 (1150, 7887), 5 (2575, 13498), 4 (3591, 16776), 2 (16617, 44056), 1 (26540, 53979), the latter corresponding to the complete topology. All the generated graphs were connected. CAIDA datasets are annotated with information about the commercial relationships established between the ASes [Gao01]. In order to compare with state-of-the-art tools, we implemented these relationships with BGP policies using the same approach that is hardwired in the C-BGP simulator [QU05]. In order to understand how the location of the originating AS affects the size of the SPVP instance, i.e., the number of paths that need to be enumerated, in our experiments we assumed to originate a prefix from a given AS picked from a significant sample of ASes belonging to different tiers of the Internet customer-provider hierarchy. Namely, we chose the following set of originating ASes (the corresponding degree is reported between parentheses): AS 701 (2643), AS 7018 (2066), AS 8001 (230), AS 10026 (227), AS 74731 (100), AS 27064 (100), AS 6746 (73), AS 721 (50), AS 3741 (50).

Unfortunately, our experiments show that enumerating paths in a naive way is not feasible. Figure 6.3 shows the number of paths that had to be enumerated (y-axis) for the topologies we generated as described above (x-axis). In order not to get biased from the location of the originating AS, we plot the median number of generated paths across the ten ASes we picked as originators. Using a commodity PC with 4GB of RAM, we were only able to generate SPVP instances from very small topologies. Namely, the naive enumeration algorithm ran out of memory with the topology pruned at degree

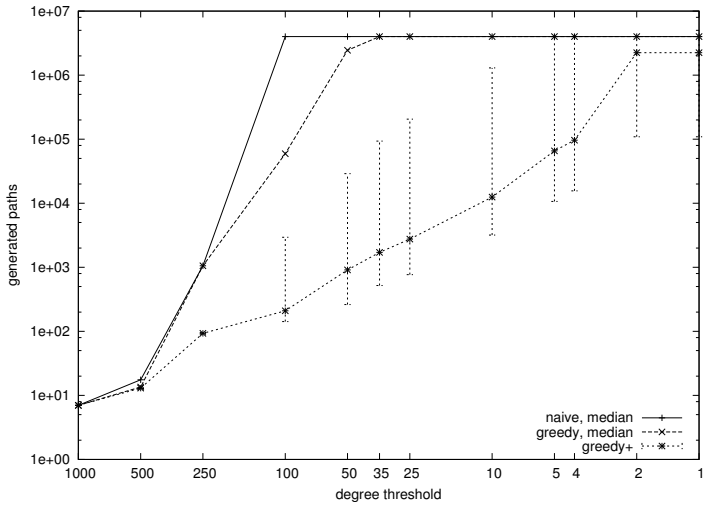


Figure 6.3: Median number of generated paths, computed on all the origin ASes. The plots show the values without optimizations (naive), with early stabilization (GREEDY) and with both early stabilization and early suppression (GREEDY$^+$). The latter plot includes the minimum and maximum values. The x-axis shows the degree threshold we used to prune CAIDA topologies.

100. In Figure 6.3, we associate an arbitrary value of 4M paths to each path generation task that went out of memory.

Clearly, we cannot enumerate paths in the naive way to translate an eBGP network to a SPVP instance. A possible way of reducing the number of paths that need to be generated is trying to run GREEDY during the dissemination phase process. The main idea is that, for each stable node that is found during the generation process, only a single path needs to be enumerated. To this end, vertex 0 marks the path announcements it sends as *reliable*. If a vertex v receives a reliable announcement $(P, \mathcal{A})$ from a neighbor u, v applies the import filter $F_{v \Leftarrow u}((P, \mathcal{A}))$ and compares the resulting $(P', \mathcal{A}')$ with the best announcement that it could ever receive from its neighbors. If, and only if, v considers $(P', \mathcal{A}')$ as most preferred, v applies the export filter $F_{v \Rightarrow w}((P', \mathcal{A}'))$, marks the announcement as reliable, and further propagates it to each neighbor $w \neq u$ (*early stabilization*). Observe that this step corresponds to precomputing a subset of the stable vertices computed by GREEDY. Based on Theorem 6.3,

a vertex v marking an announcement $(P, \mathcal{A})$ as reliable is guaranteed to select the corresponding path P. This allows us to only generate a single `as_path` for each stabilized vertex. In order to maximize the number of early stabilized vertices, we evaluate preferences based on the `local_pref` and on the `as_path` length.

Our experiments showed that early stabilization is not enough to make Internet scale configurations tractable: as reported in Figure 6.3, early stabilization provides only very limited benefits over the naive approach. We therefore apply an additional optimization step while generating the SPVP instance: vertex v does not propagate any announcement that it considers worse than a received reliable announcement $(P, \mathcal{A})$ (*early suppression*). In fact, since paths from reliable announcements are always available, v will be unable to select an alternative path ranked worse than $(P, \mathcal{A})$. This basically corresponds to performing some of the path deletions found in GREEDY$^+$ during the dissemination phase. As shown in Figure 6.3, this technique allows us to reduce the number of paths in the SPVP instance by one or two order of magnitude. In particular, starting from degree threshold 35, in most cases the SPVP instance can *only* be generated using the optimizations enabled by GREEDY$^+$.

Spotting Potential Oscillations

For each topology and originator AS that we were able to translate to an SPVP instance, we checked the stability using an implementation of the GREEDY$^+$ algorithm. Each convergence check took between a fraction of a second and 67 seconds to complete, on a dual Xeon 2.66 GHz platform. We finally looked at the percentage of vertices of the input topology that our algorithm reported as safe. According to the results in [GGR01], if the commercial relationships are configured using a the well-known customer-provider routing policies, convergence is always guaranteed. Interestingly, depending on the originator AS, our checker reported up to 15% of vertices as potentially unstable. We further investigated into this issue by separately running the C-BGP simulator on some of the affected topology-originator pairs, and found that C-BGP was actually unable to converge on them (we halted the simulation after 15 hours, while on average C-BGP terminates in a few seconds). We ascribe this behavior to the fact that CAIDA topologies include sibling relationships, which are not envisaged in the sufficient conditions for safety [GGR01]. To further confirm this, in a separate experiment our prototype was also able to spot a triple of vertices that generated a DISAGREE (see Figure 2.1c) structure. The triple actually involved a sibling relationship and prevented C-BGP from converging.

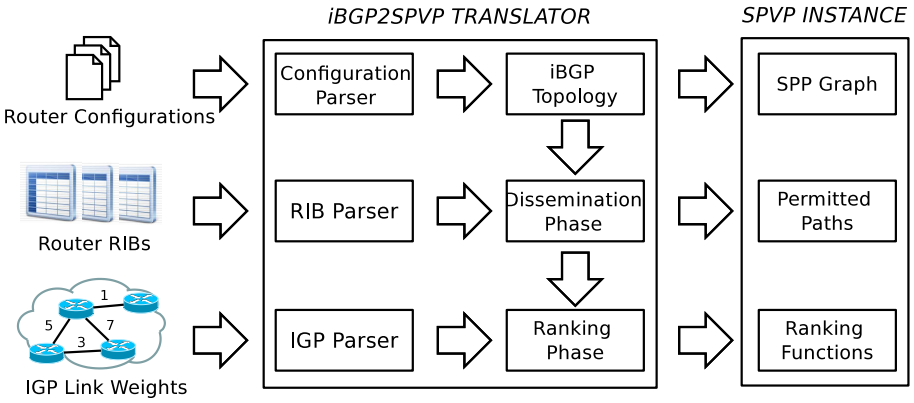


Figure 6.4: Architecture of the stability checker tool.

6.4 From iBGP Networks to SPVP Instances

Translating an iBGP network to an SPVP instance is, in principle, similar to the eBGP case. Namely, we first parse BGP configuration files to extract the iBGP peering *topology* and encode this topology in a graph G (see Section 5.2). Then we disseminate BGP routes in order to compute the set $\mathcal{P}^u$ of permitted paths at each node u. To do that, we need to know the eBGP routes injected by border routers and to enumerate all valid signaling paths. Hence, we first extract eBGP routes from the BGP Routing Information Base (RIB) of each border router. Second, we simulate the propagation of each route through G. Observe that, during the simulation, iBGP attributes of a route might be changed by traversed routers according to their BGP configuration. At the end of the *dissemination* phase, we end up with a set of BGP routes at each router u, which are used to compute the set of permitted paths $\mathcal{P}^u$. As a final step, we need to define the ranking function λ^u at each node u (*ranking* phase). To this end, we run the full BGP decision process at each node u, in order to obtain a sorted list of the BGP routes that were collected during the dissemination phase. The corresponding ranking is used to define function λ^u. Notice that, as the BGP decision process uses the IGP metric as a tie breaker, we need to know the underlying IGP topology.

Figure 6.4 summarizes the architecture of our tool. It takes BGP configuration files, RIBs and a map of IGP weights as inputs, performs dissemination

and ranking, and produces an SPVP instance S which is then passed to the GREEDY$^+$ algorithm.

Our tool has a core Java component which performs the dissemination phase, computes rankings, creates an SPVP instance, and runs GREEDY$^+$ on it. Besides that component, our prototype currently features:

(i) a minimal parser for Cisco configuration files, which is able to parse the most common BGP statements, based on some code from `BGP2CBGP` [Tan06];

(ii) an MRT [BKL09] parser for RIBs; and

(iii) an SNMP-based OSPF link weight parser, which computes the all-pairs shortest distance matrix.

