Governing Routing in the Evolving Internet

Stefano Vissicchio

Advisor: Prof. Maurizio Pizzonia
PhD Coordinator: Prof. Stefano Panzieri
Governing Routing in the Evolving Internet

A thesis presented by
Stefano Vissicchio
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

Spring 2012
Committee:
Prof. Maurizio Pizzonia

Reviewers:
Prof. Tim Griffin
Prof. Steve Uhlig
3.3 No-DR, Safety Under Filtering and Robustness are Computationally Hard ........................................ 58
3.4 Restricting to Local Transit Policies ............................................................. 61
3.5 Achieving Efficient Checking ...................................................................... 68
3.6 Related Work .............................................................................................. 71
3.7 Conclusions ............................................................................................... 72

4 A Heuristic-Based Approach to Stability Testing 75
4.1 Introduction .................................................................................................. 75
4.2 A New Greedy Algorithm .......................................................................... 76
4.3 An Automated Convergence Checker ......................................................... 86
4.4 Walking through the Operation of the Convergence Checker .................. 90
4.5 Experimental Results and Applicability Considerations .......................... 98
4.6 Related Work .............................................................................................. 103
4.7 Conclusions ............................................................................................... 105

III Monitoring 107

5 Control-Plane Monitoring 109
5.1 Introduction .................................................................................................. 109
5.2 Motivating Scenarios .................................................................................. 111
5.3 Requirements for a BGP Monitor ............................................................... 113
5.4 Proposed Architecture and Implementation ............................................. 114
5.5 Evaluation ................................................................................................... 117
5.6 Related Work .............................................................................................. 125
5.7 Conclusions ............................................................................................... 129

6 Leveraging Router Programmability for Traffic Matrix Computation 131
6.1 Introduction .................................................................................................. 131
6.2 State of the Art ............................................................................................ 132
6.3 Leveraging Router Programmability ......................................................... 134
6.4 Implementation ............................................................................................ 137
6.5 Evaluation .................................................................................................... 140
6.6 Conclusions ............................................................................................... 142
CONTENTS

IV Evolving Routing 

7 Enabling Network-Wide IGP Reconfigurations 
7.1 Introduction ........................................... 145 
7.2 An Abstract Model for Link State IGPs ............... 147 
7.3 The IGP Migration Problem ............................ 149 
7.4 Methodology .......................................... 154 
7.5 Loop-Free Migrations .................................. 156 
7.6 The Provisioning System ............................... 165 
7.7 Evaluation ............................................ 167 
7.8 Dealing with Network Failures ......................... 173 
7.9 Design Guidelines ..................................... 176 
7.10 Related Work .......................................... 177 
7.11 Limitations and Future Work ......................... 178 
7.12 Conclusions ......................................... 180 

8 Towards an Optimized iBGP Configuration 
8.1 Introduction ............................................ 181 
8.2 A Model for iBGP Configuration ....................... 183 
8.3 Proposal 1: Adding Spurious iBGP Sessions .......... 185 
8.4 The Impact of iBGP Spurious Sessions ................ 189 
8.5 Checking Dissemination Correctness is Hard .......... 192 
8.6 Proposal 2: iBGP Attribute Changing ................. 196 
8.7 More Flexibility Implies More Instability ............ 201 
8.8 Design Guidelines ..................................... 205 
8.9 Revisiting the State of the Art ......................... 213 
8.10 Conclusions ......................................... 215 

9 Seamless BGP Reconfigurations 
9.1 Introduction ............................................ 217 
9.2 Seamless BGP Reconfigurations ....................... 219 
9.3 Current Solutions do not Work ....................... 224 
9.4 An Algorithmic Approach is not Viable ............... 231 
9.5 Problem Complexity .................................. 239 
9.6 A General Solution for BGP Migrations ............... 248 
9.7 Case Study ............................................ 252 
9.8 Related work ......................................... 254 
9.9 Conclusions ......................................... 255
Conclusions and Bibliography 257

Conclusions and Open Problems 259

Publications 263

Bibliography 265
Preamble

The Internet plays a crucial role in today’s information society, making huge amounts of data available to millions of worldwide users. Started as an academic experiment, in less than a decade, the Internet has indeed become a driving force for economy, pushing the information revolution and changing everybody’s life.

From a technical perspective, the Internet is a global network of networks, which interconnects different Autonomous Systems (ASes). Each AS typically corresponds to a different administrative entity. In turn, the network of each AS can contain hundreds of heterogeneous and geographically-distributed network devices. Several network protocols are used to regulate the communication between devices, in order to deliver data packets and to offer higher-level services to the Internet end users. Moreover, to achieve better performance and optimize resources, protocol-specific parameters are typically fine-tuned, via obscure and ambiguous device configuration languages.

Consequently, as the size of the network, the number of deployed protocols and the complexity of device configurations grow, the behavior of each AS network becomes hard to predict, transforming network management into a nightmare. Even worse, management issues are amplified by the frequent need for configuration changes dictated by evolving requirements, new opportunities for resource optimization, and availability of new technologies. Because of stringent Service Level Agreements (SLAs), configuration changes have to be performed with no impact on both offered services and already implemented requirements. Best practices and management methodologies (e.g., [Opp04, Tea07]) suggest that each configuration modification undergo a careful design process, encompassing accurate requirement identification, logical and physical design, and pre-deployment testing. Accommodation of both already implemented and new requirement should be guaranteed as a result of those phases. Also, in order to not disrupt offered services and not violate the
SLAs, the deployment of new configurations on the production network should cause no packet loss. Finally, the global behavior of the network should be conveniently monitored during operation to have the possibility to promptly react to performance problems and network faults. A sketchy representation of a typical network lifecycle, along with main management activities required in different phases, is depicted in Fig. 0.1.

Due to the complexity of today’s networks, careful network design, configuration generation and deployment, and network monitoring are all challenging tasks, each posing its own set of problems. Lots of research and industrial efforts have been devoted to support network administrators in network and configuration management. Unfortunately, recent studies (e.g., [jwp08]) report human errors to still be the main cause for network downtime, which are, in turn, responsible for significant economical losses.

In this thesis, we study how to govern routing in today’s Internet. The routing problem consists in finding a path (optimal according to some criteria) on which to forward data packets sent by a given source and targeted to a given destination. Network devices, i.e., routers, exchange information to reach common routing decisions through routing protocols. However, configuring routing is a much harder problem than it seems at a first glance. Indeed, routing is a distributed problem by nature, mandating router configurations to be always
consistent network-wide. However, network administrators must specify router configuration through vendor-dependent low-level languages, which makes all the configuration generation process hard to automate and error-prone. Even worse, routing decisions can depend on contingent factors, like message timing. Such a feature makes simulations not always suitable for verification purposes, nor static analysis always viable. Moreover, several routing protocols are typically deployed in the same network for different purposes. In particular, in a classic AS network, Interior Gateway Protocols (IGPs) and the Border Gateway Protocol (BGP) provide reachability to intra and inter-domain destinations respectively. However, different routing protocols can subtly interact, giving rise to unexpected side effects (e.g., [GW02b]). Also, some protocol-specific features, like BGP information-hiding and support for routing policies, must be taken into account in order to avoid inconsistent routing and forwarding decisions [GW99]. Hence, it is not surprising that router configuration errors are pervasive [MWA02, FB05], threatening user traffic delivery and Internet connectivity disruption.

In this thesis, we propose novel research contributions on routing problems that affect different management tasks. We follow the typical network lifecycle, from its initial deployment to operation and reconfiguration, focusing on three common management activities: pre-deployment configuration testing, network monitoring, and replacement of configurations in a running network. After introducing some background in Part I, we tackle in Part II the problem of statically checking a given network configuration for correctness. Such a static check is meant to allow router configurations to be validated before their deployment in a production network. In particular, we delve into static testing of BGP configurations for dynamic stability, that is, guaranteed convergence to a stable routing decision. After formally presenting different kinds of stability problems in Chapter 2, we prove that statically assessing BGP stability is computationally hard in Chapter 3. We adopt a more practical perspective in Chapter 4, where we propose a heuristic for solving the BGP stability checking problem, and we describe a convergence checker tool based on that heuristic.

In Part III, we propose advanced techniques to improve network monitoring. We consider protocols monitoring in Chapter 5, where we compare different approaches, and we describe a new solution based on the highly-optimized packet cloning feature available on many commercial routers. In the following Chapter 6, we explore possibilities opened by router programmability, especially focusing on data-plane monitoring and traffic matrices.