We tested our prototype both on in vitro and on real world iBGP configurations. Namely, in order to evaluate how much our approach can scale to large networks, we analyzed synthetic iBGP topologies consisting of up to 1100 iBGP speakers and route reflection hierarchies having at least three levels. As in the case of eBGP networks, the most time-consuming activity is the dissemination phase. In this case, the processing time to perform the dissemination phase depends on the number of eBGP routes that need to be propagated, or, equivalently, on the number of feasible egress points for a single destination prefix. Since this number has been found to be quite low (lower than 20 in the worst case) even for very large networks [FRBS08], we injected 20 eBGP routes for each prefix as a worst-case analysis. Unfortunately, we cannot exploit the optimizations that we used in Section 6.3, as in iBGP the `local_pref` value travels within the announcement, rather than being set at the receiving BGP speaker. In other words, contrary to the eBGP case described above, it is not possible to compare a set of BGP announcements without enumerating them all. We then resort to a different technique to reduce the number of paths that need to be enumerated: when analyzing a given prefix p, we disregard all the iBGP routers that are neither route reflectors nor egress points for p.

Figure 6.5 shows the processing time needed to run a worst-case analysis on three-levels hierarchies and a varying number of iBGP speakers.

We ran our experiments on a entry-level server equipped with two 2.6 GHz quad-core CPUs and 16 GB RAM. Observe that checking the stability for a single prefix in a large network (e.g., 600 iBGP speakers) takes 0.3 seconds in the worst case. Running the analysis for the whole Internet routing table would take several hours. However, the stability check could be run only for the prefixes that experienced some change in a given time frame, e.g., 15-30 minutes. Moreover, performance can still be improved if prefixes can be

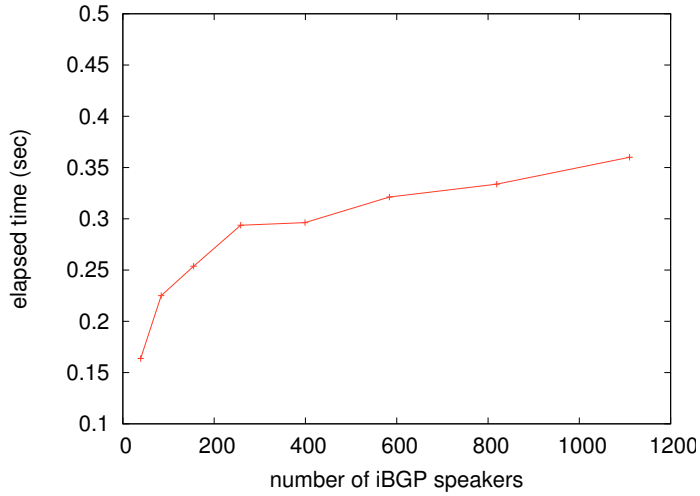


Figure 6.5: Processing time to check the stability of three-levels iBGP configurations with 20 injected eBGP routes.

grouped in equivalence classes, which is frequently the case, since BGP policies are seldom specified on a per-prefix basis. The number of equivalence classes is usually one or two orders of magnitude lower than the number of prefixes (see, e.g., [FRBS08]).

Further, in order to test all the components of the prototype, we checked the iBGP configuration of a medium-sized Italian ISP, consisting of almost 40 iBGP speakers and two route reflectors. We ran a test for every prefix in the full Internet routing table ($\approx 300,000$ prefixes) and found the configuration stable in all cases. The full test took only a few minutes.

6.5 Conclusions

Configuring routers in a way that enforces safety is a hard task, as hardly predictable interactions can give raise to routing oscillations. While considerable research efforts have focused on the theory of routing stability and on ways to design routing policies such that stability is guaranteed, the interest in testing router configurations for stability is relatively recent.

In this chapter we show that an automated check for BGP convergence is feasible even for large scale eBGP and iBGP networks. We describe a deterministic algorithm that is provably free from false positives and is able to compute a (possibly partial) stable routing tree. We prove that the algorithm is correct, that is, it never misreports a BGP network as safe if it is not. Moreover, even when the algorithm cannot prove the safety of the input BGP network, its output can be used to locate the trouble points in the network.

We exploit such algorithm in a convergence-checking tool, whose prototype implementation is efficient enough to process Internet scale eBGP topologies as well as very large iBGP networks. It should be noted that the general problem still has exponential complexity as, in the worst case, it takes exponential time to translate a real BGP network into an abstract instance that our algorithm can process. However, our results show that, on Internet-like network topologies, the complexity is still manageable. This confirms the findings in [FRBS08], with the remarkable difference that our technique supports iBGP attributes that are changed en route (see Chapter 5) as well as multiple layers of route reflection.

Chapter 7

Collecting BGP Data to Support What-If Analysis

7.1 Introduction

As we showed in Chapter 6, collecting BGP data is of primary importance if one wants to detect instabilities in eBGP or check iBGP configurations for guaranteed convergence. In the eBGP case, BGP data are needed in order to build a realistic topology and set up reasonable interdomain policies. In the iBGP case, instead, BGP data from border routers are a necessary input to translate configurations to an SPVP instance, and, ultimately, to run the stability checking algorithm (see Section 6.4).

Even though in this thesis we focused especially on BGP instabilities, monitoring BGP routes enables ISPs to perform many other business-critical activities like anomaly detection [MYC08, RGM+04], business/market intelligence [Gao01], traffic engineering [BL08], root cause analysis [CCD+08, FMM+04], routing table analysis [Hus01] and agreement compliance verification [FMR04].

Despite such a rich set of potential applications, current BGP monitoring practices are quite limited: very often, they employ open source BGP daemon implementations to establish extra BGP peerings with production routers. The daemon acts as a *route collector*, in the sense that it collects information received via those extra peerings, dumps it in some format, and stores it for future analysis. For example, this is the approach adopted by RouteViews [Ore] to collect BGP data for the Internet community. Such a practice has two major drawbacks: *(i)* it is only able to collect those routes that have been selected

as best by the routers that peer with the collector; and *(ii)* it is only able to collect BGP messages after ingress policy application, which can modify the messages themselves.

Unfortunately, these drawbacks prevent exploiting the monitoring system for interesting applications like, for example, fine tuning of ingress policies (e.g., for traffic engineering purposes) or verifying Service Level Agreements (SLAs) that involve BGP updates received by the other party. Even worse, these drawbacks prevent us from using the techniques described in Chapter 6 in order to analyze BGP stability in what-if scenarios (e.g., what if an upstream provider goes down? what if I decide to filter out a particular route?).

Recently, the BGP Monitoring Protocol [SFS] was proposed to overcome those limitations, but unfortunately it is still not deployed widely enough, as it requires firmware support on the routers. Moreover, it does not mandate real-time monitoring of BGP messages.

In this chapter we define a set of requirements that a BGP monitoring system should satisfy so that an ISP can take the biggest possible advantage from its deployment. We present a novel approach enabling real-time, non-invasive and scalable collection of all updates received by production-level BGP routers. For this purpose, we exploit a usually overlooked feature that allows a router to selectively clone IP packets and send them to a remote collector. We make use of such a feature to copy every incoming TCP segment belonging to BGP sessions and send it to a collector. After possibly reordering out-of-order segments, our collector parses the BGP messages and stores the information in the standard MRT format [BKL09].

By means of experimental evaluation on one of the cheapest commercial routers targeted to ISPs, we show that deploying our solution negligibly affects the performance of border routers with respect to traffic forwarding throughput and CPU usage. We show that our prototype implementation can monitor hundreds of BGP routers on commodity hardware. We also check the accuracy of the collected data. Finally, by comparing our approach to existing solutions for BGP data collection, we show that none of the previously proposed monitoring systems is able to fulfill all the requirements we considered.

The rest of this chapter is organized as follows. In Section 7.2, we define the requirements we mandate for an ideal BGP monitoring system. Section 7.3 surveys existing BGP monitoring solutions. In Section 7.4, we describe our proposal for a BGP monitoring system, outlining its architecture and discussing the most relevant components. Then, based on the requirements defined in Section 7.2, we evaluate our solution (Section 7.5) and we compare it with existing solutions (Section 7.6). We conclude in Section 7.7.

7.2 Requirements for a BGP Monitor

In this section, we describe a set of requirements that a BGP monitoring system should ideally fulfill.

Collection of non-best routes updates. Due to the way BGP is designed, BGP routers select a single *best* route among a set of feasible routes to a given destination, and forward traffic accordingly. Although updates related to routes not selected as best have no impact on where packets are forwarded, collecting them allows an ISP to better perform business intelligence activities like monitoring the quality of the service offered by its upstream provider at the BGP level. Another interesting application of non-best route collection is simulating what-if scenarios, answering questions like "What happens to paying traffic if I set a lower local-preference on this route?".

Accurate BGP update collection. Collected BGP data should reproduce the BGP updates received by other ISPs to the highest possible level of accuracy. This implies at least the ability to reconstruct all the BGP attributes that are involved in the best route selection process. We expect an ideal collection system to reconstruct the original BGP messages as sent by neighboring ISPs. Peculiar applications, e.g. scientific research, might also need accurate reproduction of BGP update timings. Also, collected data should not be affected by a change in the ISP's routing policies.

Real-time data collection. A BGP monitoring system should be able to collect data in real-time, or at least in near real-time. That is, a BGP update should be available for analysis within few seconds. This is a crucial requirement for troubleshooting and network diagnosing applications: network administrators want to know what is going on while it is going on, not hours later.

Low impact on router resources. A common constraint on management infrastructures is to have a small impact in terms of extra resource consumption at network devices. This is especially true for BGP monitoring, given that BGP border routers typically have to forward huge amounts of traffic between two ISPs. Hence, the operation of the monitoring system should not alter the performance of BGP routers, e.g. in terms of traffic forwarding capabilities or CPU usage.

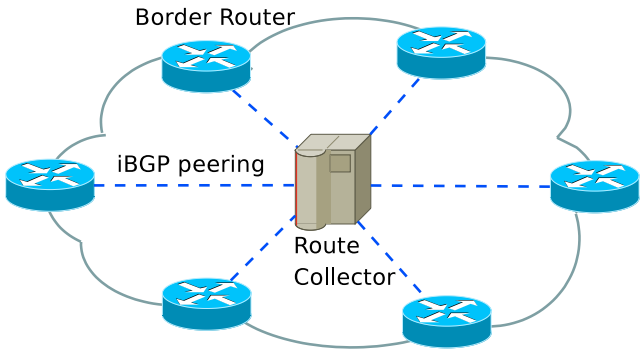


Figure 7.1: Typical architecture of BGP monitoring solutions employing BGP route collectors.

Scalability. An efficient monitoring system should be cheap in terms of network resource consumption (e.g., hardware needs, bandwidth overhead). To be realistically deployable in large scale networks, the monitoring system should be able to handle hundreds of border routers employing a handful of machines equipped with commodity hardware.

Low management overhead. A monitoring system cannot require the existing network structure to be modified. Also, deployment and management overhead, e.g., the amount of extra configuration needed to set up and maintain the system, should be as low as possible.

7.3 Related Work

In this section we describe existing solutions for BGP monitoring. We refer to Section 7.6 for a comparative analysis of prior work which shows that no existing solution satisfies all the requirements defined in Section 7.2, thus motivating new contributions in this area.

Existing solutions can be broadly classified in two categories: those employing some kind of route collectors that are pushed BGP messages by border routers, and those adopting separate protocols to pull BGP information from the routers.

The typical architecture of a BGP monitoring system belonging to the first category is depicted in Figure 7.1. A route collector is deployed inside the

network and iBGP peerings are set up with every border router. Quagga [Ish] and OpenBGPd [BJ] are probably the most famous and widespread tools to set up a route collector. Essentially, they act as a real router, but they support dumping BGP messages in MRT [BKL09] format. The Python Routing Toolkit (PyRT) [Mor], on the other hand, only implements a minimal feature set, and is lightweight enough to allow a single route collector to be deployed in a large ISP network.

BGP monitoring systems based on separate management protocols are designed to pull information from the devices. The main advantage of such an approach is that it typically does not need any extra configuration at the border routers: support for the management protocol suffices. SNMP has a number of MIB objects that are dedicated to BGP monitoring activities [HH06]. Often, operators pull information by *screen scraping*, i.e., using software that connects to the device, e.g., via Telnet or SSH, issues a specific command, e.g., `show ip bgp`, and collects the output.

Recently, a new ad-hoc protocol has been proposed in the IETF (the BGP Monitoring Protocol, or BMP) [SFS]. The idea is to send received BGP messages via a TCP connection with a monitoring station. Unfortunately, both standardization and router support are still in early stages, hence BMP is not yet a readily deployable solution in a production environment.

7.4 Proposed Architecture

In this section we propose an architecture for a BGP monitoring system. The main idea is to mandate *border routers* to capture all the incoming TCP segments belonging to BGP sessions with eBGP peers and forward them to a remote *route collector*. The route collector is responsible for reassembling the TCP segments, decoding BGP messages and storing them in the standard MRT format [BKL09]. In the following, we show that this technique can be implemented using common features available on routers together with ad-hoc software employed on the route collector side.

Figure 7.2 depicts the architecture of our solution in a typical deployment scenario. In this example, ISP A configures its border routers $BR1$ and $BR2$ to clone BGP packets and send copies to a remote Route Collector. Since packet cloning is performed before applying local policies, the route collector will receive BGP messages exactly as they are sent by eBGP peers. This feature allows ISP A to monitor what routes are announced by its peers B, C, and D. Of course, this approach supports private peerings between ISPs as well as

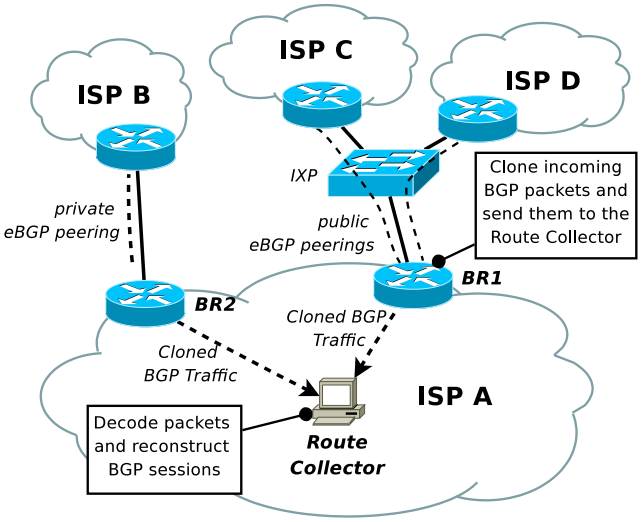


Figure 7.2: A deployment scenario of the proposed BGP monitoring system.

peerings at public Internet exchange points (IXPs).

It is easy to see in Figure 7.2 that our solution requires cooperation between two main architectural components: the border router (BR) and the route collector (RC). We now provide details regarding each component.

Border Routers: Cloning BGP Traffic

Most commercial routers provide the feature to clone IP packets and send copies to a remote machine. This is mostly used for copying traffic to Intrusion Detection Systems [cis06]. Leading vendors also provide filtering capabilities that allow operators to specify which packets must be cloned. Since such a feature is usually implemented in hardware, filters can usually be expressed based on IP source and destination addresses, and TCP source and destination ports only. However, for the purpose of cloning TCP segments belonging to BGP sessions, such simple filters suffice. To maintain a vendor-independent terminology, throughout the chapter we will refer to this feature as *Selective Packet Cloning* (SPC).

A BR configured to perform SPC copies packets received from user-specified *source interfaces* and matching an optional *filter* that selects which packets

should be cloned. Cloned traffic is forwarded via a *destination interface*.

Depending on the capabilities of the device, a destination interface can be either a physical interface (e.g., an Ethernet interface), a VLAN interface (via 802.1q encapsulation), or even a tunnel interface (e.g., IP-in-IP encapsulation or Generic Routing Encapsulation). Observe that forwarding cloned packets to a physical destination interface forces the ISP to place the RC so that it is directly connected to the BR, and is therefore unpractical.

In the following, we briefly describe the SPC feature as implemented in Cisco and Juniper devices.

The cheapest Cisco devices targeted to ISPs (e.g., Cisco 7200 and 7300 routers) provide the *Router IP Traffic Export (RITE)* feature [cis06]. A RITE-enabled router can select packets received on certain interfaces applying IP- and TCP-based filters, and forward cloned packets over a VLAN interface. More expensive Cisco routers (i.e., 7600 series or greater) support the *Encapsulated Remote SPAN (ERSPAN)* feature [cis], which provides a superset of the functionalities offered by RITE, e.g., the possibility to forward cloned traffic over a tunnel. Both RITE and ERSPAN can be used to implement the SPC feature on Cisco devices.

Juniper's SPC support is called *Port Mirroring* [jun]. Traffic received via user-specified ingress interfaces is cloned and forwarded over a VLAN or a tunnel (IP-in-IP or GRE) interface. On M7i devices and greater, cloning on a VLAN interface is performed in hardware, while tunnel interfaces are handled in software unless ad-hoc hardware (i.e., Tunnel Services PIC [JN]) is plugged into the router.

Depending on the SPC implementation, the TTL value might be decreased before cloning a packet. This is true both for Juniper's Port Mirroring and Cisco's RITE features. We cannot exclude that other SPC implementations behave the same. Unfortunately, this means that packets received with a TTL value equal to one cannot be cloned.

Since the default value of the TTL in eBGP is one, we must provide workarounds that make SPC usable within our context. For Cisco RITE, using a standard access control list, i.e., a filter that only matches fields in the IP header, solves the problem. Observe that this approach is reasonable because TCP traffic exchanged between border routers can be assumed to be mostly BGP traffic. Unfortunately, this workaround seems to be Cisco-specific. A more general workaround is to request peers to set up BGP session using a TTL greater than one. The recommended use of Generalized TTL Security Mechanism satisfies our needs since it forces the TTL to be set to the maximum value in order to protect BGP peerings from CPU-utilization based

RITE Configuration Steps

Step *(i)* - Define a *filter* to select BGP traffic

```
7201(config)#access-list 100 permit any any
```

Step *(ii)* - Define a *destination* interface

```
7201(config)#ip traffic-export profile myPr
7201(config-rite)#interface vlan1
7201(config-rite)#incoming access-list 100
                  mac-address <addr>
```

Step *(iii)* - Select one or more *source* interfaces

```
7201(config)#interface ge0/0
7201(config-if)#ip traffic-export apply myPr
```

Figure 7.3: Steps for configuring Selective Packet Cloning on Cisco routers.

attacks [GHM+07]. From the management point of view, our approach requires a small amount of extra configuration on the SPC-enabled border router. Figures 7.3 and 7.4 show the amount of extra configuration that is needed to enable the SPC feature on Cisco and Juniper routers, respectively. Steps *(i)* and *(ii)* only need to be performed once, while Step *(iii)* has to be repeated for each of the BR's interfaces that are used for BGP peerings.