In Part IV, we study how to evolve network-wide routing configuration with no impact on traffic. Our vision is that network-wide reconfigurations should
become a network primitive in the Future Internet, enabling lossless replacement or configuration fine-tuning of any network protocol without affecting offered services. In Chapter 7, we analyze IGP reconfigurations, focusing on the commonly deployed link-state IGPs. We formalize the problem of finding an operational ordering which guarantee no packet loss, we prove its complexity, and we propose algorithms and heuristics for solving it. Based on those algorithms, we build a general methodology and a prototype system able to automatically perform lossless reconfigurations. After, we turn to BGP reconfigurations. In Chapter 8, we discuss how to improve BGP configurations, by analyzing design proposals targeted to add flexibility. Regarding those proposals, we point out their risks of raising unexpected detrimental side effects, and we propose design guidelines to fix them. In Chapter 9, we tackle the problem of how to reconfigure BGP with no impact on both routing and forwarding. After formalizing the problem, we show that an algorithmic approach to find an operational ordering is not viable in the general case. Hence, we discuss a solution which leverages Virtual Routing and Forwarding (VRF) in order to run two BGP routing processes at the same time.

Finally, in the last part of this thesis, we draw conclusions, and we discuss interesting directions for future research.
Part I

Background
Chapter 1

Routing in the Internet

The Internet provides connectivity among thousands of Autonomous Systems (ASes), i.e., domains typically belonging to different administrative entities.

According to their role, ASes are typically partitioned in two main categories: stub ASes and Internet Service Providers (ISPs), also called transit ASes. Stub ASes are exclusively interested in accessing or providing content over the Internet. To this aim, however, they need to connect to ISPs. ISPs, indeed, build the infrastructure for delivering packets from any source to any destination in the Internet. The primary service they provide is connectivity among ASes, that is, ISPs sell their ability of transiting packets to a remote destination as a service. ISPs can, in turn, purchase and sell connectivity services from and to other ISPs. In general, when an AS X provides connectivity to another AS Y, the commercial relationship between the two ASes is classified as customer-provider. Also, X is said to be a provider of Y, while X is called customer of Y.

The customer-provider relationship builds an AS hierarchy consisting of many levels, or tiers. Stub ASes are the bottom tier, while ISPs that provide connectivity only to stub ASes are at the penultimate tier, and so on. The top layer, normally referred to as Tier-1, includes huge geographically distributed ISPs, which own (possibly aggregated) information about all the destination prefixes in the Internet. For sure, such an AS hierarchy represents a simplified model of commercial relationships between ASes, as real-world commercial relationships can be much more complicated [RWM+11]. For example, ISPs can (even locally) exchange (some) traffic free of charge, establishing so-called peer-to-peer relationships [Gao01]. Those relationships are far from being an
exception. In particular, recent trend is to commonly establish peer-to-peer relationships with big Internet players like Content Providers (like Google and Facebook) and Content Delivery Networks (like Akamai and Limelight) which attracts huge amount of user traffic [LIJM+10]. This allows to keep latency as low as possible and increase the quality of end-user experience for popular destinations.

In this thesis, we often take the perspective of a single ISP. Topologically, the network of an ISP typically encompasses a backbone and several Point of Presences, that can also be geographically distributed all over the world. A Point of Presence (PoP) is a location where several border routers, also called egress points, exchange traffic and interdomain routing information with routers of other ASes. The backbone connects the PoPs together. In order to route traffic, at least two kinds of routing protocols are deployed in a basic ISP configuration: an Interior Gateway Protocol (IGP) for intra-domain routing, and the Border Gateway Protocol (BGP) for inter-domain routing. The IGP has to be run on all the routers in the ISP, and provide routing information useful within a single AS. In addition, border routers are also required to speak BGP in order to get information about remote destinations (i.e., inside other ASes). Fig. 1.1 shows a high-level view of a typical ISP network topology, providing indications of the scope of different routing protocols.

In the following, we briefly recall the most important features of the routing protocols (link-state IGPs and BGP) we mainly refer to in this thesis.
1.1 Interior Gateway Protocols

Among all network routing protocols, Interior Gateway Protocols (IGPs) play a critical role. An IGP enables end-to-end reachability between any pair of routers within the network of a single AS. Many other routing protocols, like BGP, LDP or PIM, rely on an IGP to work.

IGPs are roughly divided in two main families: distance-vector and link-state protocols. Distance-vector IGPs are based on the Bellman-Ford algorithm. In these protocols, each router takes routing (and forwarding) decisions based on the information propagated by its neighbors. On the contrary, in a link-state IGP, every router is aware of the entire network topology. Typically, link-state IGPs rely on the Dijkstra algorithm, and guarantee faster convergence. Although some enterprise networks still use distance-vector protocols, most ISPs and large enterprises deploy link-state IGPs, namely OSPF [Moy98] or IS-IS [Ora90]. Hence, we mainly focus on link-state IGPs in this thesis.

Link-state IGPs can be configured either in a flat or in a hierarchical mode. In flat IGPs, every router is aware of the entire network topology and forwards IP packets according to the shortest paths towards their respective destinations. In hierarchical IGPs, routers are not guaranteed to always prefer the shortest paths. Hierarchical IGP configurations break the whole topology into a set of zones (called areas in OSPF and levels in IS-IS), which we denote as $B, Z_1, \ldots, Z_k$. $B$ is a special zone, called backbone, that connects all the other peripheral zones together, such that packets from a router in the network to a destination inside a different zone always traverse the backbone. IGP routers establish adjacencies over physical links, in order to exchange routing information. Each adjacency belongs to only one zone. By extension, we say that a router is in a zone if it has at least one adjacency in that zone. We call internal routers the routers that are in one zone only. The Zone Border Routers (ZBRs) (e.g., ABRs in OSPF and L1L2 systems in IS-IS) are the routers that are in more than one zone, among which one must be the backbone. Both internal routers and ZBRs prefer intra-zone over inter-zone paths. This means that, to choose the path on which to forward packets towards a certain destination, each router prefers a path traversing only one zone over a path traversing more than one zone, no matter what is the length of the two paths.

Moreover, in hierarchical IGPs, ZBRs can be configured to perform route summarization. In this configuration, ZBRs hide the internal topology of a zone $Z$ to routers in different zones, advertising aggregated prefixes outside $Z$. In practice, they announce their ability to reach groups of destinations with paths of a certain length. The length announced by a ZBR is the same for all
the destinations in an aggregated prefix, and either it is customly configured or
it is decided on the basis of the actual lengths of the preferred paths towards
that destinations (e.g., picking the highest one [Moy98]).

1.2 The Border Gateway Protocol

Different ASes exchange routing information via the Border Gateway Protocol
(BGP) [RLH06]. For inter-domain traffic, BGP has the final say on routing and
forwarding decisions, and routing information exchanged via BGP can have a
dramatic impact on the quality of service actually provided by an ISP to its
customers. We now briefly recall how BGP works.

BGP routing information, i.e., routes, are exchanged on transport connec-
tions called BGP sessions, or peerings. Two routers are called BGP peers,
or simply peers, if a BGP session exists between them. On those sessions,
BGP speaking routers (which we will also call BGP routers or BGP speakers
throughout the present thesis) exchange routes to IP destination prefixes using
BGP messages. BGP messages provide information about reachability given
destination prefixes, and associate each route to a set of attributes. RFC 4271
defines the following attributes.

- **AS-path** (well-known, mandatory, transitive): it is the sequence of ASes
  traversed by BGP message.

- **origin** (well-known, mandatory, transitive): it signals how the prefix has
  been injected into BGP.

- **next-hop** (well known, mandatory, transitive): it contains the IP address
  of the BGP next-hop, that is, the BGP speaking 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
  routes sent by the same AS.

- **local-preference** (well known, non-transitive): it 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 allows aggregation of contiguous IP prefixes that share the same at-
  tribute.
1.2. THE BORDER GATEWAY PROTOCOL

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

- **community** (optional, discretionary, transitive): this attribute does not have any defined semantics. It is basically a way to associate a set of tags (each tag consists of a pair of integer values) to a route. It is especially useful to add information to the route, and set the **local-preference**. For example, many ISPs allow their customers to set specific community values for traffic engineering purposes.

Since the **as-path** attribute is carried inside each BGP message, BGP is said to be a **path-vector** protocol.