Route Collector: Receiving, Reconstructing, and Storing BGP messages

Cloned TCP segments are sent to the RC which provides BGP messages to applications for further analysis. The RC performs the following activities, as summarized in Fig. 7.5.

Packet reception. The RC receives cloned packets from one of its network interfaces, and buffers them for further elaboration.

TCP stream reconstruction. Since the RC does not establish a TCP session with the BR, cloned TCP segments might arrive out of sequence. Therefore, for each eBGP peering the RC needs to reorder packets in order to extract the TCP stream. Duplicated segments are discarded. To keep resource consumption at the BR as low as possible, the RC silently ignores lost cloned TCP segments, if any.

Port Mirroring Configuration Steps

Step *(i)* - Define a *filter* to select BGP traffic

```
firewall {
  family inet {
      filter myPr {
          term bgp_mirror {
              from {port 179;}
              then {
                  port-mirror;
                  accept;
              }}
          term accept_all {then accept;}
      }}}
```

Step *(ii)* - Define a *destination* interface

```
forwarding-options {
  port-mirroring {
      family inet {
          input {rate 1;}
          output {
              interface <vlan1> {
                  next-hop A.B.C.D;
              }}}}}
```

Step *(iii)* - Select one or more *source* interfaces

```
fe-1/3/1 {
        unit 0 {
            family inet {
                filter {
                    input myPr;
                }
                ...
            }}}
```

Figure 7.4: Steps for configuring Selective Packet Cloning on Juniper routers.

BGP message decoding. The reconstructed TCP stream is analyzed to decode BGP messages and infer BGP session state changes.

BGP message storing. BGP messages and inferred state changes are stored in the standard MRT format [BKL09] or, optionally, inserted into a database ready to be analyzed by an application.

Our prototype RC implementation is based on the standard `tcpdump` utility for receiving cloned packets. We use `nice` to schedule the receiving process with high priority, and then send the received packets to a Perl script that is able to perform TCP stream reconstruction in pipeline. Finally, another Perl script takes the reconstructed stream in input and writes BGP messages in MRT format on a file. In the following, we discuss the main factors that affect the scalability of our implementation. Section 7.5 presents an experimental performance study.

Receiving speed. To avoid dropping some TCP segments, the RC must be able to receive packets at the speed they are sent on the network. Note that cloned TCP segments are received by the RC at approximately the same time when the BR received the original segments, the only difference being the cloning delay introduced by the BR and the network latency from the BR to the RC. The throughput of the TCP session between the BR and its BGP peer is limited by the TCP flow control mechanism, and it is roughly determined by the performance of the BGP software process running on the BR. The BGP software process, in turn, is bounded to the CPU speed of the BR. Given the current prices for commodity hardware, we can safely assume that the CPU speed of the RC exceeds, or is at least comparable with, the CPU speed of the BR. Moreover, the receiving process on RC just needs to buffer packets, a much less CPU-intensive task compared to what the BGP daemon on the BR needs to do. Hence, as long as the receiving process on the RC is scheduled with a sufficiently high priority, the receiving speed is not a problem.

Processing and storing speed. TCP stream reconstruction, BGP message decoding and data storage should be fast enough to sustain the average BGP traffic rate. Peak traffic rates are easy to accommodate by buffering received packets at the input of TCP stream reconstruction. All these activities take a constant amount of time for each BGP message, and the most critical with respect to processing time is the storage. A key feature of those three activities is that they are trivial to parallelize across

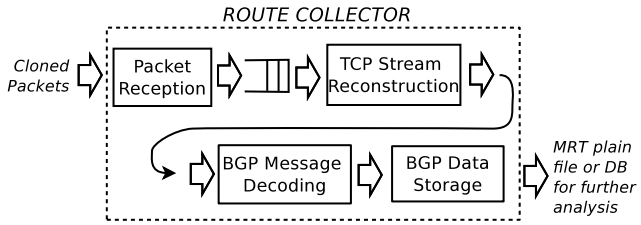


Figure 7.5: Main activities performed by the route collector software. Receiving a stream of cloned TCP segments as input, it reconstructs the TCP sessions and decodes BGP messages from them. Then, it stores BGP messages and state changes in MRT format.

multiple CPUs, allowing us to achieve good scalability by simply adding more processing resources to the RC. The possibility to improve write throughput of disks adopting RAID 0 is bounded only by the cost of additional disks.

7.5 Evaluation

In this section, we evaluate the extent to which our proposed architecture meets the requirements we defined in Section 7.2. We used a Cisco 7201 router (referred to *device-under-test* in the following) for measuring the performance of our solution. The router is equipped with four Gigabit Ethernet ports, 1 Gigabyte of RAM, and a 1.67 GHz Motorola Freescale 7448 processor. The vendor's datasheet states that this router is able to route a maximum of 2 million packets per second. We chose the Cisco 7201 because it is considered to be one of the cheapest router equipment targeted to ISPs. We aim at understanding how the SPC feature impacts the performance of the router under stress. In particular, we are interested in studying how SPC affects forwarding of regular traffic and in measuring the accuracy of BGP session reconstruction at the RC. We do not report any CPU usage measurements, since during all our experiments we were never able to appreciate any difference between SPC-enabled and SPC-disabled working mode. This can be easily explained by considering that the device-under-test implements the SPC feature entirely in hardware.

Baseline Measurement

First, we measured the performance of the device without any special configuration. In Section 7.5 and 7.5, we use the results of this experiment as a baseline for evaluating the impact of enabling the SPC feature on the device-under-test.

Figure 7.6 illustrates the baseline test topology. Our traffic generator (a SmartBits 600B) only has two interfaces and we connected both of them to the router. Note that a unidirectional traffic flow on a full-duplex Gigabit Ethernet link can generate a maximum of $1,488,095$ packets per second [KP02], which would not be enough to measure the maximum throughput of the router. For this reason, we configured our traffic generator to send bidirectional traffic, that is, traffic was sent from interface 1 to interface 2 and vice versa at the same time, as shown in Figure 7.6.

To make the router work properly in this setting, we configured it with 20 static routes, 10 for each interface connected to the traffic generator. We programmed the traffic generator to generate 100 unidirectional IP flows (i.e., source-destination pairs) by randomly picking a source address in each of the 10 prefixes configured on interface ge0/0 and a destination address in each of the 10 prefixes configured on interface ge0/1. The same was done in the opposite direction (from ge0/1 to ge0/0), for a total of 200 simulated IP flows. Traffic was sent at a fixed packet rate, evenly distributed among all flows (i.e., each flow got 1/200 of the traffic). Each packet was 64 bytes long, the minimum size allowed on Ethernet.

We measured packet loss at various packet transmission rates. Results are summarized in Figure 7.8, where we also show the results presented in the next section for comparison. The x-axis represents packet rate, expressed as the percentage with respect to maximum packet rate for full duplex Gigabit Ethernet. The y-axis represents frame loss, expressed as the ratio between lost frames and sent frames.

In our setting, the Cisco 7201 router can handle circa $1,845,000$ packets per second (62% of the maximum packet rate). The router was not able to handle the two million packets per second that the vendor's datasheet claims (vertical dashed line in Figure 7.8) without dropping frames. This is possibly a side effect of using only two interfaces or it might be due to our flows setting. Nevertheless, this fact does not affect the validity of this measure as a baseline for the following experiments.

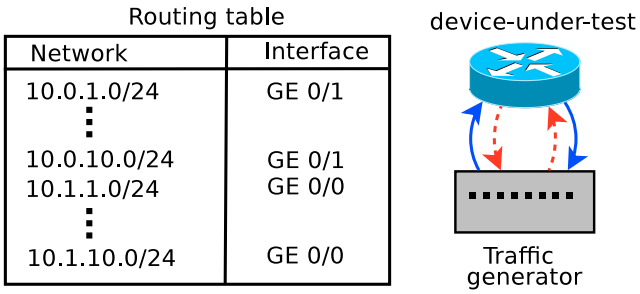


Figure 7.6: Baseline test topology. The device-under-test (Cisco 7201) is connected to the SmartBits 600B traffic generator with two cables. The traffic generator sends two flows in opposite directions.

Single Peering Scenario

After having performed the baseline measurement described in the previous section, we evaluated the router performance in a single BGP peering scenario. We set up a testbed using the topology depicted in Figure 7.7. The device-under-test was connected to the traffic generator as in the baseline experiment. Also, the device-under-test was configured in the same way and the same 200 IP flows were sent by the traffic generator to the router. On a third interface of the router we set up a BGP peering with a medium sized ISP. From this BGP peering, the router received the full routing table, containing 287,000 prefixes, and a continuous stream of real world BGP updates. We configured SPC such that incoming traffic belonging to the BGP peering was cloned on the fourth interface of the router over a VLAN. A packet sniffer was attached to the same VLAN and acted as a RC, capturing the cloned packets.

We performed the same experiment described in Section 7.5, the only difference being the size of the routing table, which, in this case, was increased by the full Internet routing table received over the BGP peering. We performed the test both with the SPC feature activated (test "BGP-updates-mirror") and not activated (test "BGP-updates-no-mirror"). Results are presented in Figure 7.8. For convenience, we also report the baseline measurement results (test "baseline") on the same diagram. It is easy to see that activating the SPC feature has no impact on the throughput achieved by the device-under-test. Moreover, we found that the presence of a single BGP peering does not cause more packets to be dropped. This can be explained by noting that, since the

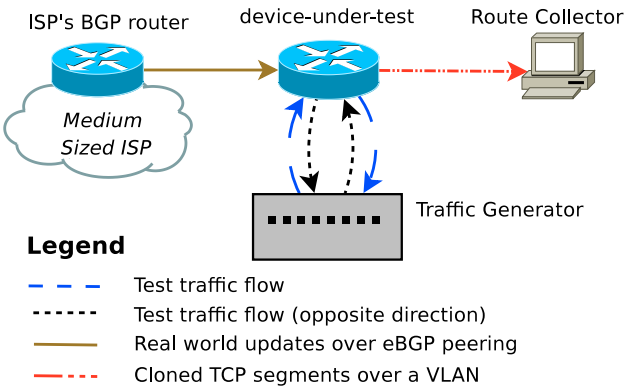


Figure 7.7: SPC test topology, an enriched version of the baseline test topology. Device-under-test is configured with a BGP peering on which real world updates are received, and SPC is enabled to clone BGP traffic toward the Route Collector.

synthetic traffic is routed using static entries, the portion of the FIB that is accessed never changes, making BGP-induced FIB changes irrelevant to the test traffic.

We repeated the experiment increasing the number of BGP peerings established by the device-under-test (up to five peerings), and found very similar results. Finally, we checked the correctness of cloned traffic by comparing the packet traces captured at the ISP's BGP router with the cloned packets received by the collector. We found that no cloned packets were dropped and BGP messages were correctly reconstructed and stored on disk. Since this check was successful in all our experiments, even when some regular traffic was dropped, we will not stress it again in the following.

Update Bursts with Multiple Peerings

We set up a second experiment to evaluate how SPC affects the performance of production BRs under heavy BGP update bursts. The topology of the testbed is similar to the one we described in the previous experiment (see Figure 7.7), except for the fact that the we interposed five BGP daemons (i.e., five Quagga [Ish] processes) between the BR and the device-under-test. Each

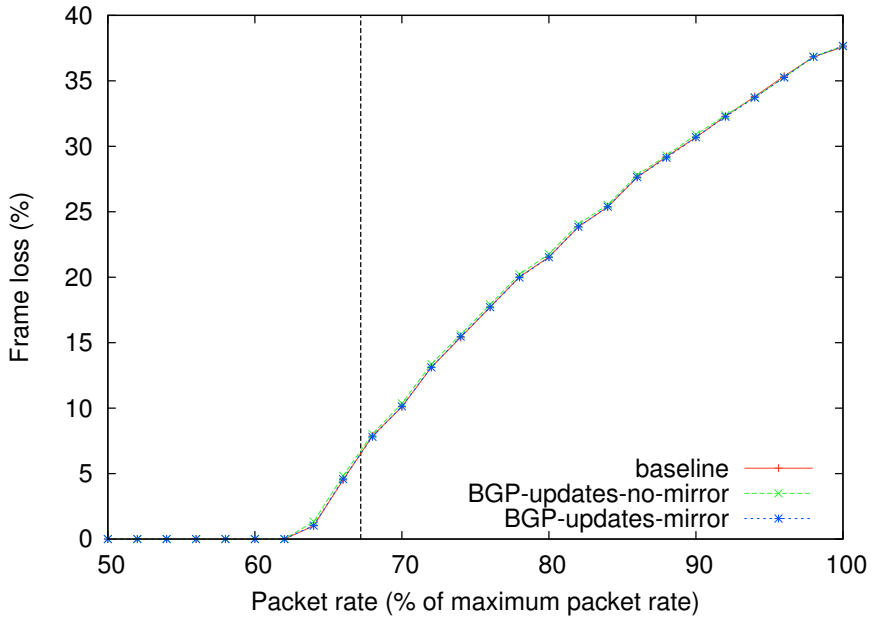


Figure 7.8: Frame loss (ratio between lost packets and sent packets) versus packet rate (percentage of the maximum packet rate obtainable on full-duplex Ethernet). The vertical dashed line represents the packet rate which should be handled without loss according to the vendor's datasheet. Frame loss increases roughly linearly with packet rate, and activating SPC does not impact the performance of the device.

BGP daemon had a peering session with the ISP and one iBGP peering session with the device-under-test. This way, whenever a BGP update was sent by the ISP's BGP router, each BGP daemon sent an update to the device-under-test, hence the update rate received at the device-under-test was multiplied by five. By tearing down the BGP sessions with the ISP's BGP router, we were then able to produce a huge amount of route withdrawals: in fact, the entire Internet full routing table was withdrawn by each of the five BGP daemons, and the device-under-test received almost 1.5 million route withdrawals. Conversely, as soon as the BGP sessions were restored, the ISP's BGP router advertised the full routing table to all BGP daemons, and the device-under-test received

almost 1.5 million route announcements.

We want to understand the impact of the SPC feature on packet loss when the BR is under extreme stress. For this purpose, we configured our traffic generator to send a critical amount of traffic, namely slightly more than 60% of the maximum packet rate obtainable on a full-duplex Gigabit Ethernet. At regular intervals, we alternately tore down and restored the BGP sessions with the ISP's BGP router, hence producing huge peaks of route withdrawals and announcements, respectively. We stress that such a scenario is extremely unrealistic, since routers of an ISP should not be (and typically are not) so overloaded by regular traffic in the real world, and do not receive such huge amounts of BGP updates. We run the experiment both with SPC disabled (test "reset-no-mirror") and enabled (test "reset-mirror"). Figure 7.9 summarizes our results: the x-axis represents time, while the y-axis represents frame loss as measured by our traffic generator. We found that the device-under-test lost a small fraction of traffic, about 0.005%, even when working with SPC disabled, as shown by the blue dotted line in Figure 7.9. As predictable, packet loss spikes correspond to the reception of BGP update bursts. The spikes are higher when the SPC feature is activated on the router, but the performance of the router is affected to a very small extent, as it is evident by observing that packet loss never reached 0.04%. Moreover, since we are very near to the maximum throughput that can be achieved by the device, packet loss is likely due to the BGP traffic itself rather than to the elaboration of FIB/RIB changes.

Performance of the Collector Software

To assess the amount of resources required on the RC side, we captured five BGP sessions during the initial full table transfer (nearly 1.5 million prefix updates, 37,157 TCP segments, most of them of the maximum length). We separately measured the processing time needed for receiving the packets, reconstructing the TCP stream, decoding BGP messages and storing them in MRT format on commodity hardware (a laptop equipped with a dual-core 2.6 GHz CPU and 4G of RAM). We stress that summing the measures we obtained in this experiment provides an upper bound on the performance that can be achieved by a RC, since processing times can be greatly enhanced by enabling pipelining and parallel processing, as all the activities are trivial to parallelize across multiple processors.

We re-played the capture file with `tcpreplay` using the topspeed option on a 100Mbit ethernet link connected to our prototypical RC. Actual throughput is about 80Mbit/sec, much higher than the throughput of regular BGP sessions.

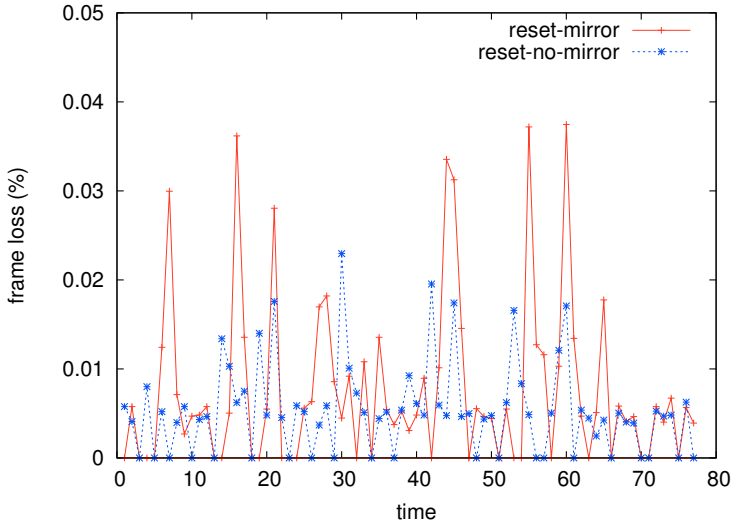


Figure 7.9: Frame loss during bursts of BGP updates with device load near to the maximum throughput. The bursts of updates are generated by announcing and withdrawing five entire BGP routing tables on five distinct peerings. Observe that the SPC feature affects performance only to a very small extent which is probably due to the BGP traffic itself.

Re-playing the capture file with `tcpreplay` took 3.38 seconds, while originally the BGP sessions lasted more than 2 minutes. A regular BGP session can reach such a high speed just sporadically. Even in this extreme experiment, we were able to capture all packets with `tcpdump` and store them to an output file. TCP stream reconstruction from the output file took 2.6 seconds, while BGP session decoding and storage in MRT format took 1.7 seconds. Overall, a single prefix update was processed in less than 5.23 μseconds on average. Given that real world BGP sessions exhibit an average of less than 100 prefix updates in a second, our prototype implementation can handle hundreds of BRs on commodity hardware.