BGP works on a per-prefix basis. From a very abstract point of view, for each prefix, the behavior of a BGP router mainly consists of three phases. In the first phase, the router collects routing information from neighboring BGP routers, and possibly modifies the BGP message by editing some of its attributes (**input policy**). Then, it selects its best route. Finally, the router creates outgoing BGP messages by possibly changing BGP attributes in the received message (**output policy**), and selectively announces its best route to its peers through such messages. This process is locally repeated at each router in the network each time it receives a new route (or a withdrawal of a previously received route).

The best route is selected by running the deterministic **BGP decision process**, summarized in Table 1.1. The BGP decision process consists of a set of rules. Whenever there are ties for a rule, the next rule is applied to break the tie. From a high-level point of view, a route for a destination prefix is selected as the best based on the values of the attributes associated to it. To ensure the BGP decision process to be deterministic, its last steps evaluate the identifier (**router-id**) and the IP address (**peer-id**) of the BGP neighbor from which the route is learned, respectively. We refer the reader to [ZB03] for a detailed description of the BGP decision process.

Observe that BGP configuration languages allow operators to modify the attributes carried by a message in order to influence the best route selection 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). Fine-tuning of the BGP decision process outcome through attribute setting allow operators to deploy high-level **routing policies**, that is, preference of a route over another. Indeed, BGP is commonly referred to as a path-vector **policy-based** routing protocol.
CHAPTER 1. ROUTING IN THE INTERNET

<table>
<thead>
<tr>
<th>Step</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prefer routes with higher <strong>local-preference</strong></td>
</tr>
<tr>
<td>2</td>
<td>Prefer routes with lower <strong>as-path</strong> length</td>
</tr>
<tr>
<td>3</td>
<td>Prefer routes with lower <strong>origin</strong></td>
</tr>
<tr>
<td>4</td>
<td>Among the routes received from the same AS neighbor, prefer those having lower <strong>MED</strong></td>
</tr>
<tr>
<td>5</td>
<td>Prefer routes learned via eBGP</td>
</tr>
<tr>
<td>6</td>
<td>Prefer routes with lower IGP distance to the <strong>next-hop</strong></td>
</tr>
<tr>
<td>7</td>
<td>Prefer routes having the lowest <strong>router-id</strong></td>
</tr>
<tr>
<td>8</td>
<td>Prefer the route having the lowest <strong>peer-id</strong></td>
</tr>
</tbody>
</table>

Table 1.1: BGP decision process.

BGP has two different modes of operations: external BGP (eBGP), which is used between different ASes, and internal BGP (iBGP), which is used within a single AS to distribute externally learned BGP routes. We separately describe the most important features of eBGP and iBGP in the following.

**eBGP**

One of the key features of BGP is to allow network operators to control inter-domain routing at a fine-grained level, by specifying routing policies. Routing policies are mainly targeted to reflect commercial relationships between ASes.

As noted in [FJB05, FJB07], BGP policies can be broken down in two components: **ranking** and **filtering**. The roles of the ranking and filtering components of routing policies are clearly distinguished. Ranking allows an AS to specify preferences over multiple candidate routes to the same destination, while filtering allows an AS to selectively accept and advertise specific routes from and to specific neighbors. To further confirm their distinction, the ranking and filtering components of routing policies are often specified in router configurations by using separate statements. Fig. 1.2 shows a sample fragment of a typical router configuration, highlighting the separation between filtering and ranking components of routing policies. In this configuration, the filtering component is such as to accept all the BGP announcements but those that traversed AS 31337. The ranking component applies a higher preference to BGP announcements that have been originated by AS 1.

BGP provides only few restrictions about routing policies that can be set. In particular, BGP provides ASes with the **autonomy** to set routing policies independently of each other, and with the **expressiveness** to specify extremely
1.2. THE BORDER GATEWAY PROTOCOL

### Figure 1.2: Ranking and filtering components in a sample router configuration.

```
ip as-path access-list 1 deny _31337_  
ip as-path access-list 1 permit .*  
route-map fooFilter permit 10  
  match as-path 1

ip as-path access-list 2 permit _1$  
route-map fooRank permit 10  
  match as-path 2  
  set local-preference 120  
route-map fooRank permit 20  
  ! other paths have default pref.
```

← route filtering

← route ranking

complex configurations. Unfortunately, autonomy and expressiveness come at the cost of possibly setting conflicting route preferences between ASes. Such conflicts can negatively affect BGP convergence, that is, BGP speakers are not guaranteed to ever achieve a common routing decision in given configurations [VGE00]. We dig into BGP convergence issues and we study the impact of autonomy and expressiveness in Part II of this thesis.

**iBGP**

The basic role of iBGP is to provide consistency among inter-domain routing information received by BGP speakers of the same AS.

Vanilla iBGP did not allow an iBGP router to relay messages to other routers, hence a full mesh of iBGP sessions was needed to ensure correct route distribution. This implies the number of iBGP sessions to be quadratic with respect to the number of routers, hence affecting resource consumption at routers and manageability of device configurations. As the network grows, a mechanism to scale the number of iBGP sessions is needed. Two mechanisms were proposed to allow iBGP topologies to scale: route reflection [BCC06] and BGP confederations [TMS07]. Route reflection modifies iBGP propagation rules and allows specific routers, the route-reflectors, to reflect routes to a set of clients. BGP confederations divide an AS into several fully-meshed or RRMed components. Special eBGP sessions are then used between components to propagate paths to other components. In this thesis, we focus on route reflection as it is the most widely adopted mechanism.

When route reflection is used, the iBGP neighbors of each router are split into three sets: *clients*, *peers* and *route-reflectors*. In the following, we refer
CHAPTER 1. ROUTING IN THE INTERNET

<table>
<thead>
<tr>
<th>Route learned from</th>
<th>Reflect to clients</th>
<th>Reflect to non-clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>eBGP</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>client</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>non-client</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 1.2: Best route propagation rules in iBGP with route reflection.

to a router that has one or more clients as a route-reflector. Also, we refer to a session between iBGP peers as an OVER session. We denote a session between a client and a route reflector as an UP session if it is traversed from the client to the route-reflector, and as a DOWN session otherwise. A fully meshed iBGP network corresponds to a degenerated route reflection topology in which all iBGP routers are peers. However, organizing routers in a hierarchy of clients and route-reflectors allows the iBGP topology to scale, since fewer sessions are needed and clients learn most routing information from their route-reflectors. Indeed, each iBGP speaker propagates its best route according to the rules depicted in Table 1.2: 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. 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.

Practically, iBGP routers are organized in clusters. A cluster consists of one or more route-reflectors and all their clients. Whenever not explicitly stated, we assume that every cluster has a single route-reflector. Each cluster is identified through a unique cluster-id. Messages carry a cluster-list attribute, which accounts for the iBGP path and is used to avoid control-plane loops. The length of the cluster-list is also considered in the BGP decision process. Indeed, route reflection prescribes to change the last steps of the BGP decision process as shown in Table 1.3. Throughout the thesis, we denote the routers that receive an eBGP route for a given prefix as egress points for that prefix. The egress-id of a route is the router-id of the egress point that announces that route.

Observe that in the iBGP decision process a fundamental role is assumed by the IGP metrics, which are used to locally discriminate among routes equally preferred according to eBGP policies. From a theoretical point of view, tweaking the IGP metrics allows network operators to specify intradomain routing policies. However, expressiveness of iBGP policies is more limited with respect to eBGP, because of both route propagation constraints (see Table 1.2) and
### 1.2. THE BORDER GATEWAY PROTOCOL

<table>
<thead>
<tr>
<th>Step</th>
<th>Criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prefer routes with higher <code>local-preference</code></td>
</tr>
<tr>
<td>2</td>
<td>Prefer routes with lower <code>as-path</code> length</td>
</tr>
<tr>
<td>3</td>
<td>Prefer routes with lower <code>origin</code></td>
</tr>
<tr>
<td>4</td>
<td>Among the routes received from the same AS neighbor, prefer those having lower <code>MED</code></td>
</tr>
<tr>
<td>5</td>
<td>Prefer routes learned via eBGP</td>
</tr>
<tr>
<td>6</td>
<td>Prefer routes with lower IGP distance to the <code>next-hop</code></td>
</tr>
<tr>
<td>7</td>
<td>Prefer routes having the lowest <code>egress-id</code></td>
</tr>
<tr>
<td>8</td>
<td>Prefer routes with shorter <code>cluster-list</code></td>
</tr>
<tr>
<td>9</td>
<td>Prefer the route having the lowest <code>peer-id</code></td>
</tr>
</tbody>
</table>

Table 1.3: BGP decision process.

...possibility to rank routes only on the basis of the iBGP egress point. However, route preference conflicts are also possible in iBGP, and routing convergence is not guaranteed [GW02b], because of reduced route visibility (i.e., clients rely on the information mirrored by their route-reflectors) traded for more scalability in route reflection. Even worse, the coupling between BGP and the underlying IGP can result in forwarding anomalies [GW02b], e.g., user packets indefinitely bounced among a limited number of routers creating a forwarding loop. We devote special attention to iBGP routing and forwarding anomalies throughout this thesis, especially in Parts II and IV.
Part II

Checking Routing Configuration for Guaranteed Convergence
Chapter 2

A Model for BGP Stability Problems *

2.1 Introduction

BGP (in both the external and the internal flavors) is renowned [GW99] to be prone to routing oscillations. Indeed, some BGP configurations can never converge to a stable routing state, either because a stable state for that configuration does not exist at all, or because the protocol gets trapped into bad event timings. Routing oscillations can and do [Ber01, Car06] happen due to BGP policy disputes, that can be due to different factors like conflicting local-preference settings in eBGP [GW99] or IGP metrics in iBGP [GW02b].

In the following, we will refer to guaranteed convergence as stability.

Because of the central role that BGP has in the Internet routing, guaranteeing its convergence to a stable routing state, however, is highly desirable for an ISP. Indeed, it has been observed that interdomain routing changes can cause performance degradation and packet losses [WMW+06], and can

severely affect the availability of services running over the Internet infrastructure [KKK07]. Moreover, in case of oscillations, precious router resources are wasted in periodically reprocessing the same messages, and bandwidth is consumed by unnecessary routing updates travelling back and forth.

In this chapter, we introduce BGP stability problems we wish to statically check for with static pre-deployment tests (i.e., before configurations are deployed, possibly giving raise to hard-to-detect convergence issues). In Section 2.2, we present the commonly-agreed SPP and SPVP models, which we adopt to study BGP stability in the thesis. For the sake of simplicity, throughout the thesis, we exclude from our analysis the MED attribute and the routing instabilities caused by MED values [GW02a]. However, most of our studies and findings can be extended to deal with MED, since known techniques can be adopted to model MED behavior in the SPP and SPVP models [GW02a]. Then, we formally define the most important decision problems related to BGP stability in Section 2.3. In particular, we focus on safety, suf, and robustness problems. Intuitively, the safety problem consists in determining if a BGP configuration is guaranteed to converge to a stable state, whatever the message timing is; safety under filtering (suf) and robustness problems are two natural extensions of safety dealing with the possibility of autonomously configuring route filters on each BGP speaker and with link/router failures, respectively. In the same section, we also review known sufficient conditions and known necessary conditions for the considered BGP stability problems. In particular, we recall the definition of dispute wheel and dispute ring.

In Section 2.4, we extend the state of the art by defining the no-dr condition and proving that it is a necessary and sufficient condition for safety under filtering. Our result is based on the presence of a static structure called dispute reel (DR), which is both a special case of a dispute wheel and a generalization of a dispute ring. Being static structures, dispute reels inherit from the dispute wheels 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 the BGP protocol dynamics. To the best of our knowledge, this is the first characterization of stability in policy routing. Moreover, the no-dr condition fills the large gap between previously known sufficient conditions and necessary conditions. As a side effect, dispute reels can replace dispute wheels in several results on different BGP stability problems.

In Section 2.5, we clarify the inclusion relationships between BGP stability problems. In particular, we show that, in a network admitting multiple stable routing states, safety under filtering is provably compromised. Thus, the presence of the so-called BGP wedgies [TG05] is a sufficient condition for
making a network unsafe under filtering. Moreover, we prove that robustness does not necessarily imply safety under filtering. This means that, in a sense, the autonomy of adding (possibly misconfigured) filters can do more harm than network faults.

Finally, related work is discussed in Section 2.6, and conclusions are drawn in Section 2.7.

2.2 Modeling BGP

In this section, we describe the well-known Stable Paths Problem (SPP) and Simple Path Vector Protocol (SPVP) formalisms introduced in [GSW02] and in [GSW09], respectively. Terminology and notation used in this section is adopted in the rest of the thesis.

BGP Preferences: SPP

In order to vehiculate and exchange routing information (e.g., IP prefix reachability), BGP routers must establish adjacencies. Abstractly, those adjacencies build a logical graph. SPP models a BGP network as an undirected graph \( G = (V,E) \). \( V = \{0,1,\ldots,n\} \) is a set of vertices representing BGP speakers, typically ASes in eBGP or routers in iBGP. Edges in \( E \) correspond to peerings between BGP speakers. Vertex 0 is special since it is the destination every other vertex attempts to establish a path to. Since different destinations are independently handled by BGP [RLH06], 0 is assumed, without loss of generality, to be the only destination. In the following, we often use paths on the logical graph \( G \) to characterize the BGP routes that are propagated on that paths. More formally, a path \( P \) is a sequence of \( k+1 \) vertices \( P = (v_k v_{k-1} \ldots v_1 v_0) \), \( v_i \in V \), such that \( (v_i,v_{i-1}) \in E \) for \( i = 1,\ldots,k \). Vertex \( v_{k-1} \) is the next hop of \( v_k \) in \( P \). For \( k = 0 \) we obtain the trivial path \( (v_0) \) consisting of vertex \( v_0 \) alone.

The empty path represents inability to reach the destination and is denoted by \( \epsilon \). The concatenation of two nonempty paths \( P = (v_k v_{k-1} \ldots v_i) \), \( k \geq i \), and \( Q = (v_i v_{i-1} \ldots v_0) \), \( i \geq 0 \) is path \( PQ = (v_k v_{k-1} \ldots v_i v_{i-1} \ldots v_0) \). We assume that \( P\epsilon = \epsilon P = \epsilon \), that is, the empty path can never extend or be extended by other paths.

To represent the outcome of the BGP decision process, including input and output policies, SPP introduces the concepts of permitted paths and ranking function. SPP models the effects of both the ranking and the filtering components in BGP route processing by explicitly listing all routes that are not filtered out and defining their respective preference. More precisely, each
vertex \( u \in V \) is assigned with a set of permitted paths \( \mathcal{P}^u \) representing the paths that \( u \) can use to reach 0. All the paths in \( \mathcal{P}^u \) are simple (i.e., without repeated vertices), start from \( u \) and end in 0. The empty path, representing unreachability of 0, is permitted at each vertex \( u \neq 0 \). Vertex 0 can reach itself only directly, hence \( \mathcal{P}^0 = \{ (0) \} \). Let \( \mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u \) the set of permitted paths on all the vertices. For each \( u \in V \), a ranking function \( \lambda^u : \mathcal{P}^u \to \mathbb{N} \) determines the preference level \( \lambda^u(P) \) assigned by \( u \) to path \( P \). If \( P_1, P_2 \in \mathcal{P}^u \) and \( \lambda^u(P_2) < \lambda^u(P_1) \), then \( P_2 \) is preferred over \( P_1 \). Let \( \Lambda = \{ \lambda^u | u \in V \} \).

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

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

(ii) \( \forall P_1, P_2 \in \mathcal{P}^u, P_1 \neq P_2 : \lambda^u(P_1) = \lambda^u(P_2) \Rightarrow P_1 = (u \ x)P'_1, P_2 = (u \ x)P'_2 \) (strict ranking is assumed on all paths but those with the same next hop).

An instance of SPP is a triple \( S = (G, \mathcal{P}, \Lambda) \), where \( G = (V, E) \) is a simple undirected graph, \( \mathcal{P} \) is the set of permitted paths, and \( \Lambda \) is the set of ranking functions. Fig. 2.1 shows an instance of SPP, called Disagree [GSW02]. The graphical convention we use in this figure will be adopted for depicting SPP instances throughout the thesis. Namely, each vertex \( v \) is equipped with a list of paths representing \( \mathcal{P}^v \), sorted by decreasing values of \( \lambda^v \) (i.e., the higher in the list the more preferred). The empty path and \( \mathcal{P}^0 \) are normally omitted for brevity. In the Disagree instance, in particular, path preferences (according to the BGP decision process) are set so that vertex 1 prefers the path announced by 2 to reach the destination (i.e., vertex 0) and vice versa. We will refer to this kind of conflicting policies as policy dispute.