	Quagga Open-BGPd	**PyRT**	**SNMP Screen scraping**	**BMP**	**SPC**
non-best routes	no	no	yes	yes	yes
accuracy	bad	bad	bad	good	perfect
real-time collection	no (see text)	no (see text)	no	almost (see text)	yes
impact on router resources	low	low	heavy	low	very low
scalable deployment	no	yes	yes	yes	yes
management overhead	low	low	none	requires support from routers	low

Table 7.1: Comparison between our solution and related work with respect to the requirements defined in Section 7.2.

7.6 Comparison with Related Work

We now compare our proposal with the existing solutions that we listed in Section 7.3, assessing the extent to which the requirements defined in Section 7.2 are satisfied by each solution. A summary of the comparison is presented in Table 7.1. In the following, we discuss the comparison results displayed in the table.

Collection of Non-Best Routes Since Quagga, OpenBGPd, and PyRT are based on an iBGP peering, updates for routes that the BR does not select as best routes will never be collected at the RC. Non-best routes can be collected by screen scraping (e.g., via `show ip bgp` queries), and there exist SNMP managed objects for every route received by a BGP peer. BMP and the solution we present in this chapter are currently the only way to continuously monitor non-best routes.

Accuracy of BGP Session Reconstruction Quagga, OpenBGPd, and PyRT can only monitor routes selected as best, and they are forced to collect BGP messages after ingress policy application. Polling-based mechanisms

such as SNMP and screen scraping are restricted to periodic snapshots of the received routes. For these reasons, the BGP session cannot be accurately reconstructed using those tools. BMP provides a more accurate view of the BGP session, however multiple BGP updates could be collapsed into a single one, and the timings of the messages could be altered. Moreover, BMP does not collect BGP control messages such as keepalives. Our solution, instead, clones each packet belonging to the BGP session as soon as it arrives to the BR, and provides a very good approximation of the time when the BGP message was received, the only delay being the network latency from the BR to the RC.

Real-Time Collection Solutions that employ additional iBGP peerings, such as Quagga, OpenBGPd and PyRT, are, in principle, capable of collecting BGP messages in real time. However, two issues should be considered. *(i)* The BR should send updates as soon as possible, which is the default behavior in iBGP; and *(ii)* if messages are dumped periodically, additional delay is introduced before data are available for an application to analyze. For example, Quagga can dump BGP data not faster than one file per minute. Real-time is of course unfeasible with SNMP and other polling-based mechanisms. The current BMP specification asserts that BMP messages "are not real time replicated messages received from a peer" [SFS].

Impact on Router Resources Handling an iBGP peering is a lightweight task for a BR, hence solutions based on Quagga, OpenBGPd, or PyRT do not put stress on routers. On the other hand, polling-based solutions employing SNMP or screen scraping heavily affect the performance at the BR, since it must process the whole BGP table and send a snapshot to the monitoring station. Since the SPC feature is performed in hardware, our approach affects the performance of the BR only minimally, see Section 7.5. Since BMP uses a TCP connection, it is not clear what the router resource consumption would be under extreme circumstances, e.g., when the RC tries to slow down the BR by shrinking the TCP window.

Scalability Since Quagga and OpenBGPd emulate a real router, CPU cycles and memory are wasted at the route collector for activities that are useless to a BGP monitoring system, e.g., performing the best route selection process. This makes them unable to handle a large number of peers providing a full Internet routing table, in turn making the deployment of a BGP monitoring system harder since multiple collectors must be

installed. PyRT is not affected by this problem since it only implements a minimal feature set, disregarding activities that are not relevant to the monitoring system. Since SNMP and screen scraping have no real-time constraint, a single monitor could be able to handle hundreds of BRs. The performance study in Section 7.5 ensures that a single RC is able to handle cloned BGP messages (and, with slight modifications, BMP messages as well) coming from hundreds of BRs.

Management Overhead Quagga, OpenBGPd and PyRT incur little management overhead, since all that is needed is to configure iBGP peerings between the BRs and the RC(s). SNMP and screen scraping virtually incur no management overhead, since they are commonly already used for other purposes in most ISP networks. The management overhead for our approach consists of extra router configuration as discussed in Section 7.4, plus the setup of a VLAN or tunnel from the BR to the RC. Deploying BMP, on the other hand, requires non-negligible firmware and/or hardware upgrade efforts: only JunOS versions later than 9.5 currently support BMP.

7.7 Conclusions

Once an algorithm for detecting BGP instabilities is available, perhaps its most interesting application is analyzing what-if scenarios, for example, to test a new configuration before it gets deployed.

To support the above scenario, as well as better troubleshooting and other business intelligence analyses, we propose an innovative technique for real-time collection of all BGP messages sent by BGP peers.

Through experiments, we show that our approach accurately records the BGP updates received, it is easy to configure on current routers, it is scalable, and it has a negligible impact on the performance of the monitored border routers.

We believe that our approach based on selective packet cloning could turn out to be useful also for monitoring other routing protocols.

Conclusions and Bibliography

Conclusions and Open Problems

Routing is a much harder problem than it seems at a first glance, and this is especially true for interdomain routing protocols, where the need to account for routing policies prevents us from simply representing a network as a graph and trying to optimize a single metric network-wide. Instead, with BGP we have a huge network of independent ASes, each trying to fulfill its own purpose while cooperating with its neighbors to disseminate reachability information.

Unfortunately, unrestricted local policies are incompatible with guaranteed convergence. Thanks to our characterization of BGP safety under filtering, we are now able to define exactly the amount of expressiveness that needs to be sacrificed in BGP to preserve complete filtering autonomy while still ensuring global stability. A similar relation exists between expressiveness and stability when we focus on the internal version of BGP, iBGP. In particular, we study the impact of iBGP attribute manipulation and, once again, we find that with more expressiveness comes an increased risk for routing oscillations.

Interestingly, we find that the foundational properties that characterize BGP stability can be found just in a static description of the network, without the need to deal with the complexity of the dynamic, message-oriented nature of BGP. This insight suggested us that it is possible to infer the stability of a BGP network by just looking at its configuration. We then devised an algorithm that is able to tell whether a given policy configuration is stable. Our algorithm takes as input an abstract representation of BGP routing policies, and computes a (possibly partial) stable routing state. If the output is a complete routing state, the network provably converges to it. Otherwise, if the output is a partial routing state, then the network is potentially unstable, and the trouble points must be searched among those portions of the network that are not included in the output. We show that the algorithm as well as the translation steps from BGP topologies or iBGP configuration files can be implemented efficiently enough to analyze large scale BGP and iBGP networks.

Finally, we observe that our techniques could be applied to analyze what-if scenarios, allowing us to test a BGP configuration for stability before it gets deployed. In order to perform this kind of analysis, however, BGP data obtained through standard collection systems do not suffice, since they are not able to provide those routes that are not selected as best and they are unavoidably biased by ingress policy application. Hence, we propose a scalable and efficient BGP data collection system which is able to overcome such limitations.

Another contribution of this thesis is the comparison between different variants of BGP models that have been extensively used in the literature. We provide a taxonomy of the proposed BGP models, and we mathematically prove that variations can impact the ability of the model to capture special kinds of routing oscillations.

Yet, as pointed out in our review of related work, there is plenty of room for further research activities. In particular, it is still not clear how hard it is to decide whether a given BGP network is safe under filtering or not. While we have provided a formal tool, namely the dispute reel, to statically analyze safety under filtering, the computational complexity of finding such a structure in a given policy configuration is still unknown.

Another important theoretical problem is how hard it is to decide whether a given BGP network has a stable state when the policy configuration is expressed using a compact language, as it is the case today. While we show some preliminary results for the special case where the ranking component is next-hop based or where no route filters are allowed, the general problem remains open.

In our opinion, the existence of incompatible sets of policies leading to routing oscillations and the fact that most of the interesting problems are known (or conjectured) to be computationally intractable indicate that the Internet community should have investigated BGP theory before its wide deployment, rather than after. Unfortunately, the demand for features resulted in having the BGP code long before we had a reliable BGP model. As it often happens in computer science, inferring a model from running code is a much harder task than writing the code that implements a running model. As a result, none of the currently existing models for BGP can claim to realistically represent the real functioning of a router. Hence, bridging the gap between the model and the implementation becomes a crucial and challenging task.

From a more general point of view, perhaps the most important problem of the current interdomain routing infrastructure is the fact that it is overworked. In order to accommodate the growing complexity of the Internet, BGP is now much more than a routing protocol: besides disseminating reachability infor-

mation, it is used as a mean to do load balancing, traffic engineering, failover, content distribution, etc. All these features push the demand for expressive routing policies, while we showed that there is a clear limit to the expressiveness that can be supported if one requires guaranteed convergence. In the light of the new research efforts trying to redesign the Internet architecture from scratch, we believe that a next generation interdomain routing protocol should promote a clean separation between the routing functionality and the traffic engineering mechanisms.

Other Research Activities

This thesis was originally spurred by a research interest on BGP instabilities. From time to time, the general interest in understanding and improving BGP led us to tackle a number of side research issues by exploring related research areas. Since those side research activities do not perfectly fit the scope of this thesis, they are briefly summarized in this chapter.

- *Clean Slate Design.* While it seems that the Internet, or at least the vast majority of it, is able to reach a stable routing with BGP, the number of BGP updates that are exchanged in the network steadily increases. The current size of the Internet poses a trade-off if BGP is used to disseminate routing information for the whole network, as every additional BGP message has the potential to trigger a new routing table computation, slowing down the routers. In this context, we designed HAIR, a routing architecture that splits the Internet into a hierarchy and keeps BGP updates as localized within the same hierarchical component as possible.