Observe that the SPP model is general enough to model both eBGP and iBGP. Indeed, path preferences can derive from eBGP policy setting (e.g., local-preference values reflecting commercial agreements), from iBGP attributes (e.g., IGP metrics), or from a combination of the two.
2.2. MODELING BGP

BGP Dynamics: SPVP

The dynamic behavior of BGP is modeled by a distributed asynchronous algorithm known as Simple Path Vector Protocol (SPVP) [GSW99, GW00] running over an SPP instance. The SPVP algorithm is shown in Fig. 2.2. In the SPVP algorithm, vertices exchange messages containing permitted paths in order to establish a path to 0. It is assumed that message exchanges are reliable and edges introduce an arbitrary finite delay. Communication between vertices takes place in a totally asynchronous way.

To describe the SPVP algorithm in Fig. 2.2 we need a few more definitions. Let peers\((v)\) be the set of neighbors of \(v\). Two data structures are used at each vertex \(v\) to represent the information \(v\) is aware of at time \(t\): the path \(\text{rib}_t(v)\) that is used to reach 0 and a table \(\text{rib-in}_t(v \Leftarrow u)\) that stores the latest path received from neighbor \(u \in \text{peers}(v)\). Thus, vertex \(v\) can select a path to 0 among the choices available in

\[
\text{choices}_t(v) = \{(v \ u)P \in \mathcal{P}^v | P = \text{rib-in}_t(v \Leftarrow u)\}
\]

Let \(W\) be a subset of the permitted paths \(\mathcal{P}^v\) at vertex \(v\), such that each path in \(W\) has a distinct next hop. Then the best path at \(v\) in \(W\) is

\[
\text{best}(W, v) = \begin{cases} 
P \in W | P = \arg \min \lambda^v(P) & (W \neq \emptyset) \\
\epsilon & (W = \emptyset)
\end{cases}
\]

and the overall best path \(v\) is aware of at time \(t\) is \(\text{best}_t(v) = \text{best}(\text{choices}_t(v), v)\).

In SPVP, each vertex executes an instance of the algorithm in Fig. 2.2. When a vertex \(v\) receives a path \(P\) from the neighboring vertex \(u\), it stores
\( P \) in the local data structure \( \text{rib-in}_u(v \leftarrow u) \) and recomputes its best path. If the computed best path \( Q \) differs from the previously selected path, \( u \) sends a message containing \( P \) to all of its neighbors.

The order in which announcements are exchanged among vertices is modeled in SPVP by activation sequences [GSW02]. We say that an edge \((u, v)\) is activated at time \( t \) if \( v \) executes the algorithm in Fig. 2.2 to process the latest message received from \( u \) at time \( t \). The order in which protocol messages are exchanged does not need to be total, i.e., at a given time more than one message can be processed. This partial order is represented using activation sequences. An activation sequence \( \sigma \) is a (possibly infinite) sequence of sets \( \sigma = \{A_0, A_1, \ldots, A_i, \ldots\} \), in which each set \( A_i \) contains the edges that are activated at time \( t \). Each edge in \( A_i \) is considered oriented according to the direction of its activation.

We say that an activation sequence is fair [GSW02] if, whenever vertex \( u \) sends a message at time \( t \) (Step 7 of SPVP), there exists a time \( t' > t \) at which the message is delivered and processed by its recipient. This means that edge \((u, v)\) is eventually activated when \( u \) sends a message to \( v \).

To model the BGP decisions made by different vertices at different times, the concept of path assignment is defined. A path assignment is a function \( \pi \) that maps each vertex \( v \in V \) to a path \( \pi(v) \in \mathcal{P}^v \). Observe that, at any time \( t \), the SPVP algorithm defines a path assignment \( \pi_t \) where \( \pi_t(v) = \text{rib}_t(v) \) and each vertex always selects the best available path. We have that \( \forall t \pi_t(0) = (0) \). Also, \( v \neq 0 \) cannot reach vertex 0 at time \( t \) if \( \pi_t(v) = \epsilon \). Given an SPP instance \( S \), we say that an activation sequence \( \sigma \) on \( S \) leads to path assignment \( \pi_{t_{2}} \) starting from path assignment \( \pi_{t_{1}} \), denoted by \( \pi_{t_{1}} \xrightarrow{\sigma} \pi_{t_{2}} \), if, after activating edges according to \( \sigma \), \( S \) changes its state from \( \pi_{t_{1}} \) to \( \pi_{t_{2}} \).

In the following, we will refer to \( \pi \) (or \( \pi_t \)) as a state of the SPP instance (at time \( t \)). A state \( \pi_t \) of an SPP instance is a stable state if \( \forall v \in V: \pi_t(v) = \text{best}_t(v) \). This means that every vertex has settled to the best available path and will never change its selection. For example, two stable states \( \pi_1 \) and \( \pi_2 \) for \textsc{Disagree} are described in right part of Fig. 2.1. In the figure, each row of the table represents a state, and each column specifies the path selected by every vertex in that state.

Finally, we say that an instance \( S \) of SPP is consistent if, for any \( u \in V \) and \( P \in \mathcal{P}^u \), we have that \( P = (u, v)P' \) and \( P' \in \mathcal{P}^v \). We stress that the presence of a path violating this condition cannot affect the behavior of the SPVP algorithm on \( S \). Therefore, we always assume, without loss of generality, that all SPP instances are consistent, unless the contrary is explicitly stated.
2.3. BGP STABILITY PROBLEMS

<table>
<thead>
<tr>
<th>$t$</th>
<th>$A_t$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${(0, 1), (0, 2)}$</td>
<td>(1 0)</td>
<td>(2 0)</td>
</tr>
<tr>
<td>2</td>
<td>${(1, 2), (2, 1)}$</td>
<td>(1 2 0)</td>
<td>(2 1 0)</td>
</tr>
<tr>
<td>3</td>
<td>${(1, 2), (2, 1)}$</td>
<td>(1 0)</td>
<td>(2 0)</td>
</tr>
</tbody>
</table>

Table 2.1: A fair activation sequence for DISAGREE (Fig. 2.1) which can be indefinitely repeated without settle to a stable state.

Observe that variants of the SPVP algorithm where only some classes of activation sequences are allowed, have been proposed in literature (e.g., [GSW99, GR00, BOR+02, FJB07]). For the sake of completeness, we base our study on the original version of the algorithm, in which edges are activated independently and simultaneous activations are allowed. Indeed, it has been shown [CDR08] that any relaxed version of the original SPVP model is only able to capture a strictly smaller set of routing oscillations.

Among BGP stability problems, a basic problem is SOLVABILITY [GSW99], defined as follows.

**Problem 2.1** Given an SPP instance $S$, SOLVABILITY is the problem of determining whether $S$ admits a stable state.
Indeed, there are SPP instances that do not have any stable state. An example of instance with no stable states is Bad-Gadget [GW99], shown in Fig. 2.3. Intuitively, this depends on the fact that all vertices prefer their clockwise neighbor instead of their own direct path to 0, because of a cyclic structure of preferences. An infinite fair activation sequence on Bad-Gadget is described in Table 2.2. The initial state is assumed to be \( \pi_0(v) = \epsilon \) \( \forall v \in V - \{0\} \). After vertices 1 and 2 have succeeded in choosing their preferred path at \( t = 2 \), edges (1, 2), (2, 3), and (3, 1) can be sequentially activated in counter-clockwise order, while never converging to a stable state. Moreover, edges (0, 1), (0, 2), (0, 3), (1, 0), (2, 0), (3, 0), (2, 1), (3, 2), and (1, 3) can be activated after time 8 in order to ensure the fairness of the activation sequence, without changing the state of any vertex in the instance. The activation sequence reported in Table 2.2 from time 3 to time 8 can be repeated indefinitely to obtain an infinite fair activation sequence.

Even if a stable state exists, BGP can still get trapped [GW99, GSW02] into routing oscillations. Indeed, we already showed that the Disagree instance is guaranteed to converge to any of the two stable states it admits (see Fig. 2.1). We then define the safety problem as follows.

**Definition 2.1** An SPP instance \( S \) is safe if any fair activation sequence on \( S \) eventually leads to a stable state, that is, SPVP is guaranteed to converge on \( S \).

**Problem 2.2** Given an SPP instance \( S \), safety is the problem of determining whether \( S \) is safe.

Also, we formally define the concept of safety under filtering [FJB05] and robustness [GSW02], already presented in the introduction of this chapter.
Table 2.2: A snapshot of an infinite fair activation sequence for Bad-Gadget (Fig. 2.3).

<table>
<thead>
<tr>
<th>$t$</th>
<th>$A_t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$(0,1), (0,2), (0,3)$</td>
<td>$(1\ 0)$</td>
<td>$(2\ 0)$</td>
<td>$(3\ 0)$</td>
</tr>
<tr>
<td>2</td>
<td>$(3,1), (1,2)$</td>
<td>$(1\ 3\ 0)$</td>
<td>$(2\ 1\ 0)$</td>
<td>$(3\ 0)$</td>
</tr>
<tr>
<td>3</td>
<td>$(1,2)$</td>
<td>$(1\ 3\ 0)$</td>
<td>$(2\ 0)$</td>
<td>$(3\ 0)$</td>
</tr>
<tr>
<td>4</td>
<td>$(2,3)$</td>
<td>$(1\ 3\ 0)$</td>
<td>$(2\ 0)$</td>
<td>$(3\ 2\ 0)$</td>
</tr>
<tr>
<td>5</td>
<td>$(3,1)$</td>
<td>$(1\ 0)$</td>
<td>$(2\ 0)$</td>
<td>$(3\ 2\ 0)$</td>
</tr>
<tr>
<td>6</td>
<td>$(1,2)$</td>
<td>$(1\ 0)$</td>
<td>$(2\ 1\ 0)$</td>
<td>$(3\ 2\ 0)$</td>
</tr>
<tr>
<td>7</td>
<td>$(2,3)$</td>
<td>$(1\ 0)$</td>
<td>$(2\ 1\ 0)$</td>
<td>$(3\ 0)$</td>
</tr>
<tr>
<td>8</td>
<td>$(3,1)$</td>
<td>$(1\ 3\ 0)$</td>
<td>$(2\ 1\ 0)$</td>
<td>$(3\ 0)$</td>
</tr>
</tbody>
</table>

**Definition 2.2** An SPP instance $S = (G, \mathcal{P}, \Lambda)$ is safe under filtering if, for any $\mathcal{P}' \subseteq \mathcal{P}$, the instance $(G, \mathcal{P}', \Lambda)$ is safe.

**Problem 2.3** Given an SPP instance $S$, **safety under filtering (suf)** is the problem of determining whether $S$ is safe under filtering.

**Definition 2.3** An SPP instance $S = (G = (V,E), \mathcal{P}, \Lambda)$ is robust if the instance $(G' = (V,E'), \mathcal{P}', \Lambda)$ is safe, for any $E' \subseteq E$, and $\mathcal{P}' = \mathcal{P}$ except for the paths removed by the modification of $G$.

**Problem 2.4** Given an SPP instance $S$, **robustness** is the problem of determining whether $S$ is robust.

Intuitively, suf assesses whether a BGP configuration is guaranteed to converge to a stable state after application of any combination filters applied by any BGP speaker in the network. Observe that achieving safety under filtering also guarantees that configuration changes cannot adversely impact the operation of a running network. Moreover, robustness aims at determining whether an SPP instance remains safe after the removal of any subset of the vertices or edges, hence after any combination of network failures. Observe that we restrict to consider link failures only in Definition 2.3. Such a restriction is without
loss of generality, since the removal of a vertex \( v \) from an SPP instance has the same effect of removing all the links incident on \( v \).

**Dispute Wheels and Dispute Rings**

It has been shown in \cite{GSW02, FJB07} that BGP stability problems can be studied by analyzing structural properties of SPP instances, without the need to deal with the details of protocol dynamics. The most known structural properties are based on (the absence of) cyclic dependencies among routing preferences. In the following, we recall the formal definition of those structures, called dispute wheels and dispute rings.

**Definition 2.4** A dispute wheel (DW) \([GSW02]\) \( \Pi = (\vec{U}, \vec{Q}, \vec{R}) \) is a triple consisting of a sequence of vertices \( \vec{U} = (u_0 \ u_1 \ldots u_{k-1}) \) and two sequences of nonempty paths \( \vec{Q} = (Q_0 \ Q_1 \ldots Q_{k-1}) \) and \( \vec{R} = (R_0 \ R_1 \ldots R_{k-1}) \) such that for each \( i = 0, \ldots, k - 1 \) we have:

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

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

(iii) \( R_iQ_{i+1} \in \mathcal{P}^{u_i} \)

(iv) \( \lambda^{u_i}(R_iQ_{i+1}) \leq \lambda^{u_i}(Q_i) \).

We refer to vertices \( u_i \) as pivot vertices, to paths \( Q_i \) as spoke paths, and to paths \( R_i \) as rim paths. Throughout the thesis, we intend subscripts of vertices and paths in a dispute wheel to be interpreted modulo \( k \), where \( k = |\vec{U}| \).

The elements that constitute a dispute wheel are graphically depicted in Fig. 2.4a. Observe that the DISAGREE instance in Fig. 2.1 contains a simple dispute wheel, with two pivot vertices \( \vec{U} = (1 \ 2) \), spoke paths \( \vec{Q} = ((1 \ 0) \ (2 \ 0)) \), and rim paths \( \vec{R} = ((1 \ 2) \ (2 \ 1)) \). Also, the policy dispute in the BAD-GADGET instance in Fig. 2.3 is generated by a dispute wheel having vertices 1, 2, and 3 as pivot vertices, and paths \( ((1 \ 0) \ (2 \ 0) \ (3 \ 0)) \) and \( ((1 \ 3 \ 0) \ (2 \ 1 \ 0) \ (3 \ 2 \ 0)) \) as spoke and rim paths, respectively. In the general case, however, dispute wheels occurring in SPP instances can be much more complex, since spoke and rim paths can arbitrarily cross each other. Fig. 2.4b shows a sketch of a complex dispute wheel where there are intersections between spoke paths \( Q_0 \) and \( Q_1 \), spoke path \( Q_1 \) and rim path \( R_0 \), and rim paths \( R_0 \) and \( R_1 \).

The absence of a dispute wheel, commonly referred to as the **no-dw condition**, has been proved \cite{GSW02, FJB07} to be a sufficient condition for safety,
2.3. **BGP STABILITY PROBLEMS**

![Diagram](image)

**Figure 2.4:** (a) Ideal structure of a dispute wheel. (b) Spoke and rim paths of a dispute wheel can intersect in real SPP instances.

![Diagram](image)

**Figure 2.5:** Fundamental relationships between BGP stability problems and sufficient and necessary conditions before (a) and after (b) our contribution.

safety under filtering, and robustness. We define the NO-DW problem as follows.

**Problem 2.5** Given an SPP instance $S$, the NO-DW problem consists in determining if $S$ contains a dispute wheel.

Also, 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
CHAPTER 2. A MODEL FOR BGP STABILITY PROBLEMS

and such that each vertex appears only once in the wheel. In practice, a dispute ring looks like the dispute wheel depicted in Fig. 2.4a. The dispute wheel in the Bad-Gadget SPP instance is an example of dispute ring. However, the absence of a dispute ring does not guarantee safety, and does not even imply that the SPP instance admits a stable path assignment.

Fig. 2.5a shows how the no-dw and “no dispute ring” conditions relate to solvability, safety, and suf. We stress the large gap between the two conditions, highlighted in the figure by the gray zones with thick border.

2.4 Characterizing Safety Under Filtering

In this section, we introduce a particular type of the dispute wheels, which we call dispute reels. Further, we show that the absence of dispute reels is a sufficient and necessary condition for the suf problem. Figure 2.5b shows how the relationship between BGP stability problems changes as a consequence of this result. Robustness is separately discussed in the following section.

Wheels + Rings = Reels

Intuitively, a dispute reel, or simply 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 other words, out of all the crossings represented in Fig. 2.4b, only those between rim paths ($R_0$ and $R_1$ in the figure) are allowed in a dispute reel. Hence, a dispute reel looks much like a dispute ring (see Fig. 2.4a), except that it can contain arbitrary intersections between rim paths.

In order to formally define the concept of dispute reel, we use $P[v]$ to denote the subpath of $P$ starting at vertex $v$, that is, $P = (u \ldots v)P[v]$. This implies $P[0] = (0)$ for any $P$.

**Definition 2.5** A dispute reel (DR) is a dispute wheel which satisfies the following conditions:

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

(ii) (Spoke paths do not intersect any rim path) – for each $u \not\in \bar{U}$, 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 $Q_i, Q_j \in \bar{Q}$, if $v \in Q_i \cap Q_j$, then $Q_i[v] = Q_j[v]$. 
2.4. CHARACTERIZING SAFETY UNDER FILTERING

Figure 2.6: An SPP instance, showed in [FJB07], which is safe under filtering but contains DWs. However, none of these DWs is a DR.