- *Root Cause Analysis.* BGP is an incremental protocol, designed to generate messages only in response to network events (e.g., a link fault, or a router reset). Yet, BGP update rates in the Internet are so high that it is extremely hard to tell whether they are caused by BGP instabilities or they are part of the "normal" functioning of the network. Root Cause Analysis is a research field that aims at identifying and locating the root cause of BGP messages, mostly for debugging and troubleshooting purposes. In this context, we proposed a methodology which is based on the number of prefixes routed on each interdomain link. The methodology is partially supported by a graphical tool that aids the analysis of a big collection of BGP messages.

- *IPv4 Address Space Deaggregation.* Currently, the most effective way that an AS has at its disposal to control how traffic enters is prefix deaggregation, that is, using BGP to advertise several more specific prefixes alongside with the aggregate. There is widespread belief in a high and recently growing number of ASes that inject deaggregated prefixes, e.g., for due to multihoming or for the purpose of traffic engineering. In this context, we show that the there is no trend towards more aggressive prefix deaggregation or traffic engineering over time. With respect to BGP dynamics, we observe that deaggregated prefixes do not, generally, generate the disproportionate amount of BGP updates they are believed to.

Publications

Conference Publications

1. L. Cittadini, G. Di Battista, S. Vissicchio. Doing Don'ts: Modifying BGP Attributes within an Autonomous System. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2010)*, IEEE, 2010.

2. A. Feldmann, L. Cittadini, W. Mühlbauer, R. Bush, O. Maennel. HAIR: Hierarchical Architecture for Internet Routing. In *Proc. ReArch 2009*, ACM, 2009.

3. L. Cittadini, G. Di Battista, M. Rimondini, S. Vissicchio. Wheel + Ring = Reel: the Impact of Route Filtering on the Stability of Policy Routing. In *Proc. International Conference on Network Protocols (ICNP 2009)*, IEEE, 2009.

4. P. Angelini, L. Cittadini, G. Di Battista, W. Didimo, F. Frati, M. Kaufmann, A. Symvonis. On the Perspectives Opened by Right Angle Crossing Drawings. In *Proc. International Symposium on Graph Drawing (GD 2009)*, Springer, 2009.

5. L. Cittadini, M. Rimondini, M. Corea, G. Di Battista. On the Feasibility of Static Analysis for BGP Convergence. In *Proc. International Symposium on Integrated Network Management (IM 2009)*, IEEE, 2009.

6. A. Di Menna, T. Refice, L. Cittadini, G. Di Battista. Measuring Route Diversity in the Internet from Remote Vantage Points. In *Proc. International Conference on Networks (ICN 2009)*, IEEE, 2009.

7. L. Cittadini, G. Di Battista, M. Rimondini. (Un)-Stable Routing in the Internet: A Survey from the Algorithmic Perspective. In *Proc. International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008)*, Springer, 2008.

8. L. Cittadini, T. Refice, A. Campisano, G. Di Battista, C. Sasso. Policy-aware Visualization of Internet Dynamics. In *Proc. International Symposium on Graph Drawing (GD 2008)*, Springer-Verlag, 2008.

9. A. Campisano, L. Cittadini, G. Di Battista, T. Refice, C. Sasso. Tracking Back the Root Cause of a Path Change in Interdomain Routing. In *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, IEEE, 2008.

10. L. Cittadini, T. Refice, A. Campisano, G. Di Battista, C. Sasso. Measuring and Visualizing Interdomain Routing Dynamics with BGPath. In *Proc. IEEE Symposium on Computers and Communications (ISCC 2008)*, IEEE, 2008.

Internet-Drafts

1. G. Bajko, S. M. Bellovin, R. Bush, L. Cittadini, A. Durand, O. Mannel, T. Savolainen, J. Zorz. The A+P Approach to the IPv4 Address Shortage. Internet draft draft-ymbk-aplusp-05.txt, work in progress, Internet Engineering Task Force, 2009.

Technical Reports

1. A. Feldmann, R. Bush, L. Cittadini, O. Maennel, W. Mühlbauer. HAIR: Hierarchical Architecture for Internet Routing. Technical Report 2008-14, Technische Universitaet Berlin, 2008.

2. O. Maennel, R. Bush, L. Cittadini, S. M. Bellovin. A Better Approach than Carrier-Grade-NAT. Technical Report CUCS-041-08, Dept. of Computer Science, Columbia University, 2008.

3. L. Cittadini, G. Di Battista, M. Rimondini. How Stable is Stable in Interdomain Routing: Efficiently Detectable Oscillation-Free Configurations. Technical Report RT-DIA-132-2008, Dept. of Computer Science and Automation, Roma Tre University, 2008.

4. A. Antony, L. Cittadini, D. Karrenberg, R. Kisteleki, T. Refice, T. Vest, R. Wilhelm. Mediterranean Fiber Cable Cut (January-February 2008) Analysis of Network Dynamics. Technical Report RT-DIA-124-2008, Dept. of Computer Science and Automation, University of Roma Tre, 2008.

5. A. Campisano, L. Cittadini, G. Di Battista, T. Refice, C. Sasso. Update-Driven Root Cause Analysis in Interdomain Routing. Technical Report RT-DIA-117-2007, Dept. of Computer Science and Automation, University of Roma Tre, 2007.

Bibliography

[AKS06] Ehoud Ahronovitz, Jean-Claude Konig, and Clément Saad. A distributed method for dynamic resolution of BGP oscillations. In *Proc. IPDPS 2006*, Apr 2006.

[AVG+99] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, and M. Terpstra. Routing Policy Specification Language (RPSL). RFC 2622, 1999.

[BBAS03] Anat Bremler-Barr, Yehuda Afek, and Shemer Schwarz. Improved BGP convergence via ghost flushing. In *Proc. INFOCOM 2003*, volume 2, pages 927–937, 2003.

[BCC06] T. Bates, E. Chen, and R. Chandra. BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP). RFC 4456, 2006.

[BJ] H. Brauer and C. Jeker. OpenBGPd. `www.openbgpd.org`.

[BKL09] L. Blunk, M. Karir, and C. Labovitz. MRT routing information export format. Internet-Draft, draft-ietf-grow-mrt-10.txt, 2009.

[BL08] Simon Balon and Guy Leduc. Combined intra- and inter-domain traffic engineering using hot-potato aware link weights optimization. In *Proc. SIGMETRICS*, 2008.

[BOR+02] Anindya Basu, Chih-Hao Luke Ong, April Rasala, F. Bruce Shepherd, and Gordon Wilfong. Route oscillations in i-bgp with route reflection. In *Proc. SIGCOMM*, 2002.

[CAI] CAIDA. AS topologies annotated with AS relationships. http://www.caida.org/data/active/as-relationships/index.xml.

[CCD+08] Alessio Campisano, Luca Cittadini, Giuseppe Di Battista, Tiziana Refice, and Claudio Sasso. Tracking back the root cause of a path change in interdomain routing. In *Proc. NOMS*, 2008.

[CGM03] Jorge Arturo Cobb, Mohamed G. Gouda, and Ravi Musunuri. A stabilizing solution to the stable path problem. In *Proc. Self-Stabilizing Systems*, pages 169–183, 2003.

[cis] Configuring local span, remote span (rspan), and encapsulated rspan (erspan). Cisco Systems, Inc. Official Cisco ERSPAN documentation.

[cis06] Router ip traffic export packet capture enhancements. Cisco Systems, Inc., 2006. Official Cisco RITE documentation.

[CR05] Matthew Caesar and Jennifer Rexford. Bgp routing policies in isp networks. *IEEE Network*, 19(6):5–11, 2005.

[DD08] Amogh Dhamdhere and Constantine Dovrolis. Ten Years in the Evolution of the Internet Ecosystem. In *Proc IMC*, 2008.

[DEH+07] Giuseppe Di Battista, Thomas Erlebach, Alexander Hall, Maurizio Patrignani, Maurizio Pizzonia, and Thomas Schank. Computing the types of the relationships between autonomous systems. *IEEE/ACM Trans. on Networking*, 15(2):267–280, 2007.

[DRCD09] Andrea Di Menna, Tiziana Refice, Luca Cittadini, and Giuseppe Di Battista. Measuring Route Diversity in the Internet from Remote Vantage Points. In *Proc. ICN*, 2009.

[ERC+07] Cheng Tien Ee, Vijay Ramachandran, Byung-Gon Chun, Kaushik Lakshminarayanan, and Scott Shenker. Resolving inter-domain policy disputes. Technical Report UCB/EECS-2007-27, EECS Department, University of California, Berkeley, Feb 2007.

[FB05] Nick Feamster and Hari Balakrishnan. Detecting BGP configuration faults with static analysis. In *Proc. NSDI*, 2005.

[FBR04] Nick Feamster, Hari Balakrishnan, and Jennifer Rexford. Some Foundational Problems in Interdomain Routing. In *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[FJB05] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Stable policy routing with provider independence. Technical Report MIT-LCS-TR-981, MIT, 2005.

[FJB07] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. Implications of autonomy for the expressiveness of policy routing. *IEEE/ACM Trans. on Networking*, 15(6):1266–1279, Dec 2007.

[FMM+04] Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs. Locating Internet Routing Instabilities. In *Proc. SIGCOMM*, 2004.