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 verify that DISAGREE (Fig. 2.1) is an example of a DR. Conversely, the instance in Fig. 2.6, 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. Consider, for example, the DW II in Fig. 2.6 having pivot vertices are $\vec{U} = (1 \ 2 \ 3)$, spoke paths are $\vec{Q} = ((1 \ 0) \ (2 \ 0) \ (3 \ 0))$, and rim paths are $\vec{R} = ((1 \ 3 \ 2) \ (2 \ 1 \ 3) \ (3 \ 2 \ 1))$. II is not a dispute reel since pivot vertex 1 appears in all rim paths, thus violating Condition (i) of Definition 2.5. The instance in Fig. 2.6 also contains DW II’ 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 $R_1 = (2 \ 1)$. II’ 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 Fig. 2.6.

We also define a particular family of simple dispute reels, which we call dispute duo. Intuitively, dispute duos are a generalization of the DW in the DISAGREE gadget.

**Definition 2.6** A dispute duo is a dispute reel such that $|\vec{U}| = 2$ and $R_0 \cap R_1 = \{u_0, u_1\}$.

The simple structure of DRs allows us to identify two classes of activation sequences leading to two “natural” classes of path assignments. Given an SPP instance $S$ containing a DW II, the supporting instance $S[II]$ of II is the minimal SPP instance which contains the vertices, edges and paths of II. Intuitively, $S[II]$ can be obtained from $S$ by filtering all paths but those used in the DW. Observe that, if II is a DR, then in $S[II]$ pivot vertices have exactly two permitted paths, and vertices along the spoke paths (except pivots) have exactly one permitted path.
CHAPTER 2. A MODEL FOR BGP STABILITY PROBLEMS

Let $S$ be an SPP instance containing a DR $\Pi$ and let $S[\Pi]$ be the supporting instance of $\Pi$. The all-spoke path assignment (see Fig. 2.7a) is a path assignment $\pi$ such that $\pi(u) = Q_i[u]$ if $u \in Q_i$, $\pi(u) = \epsilon$ 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 $\sigma_{\text{spoke}}$ leading to an all-spoke path assignment.

Similarly, we define the one-rim path assignment for pivot $u_i$ (see Fig. 2.7b) as a path assignment $\tilde{\pi}_i$ such that:

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

In order to build an activation sequence that leads to $\tilde{\pi}_i$, we can extend $\sigma_{\text{spoke}}$ 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 $R_i$, $\tilde{\pi}(v) = \epsilon$. 

---

**Figure 2.7:** Two special path assignments of a dispute reel. The selected paths are highlighted using solid stroke. Note that in $\tilde{\pi}_i$, $u_i$ is the only vertex in $Q_i$ which is not selecting a subpath of $Q_i$. 

(a) All-spoke path assignment $\pi$. 

(b) One-rim path assignment $\tilde{\pi}_i$. 

2.4. CHARACTERIZING SAFETY UNDER FILTERING

Throughout the thesis, we refer to the absence of dispute reels in an SPP instance as the NO-DR condition. Also, we define the NO-DR problem as follows.

**Problem 2.6** Given an SPP instance $S$, the NO-DR problem consists in determining if $S$ contains a dispute reel.

**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 SPP 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 2.1). Second, we generalize this result by stating that if \( S \) contains a DR consisting of two pivot vertices, then \( S \) is not SUF (Lemma 2.2). Last, we show that if an instance \( S \) contains a DR \( \Pi \), then an oscillation can always be constructed, either by cycling through one-rim path assignments on \( \Pi \) (Lemma 2.3), or by exploiting a different DR consisting of two pivot vertices (Lemma 2.4). 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 SPP instance not safe under filtering. First, we generalize the routing oscillation showed in Tab. 2.1 for DISAGREE to the broader class of dispute duos.

**Lemma 2.1** An SPP instance that contains a dispute duo is not safe under filtering.

**Proof:** Let \( S \) be an SPP instance containing a dispute duo \( \Pi = (\vec{U}, \vec{Q}, \vec{R}) \) and consider \( S[\Pi] \). We now construct a fair activation sequence that induces an oscillation on \( S[\Pi] \). Fig. 2.8 depicts the most relevant path assignments in the activation sequence we build 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 \( \pi \) is clearly not stable, so 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 \( \Pi \) is a DR, we can construct on \( S[\Pi] \) an activation sequence that leads to the all-spoke path assignment \( \pi_{t_1} \) at some time \( t_1 \). The resulting path assignment is shown in Fig. 2.8a. 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 \( u_0 \) (\( u_1 \)), obtaining the path assignment represented in Fig. 2.8b. Formally, let \( R_0 = (v_0, v_1, \ldots, v_k) \), 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,

\[
\sigma_{R_0} = \{(v_k, v_{k-1})\} \{(v_{k-1}, v_{k-2})\} \cdots \{(v_2, v_1)\}.
\]

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

\[
\sigma_{R_1} = \{(w_j, w_{j-1})\} \{(w_{j-1}, w_{j-2})\} \cdots \{(w_2, w_1)\}.
\]
2.4. CHARACTERIZING SAFETY UNDER FILTERING

We activate edges according to $\sigma_{R_0}$, and then according to $\sigma_{R_1}$. 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 $\pi_{t_2}$ (see Fig. 2.8c) 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 $v_1$ ($w_1$), thus reaching the path assignment depicted in Fig. 2.8d, 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 an all-spoke path assignment $\pi_{t_3}$, which implies $\pi_{t_3}(u) = \pi_{t_1}(u)$ for every vertex $u$, as shown in Fig. 2.8a. 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 2.1 can be generalized, as DRs having two pivot vertices always imply the existence of a dispute duo. As an example, consider the instance in Fig. 2.9. 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 $\lambda^Y$, 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 2.2** An SPP instance that contains a dispute reel having exactly 2 pivot vertices is not safe under filtering.

**Proof:** Let $S$ be an SPP instance containing a dispute reel $\Pi = (\vec{U}, \vec{Q}, \vec{R})$, with $|\vec{U}| = 2$. First, we show that the presence of $\Pi$ implies that $S$ contains a
dispute duo $\Pi'$, then we use Lemma 2.1 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 $\Pi$ is a dispute duo and the statement directly follows from Lemma 2.1. 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 $v_j \in \{v_1, \ldots, v_k\}$ be the vertex of $R_1$ that is "farthest away" from $u_1$, 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 $\Pi'$ having $u_0$ and $v_j$ as pivot vertices, or there exists a DR $\Pi''$ consisting of two pivot vertices $v_j$ and $u_1$ and having strictly less intersections between its rim paths than $\Pi$.

Refer to Fig. 2.10. Split $R_1$ and $R_0$ 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, $\Pi'$ 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 2.5 is satisfied. Condition (ii) is also satisfied, since $Q_0 \cap R = Q_0 \cap P = \{u_0\}$ and $Q_1 \cap R = Q_1 \cap P = \emptyset$ are guaranteed by the fact that $\Pi$ 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 $\Pi$ is a DR, and $Q \cap Q_0 = \emptyset$.

(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, $\Pi''$ has strictly less intersections between rim paths than $\Pi$. Observe that $v_j$ appears only in

Figure 2.9: An SPP instance containing a DR consisting of two pivot vertices (1 and 2) and whose rim paths intersect at vertices $X$, $Y$, and $Z$. 
2.4. CHARACTERIZING SAFETY UNDER FILTERING

Figure 2.10: 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 2.2. Different paths are represented using different strokes. In particular, spoke paths $Q_0$ and $Q_1$ are in thicker stroke.

$PQ_0$, $Q$, and $A$, while $u_1$ appears only in $Q_1$, $Q$, and $A$. Hence, Condition (i) of Definition 2.5 is satisfied. Condition (ii) is also satisfied, since $Q_0 \cap Q = Q_0 \cap A = \emptyset$ and $Q_1 \cap Q = Q_1 \cap A = \{u_1\}$ are guaranteed by the fact that $\Pi$ 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 $\Pi$ is a DR, and $P \cap Q_1 = \emptyset$.