[FMR04] Nick Feamster, Zhuoqing Morley Mao, and Jennifer Rexford. BorderGuard: detecting cold potatoes from peers. In *Proc. IMC*, 2004.

[FP08] Alex Fabrikant and Christos Papadimitriou. The complexity of game dynamics: BGP oscillations, sink equilibria, and beyond. In *Proc. SODA*, pages 844–853, 2008.

[FR07] Nick Feamster and Jennifer Rexford. Network-wide prediction of BGP routes. *IEEE/ACM Trans. Netw.*, 15(2):253–266, 2007.

[FR09] Ashley Flavel and Matthew Roughan. Stable and flexible ibgp. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009.

[FRBS08] Ashley Flavel, Matthew Roughan, Nigel Bean, and Aman Shaikh. Where's Waldo? Practical Searches for Stability in iBGP. In *Proc. ICNP*, 2008.

[FSS06] Joan Feigenbaum, Rahul Sami, and Scott Shenker. Mechanism design for policy routing. *Distributed Computing*, pages 293–305, 2006.

[FSS07] Joan Feigenbaum, Michael Schapira, and Scott Shenker. Algorithmic game theory. In *Distributed Algorithmic Mechanism Design*,

pages 363–384, New York, NY, USA, 2007. Cambridge University Press.

[Gao01] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. on Networking*, 9(6):733–745, 2001.

[GGR01] Lixin Gao, Timothy Griffin, and Jennifer Rexford. Inherently safe backup routing with BGP. In *Proc. INFOCOM 2001*, pages 547–556, 2001.

[GHM+07] V. Gill, J. Heasley, D. Meyer, P. Savola, and C. Pignataro. The generalized ttl security mechanism (GTSM). RFC 5082, 2007.

[GJR03] Timothy G. Griffin, Aaron D. Jaggard, and Vijay Ramachandran. Design principles of policy languages for path vector protocols. In *Proc. SIGCOMM 2003*, pages 61–72, New York, NY, USA, 2003. ACM Press.

[GR00] Lixin Gao and Jennifer Rexford. Stable Internet routing without global coordination. In *Proc. SIGMETRICS 2000*, pages 307–317, 2000.

[GS05] Timothy G. Griffin and Joao Luís Sobrinho. Metarouting. In *Proc. SIGCOMM 2005*, pages 1–12, 2005.

[GSW99] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. Policy disputes in path-vector protocols. In *Proc. ICNP 1999*, pages 21–30, 1999.

[GSW02] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. on Networking*, 10(2):232–243, 2002.

[GW99] Timothy G. Griffin and Gordon Wilfong. An analysis of BGP convergence properties. *Proc. SIGCOMM 1999*, 29(4):277–288, 1999.

[GW00] Timothy G. Griffin and Gordon T. Wilfong. A safe path vector protocol. In *Proc. INFOCOM 2000*, pages 490–499, 2000.

[GW02a] Timothy G. Griffin and Gordon Wilfong. On the correctness of IBGP configuration. *Proc. SIGCOMM 2002*, 32(4):17–29, 2002.

[GW02b] Timothy G. Griffin and Gordon T. Wilfong. Analysis of the MED oscillation problem in BGP. In *Proc. ICNP 2002*, pages 90–99, 2002.

[HH06] J. Haas and S. Hares. Definitions of managed objects for BGP-4. RFC 4273, 2006.

[Hus01] Geoff Huston. Analyzing the internet's BGP routing table. *The Internet Protocol Journal*, 4(1), 2001.

[HW08] P. E. Haxell and G. T. Wilfong. A fractional model of the border gateway protocol (BGP). In *Proc. SODA '08*, 2008.

[ISC09] Internet Systems Consortium ISC. ISC domain survey. http://www.isc.org/solutions/survey, 2009.

[Ish] K. Ishiguro, et al. Quagga routing suite. `www.quagga.net`.

[JN] Inc. Juniper Networks. Tunnel service pic datasheet. Datasheet.

[JR04] Aaron D. Jaggard and Vijay Ramachandran. Robustness of class-based path-vector systems. In *Proc. ICNP 2004*, pages 84–93, Oct 2004.

[JR05] Aaron D. Jaggard and Vijay Ramachandran. Relating two formal models of path-vector routing. In *Proc. INFOCOM 2005*, pages 619–630, Mar 2005.

[JR06] Aaron D. Jaggard and Vijay Ramachandran. Robust path-vector routing despite inconsistent route preferences. In *Proc. ICNP 2006*, pages 270–279, 2006.

[jun] Configuring port mirroring. Juniper Networks, Inc. Official Juniper Port Mirroring Documentation.

[kC06] Chi kin Chau. Policy-based routing with non-strict preferences. In *Proc. SIGCOMM 2006*, pages 387–398, 2006.

[kCGG06] Chi kin Chau, Richard Gibbens, and Timothy G. Griffin. Towards a unified theory of policy-based routing. In *Proc. INFOCOM 2006*, pages 1–12, Apr 2006.

[KCM04] Tomas Klockar and Lenka Carr-Motyčková. Preventing oscillations in route reflector-based I-BGP. In *Proc. ICCCN 2004*, pages 53–58, 2004.

[Kin08] Shiva Kintali. A distributed protocol for fractional stable paths problem. In *Proc. DIMACS/DyDAn Workshop on Secure Internet Routing*, 2008.

[KKK07] Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now?! It must be BGP. In *Computer Communication Review*, 2007.

[KLMS00] Stephen Kent, Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure border gateway protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18:103–116, 2000.

[KMT06] Sven Kosub, Moritz G. Maaß, and Hanjo Täubig. Acyclic type-of-relationship problems on the Internet. In *In Proceedings of the 3rd Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN 06)*, pages 98–111. Springer-Verlag, 2006.

[KP02] Scott Karlin and Larry Peterson. Maximum packet rates for full-duplex ethernet. Technical Report TR64502, Department of Computer Science Princeton University, 2002.

[LXHL02] Jiazeng Luo, Junqing Xie, Ruibing Hao, and Xing Li. An approach to accelerate convergence for path vector protocol. In *Proc. GLOBECOM 2002*, volume 3, pages 2390–2394, 2002.

[MC04a] Ravi Musunuri and Jorge Arturo Cobb. A complete solution for IBGP stability. In *Proc. IEEE International Conference on Communications (ICC 2004)*, volume 2, pages 1177–1181, Jun 2004.

[MC04b] Ravi Musunuri and Jorge Arturo Cobb. Enforcing ibgp convergence. In *Proc. of the 12th IEEE International Conference on Networks*, pages 511–517, 2004.

[MFM+06] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Building an AS-Topology Model that Captures Route Diversity. In *Proc. SIGCOMM*, 2006.

[Mor] Richard Mortier. PyRT. `research.sprintlabs.com/pyrt/`.

[MYC08] Jianning Mai, Lihua Yuan, and Chen-Nee Chuah. Detecting BGP anomalies with wavelet. In *Proc. NOMS*, 2008.

[Ore] Oregon RouteViews Project. http://www.routeviews.org.

[PZW+02] Dan Pei, Xiaoliang Zhao, Lan Wang, Daniel Massey, Allison Mankin, S. Felix Wu, and Lixia Zhang. Improving BGP convergence through consistency assertions. In *Proc. INFOCOM 2002*, volume 2, pages 902–911, 2002.

[QN04] Xiaohu Qie and Sanjai Narain. Using service grammar to diagnose BGP configuration errors. *Science of Computer Programming*, 53(2):125–141, 2004.

[QU05] Bruno Quoitin and Steve Uhlig. Modeling the routing of an autonomous system with C-BGP. *IEEE Network*, 19(6), 2005.

[RGM+04] Matthew Roughan, Tim Griffin, Z. Morley Mao, Albert Greenberg, and Brian Freeman. IP forwarding anomalies and improving their detection using multiple data sources. In *Proc. SIGCOMM workshop on Network troubleshooting*, 2004.

[RIP] RIPE Routing Information Service (RIS). http://www.ripe.net/ris.

[RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[RS06] Anuj Rawat and Mark A. Shayman. Preventing persistent oscillations and loops in IBGP configuration with route reflection. *Computer Networks*, 50(18):3642–3665, Dec 2006.

[SFS] J. Scudder, R. Fernando, and S. Stuart. BGP monitoring protocol. Internet-Draft, draft-ietf-grow-bmp-02.txt, 2009.

[Sob05] Joao Luís Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Trans. on Networking*, 13(5):1160–1173, 2005.

[SSZ09] Rahul Sami, Michael Schapira, and Aviv Zohar. Searching for stability in interdomain routing. In *Proc. INFOCOM 2009*, 2009.

[Tan06] Sebastien Tandel. BGP Converter - AS-wide conversion for C-BGP. http://alumni.info.ucl.ac.be/standel/bgp-converter/, 2006.

[TG05] T. Griffin and G. Huston. BGP Wedgies. RFC 4264, Nov 2005.

[VGE00] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 32(1):1–16, Jan 2000.

[WMW+06] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end Internet performance. In *Proc. SIGCOMM*, 2006.

[WSR09] Yi Wang, Michael Schapira, and Jennifer Rexford. Neighbor-Specific BGP: More flexible routing policies while improving global stability. In *Proc. SIGMETRICS 2009*, 2009. To appear.

[ZAL04] Hongwei Zhang, Anish Arora, and Zhijun Liu. A stability-oriented approach to improving BGP convergence. In *Proc. IEEE Intl. Symposium on Reliable Distributed Systems 2004*, pages 90–99, 2004.