Hence, in the first case we find a dispute duo $\Pi'$. In the second case, we find another dispute reel $\Pi''$ having two pivot vertices and having strictly less intersections between rim paths than $\Pi$. By iterating this argument, we eventually end up finding a dispute duo. We then use the result from Lemma 2.1 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 SPP 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 $\Pi = (\vec{U}, \vec{Q}, \vec{R})$, with $|\vec{U}| = k > 2$, we say that $\Pi$ is rim-by-rim if $\forall i \in \{0, \ldots, k - 1\}$ there exists an activation sequence $\sigma_i$ on $S[\Pi]$ such that $\bar{\pi}^i \xRightarrow{\sigma_i} \bar{\pi}^{i+1}$. That is, starting from the one-rim path assignment for any pivot $u_i$, $\sigma_i$ leads to the one-rim path assignment for pivot $u_{i+1}$ (see Fig. 2.7b). The following property is a straightforward consequence of the definition of rim-by-rim DR.

**Property 2.1** $\sigma_i$ activates all the edges in $R_{i+1}$ at least once.

Observe that the well known instance Bad-Gadget, first defined in [GW99], 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 2.3** An SPP instance containing a rim-by-rim dispute reel is not safe under filtering.

**Proof:** Let $S$ be an SPP instance containing a rim-by-rim dispute reel $\Pi$. Using the fact that $\Pi$ 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. The activation sequence is based on the convenient propagation of announcements from one pivot vertex to its predecessor.

As we have already seen, since $\Pi$ 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 $\Pi$ is rim-by-rim, there exist activation sequences $\sigma_j$ such that $\bar{\pi}^i \xRightarrow{\sigma_i} \bar{\pi}^{i+1} \xRightarrow{\sigma^{i+1}} \cdots \xRightarrow{\sigma^{i-1}} \bar{\pi}^i$. 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 $\sigma$ on $S[\Pi]$. By Property 2.1, 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 $\sigma_{\text{spoke}}$ 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. \qed

Now consider the instance in Fig. 2.11. Clearly, this instance contains a DR $\Pi$ 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 $R_2 = (3 \ 1)$. $\Pi$ is not rim-by-rim: in particular, no activation sequence exists that, starting from the one-rim path assignment.
2.4. CHARACTERIZING SAFETY UNDER FILTERING

for pivot \( u_0 (\tilde{\pi}^0) \), makes path \( R_1Q_2 \) available at vertex 2. In fact, assume that the instance is in state \( \tilde{\pi}^0 \), that is, vertices 2 and 3 select their spoke paths, while vertices on \( R_0 \) select a subpath 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 2.2, we are then able to show an oscillation even on DRs that are not rim-by-rim.

**Lemma 2.4** An SPP instance containing a dispute reel which is not rim-by-rim is not safe under filtering.

**Proof:** Let \( S \) be an SPP instance containing a dispute reel \( \Pi = (\vec{U}, \vec{Q}, \vec{R}) \) which is not rim-by-rim. If \( |\vec{U}| = 2 \), the statement follows from Lemma 2.2. Otherwise, consider \( S[\Pi] \). Since \( \Pi \) is not rim-by-rim by hypothesis, there are at least \( \tilde{\pi}^i \) and \( \tilde{\pi}^{i+1} \) such that \( \not\exists \sigma : \tilde{\pi}^i \not\succ \tilde{\pi}^{i+1} \). 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 Fig. 2.12.

Let \( \Sigma \) be the set of all the activation sequences that, starting from the one-rim path assignment \( \tilde{\pi}^0 \), make path \( Q_2 \) available in the set of choices of some vertex \( v_m \). More formally, \( \forall \sigma \in \Sigma, \tilde{\pi}^0 \not\succ \tilde{\pi}_t \), where \( R_1[v_m]Q_2 \in \text{choices}_t(v_m) \)

![Figure 2.11: A DR which is not rim-by-rim. Vertex 0 is omitted for brevity.](image)
for some \( m \) and \( t \). Note that \( \Sigma \) contains at least the activation sequence obtained by activating the edges of \( R_1 \) in reverse order. Such activation sequence 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 “farthest away” from \( u_1 \) 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 \( \pi_0 \sigma' \sim \pi_t \), where \( \pi_t(u_1) = R_1Q_2 \). 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 \( \#\sigma : \bar{\pi}_0 \sim \bar{\pi}_1 \).

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 possible 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 \( \sigma' \). Let \( \sigma_1 \) consist of the activations of all the edges

Figure 2.12: A portion of a dispute reel which is not rim-by-rim, used in the proof of Lemma 2.4. Different paths are represented using different strokes. Spoke paths \( Q_0 \), \( Q_1 \), and \( Q_2 \) are in thicker stroke.
2.4. CHARACTERIZING SAFETY UNDER FILTERING

in $R_0$ in reverse order, starting from $v_m$. Let $\pi_t$ be the path assignment after $\sigma_1$, that is, $\pi_t \overset{\sigma_1}{\Rightarrow} \pi_t$. Note that $\pi_t$ is such that path $R_0[v_h]Q_1$ has been withdrawn at each $v_h$, $h < m$. We now construct $\sigma_2$ 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 subpath of $R_1$ from $u_1$ to $v_m$, that is, $R_1 = \bar{R} [v_m]$. Now consider the dispute wheel $\Pi' = ((v_m u_1), (R_1[v_m]Q_2 Q_1), (R[v_m] \bar{R}))$. We now show that $\Pi'$ is a DR. Being $\Pi$ a DR, Condition (i) of Definition 2.5 holds since $v_m \not\in Q_1$ and $u_1 \not\in R_1[v_m]Q_2$. Condition (ii) is trivially satisfied by vertices on paths $Q_1$ and $Q_2$, because both are spoke paths in $\Pi$. 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 $v_m$, $v_j \not\in R_1[v_m]$ if $j > m$, and $v_j \not\in R_0[v_m]$ if $j < m$. Again, being $\Pi$ a DR, Condition (iii) holds for paths $Q_1$ and $Q_2$, and we have $R_1[v_m] \cap Q_1 = \emptyset$.

We then conclude that if $\Pi$ is not rim-by-rim, then it contains a dispute reel having two pivot vertices. By Lemma 2.2, instance $S$ is not safe under filtering.

By combining Lemmas 2.2, 2.3, and 2.4, we can state the following theorem.

**Theorem 2.1** An SPP instance containing a dispute reel is not safe under filtering.

**Multiple Solutions and Safety Under Filtering**

We now exploit Theorem 2.1 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. We highlight that uniqueness of the stable state has already been proved to be a necessary condition for safety [SSZ09]. Besides independently asserting the necessity of this condition for safety under filtering based on the presence of dispute reels in the SPP instance, we provide a way
for network operators to pinpoint the portions of the BGP configuration that are responsible for the potential instability.

**Theorem 2.2** If an SPP 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 $\Pi$. $\Pi$ is derived by merging two stable path assignments $\pi_1$ and $\pi_2$. Let $T_1$ and $T_2$ be the routing trees induced by $\pi_1$ and $\pi_2$, and let $T = T_1 \cap T_2$. Each spoke path in $\Pi$ 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 2.5). Rim paths are built up by vertices which are not in the intersection of $\pi_1$ and $\pi_2$, 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, $\Pi$ is a dispute reel. By Theorem 2.1, the presence of a dispute reel in $S$ is enough to conclude that $S$ is not SUF. $\square$

An important consequence of Theorem 2.2 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 without any knowledge of the routing policies. 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.

**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 SPP 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
2.4. CHARACTERIZING SAFETY UNDER FILTERING

in Definition 2.5. Then, we show that the presence of such a dispute wheel implies the existence of a dispute reel.

Lemma 2.5 Consider an SPP instance $S$. If $S$ is not safe under filtering, then there exists a dispute wheel $\Pi$ which satisfies the following conditions:

(i) Conditions (ii) and (iii) of Definition 2.5.

(ii) For all $u_i \in \vec{U}$, $u_i$ 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 $\Pi$ satisfying the above conditions. The statement follows by noting that $\Pi$ must also be present in $S$. □

Observe that the dispute wheel of Lemma 2.5 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 2.6 Given an instance $S$, suppose it contains a dispute wheel $\Pi = (\vec{U}, \vec{Q}, \vec{R})$ satisfying the conditions in the statement of Lemma 2.5. Then, $S$ contains a dispute reel.

Proof: If $\Pi$ is already a DR, the statement trivially holds. Otherwise, for $\Pi$ not to be a reel (while still satisfying the conditions in the statement of Lemma 2.5), 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 $i_j \notin \{i-1, i\}$ (see Fig. 2.13). Without loss of generality, assume that $i_k < i$ is the closest index to $i$ in the order induced by $\vec{U}$. Condition (iii) of Lemma 2.5 ensures that $u_i$ prefers path $R_{i_k}[u_i]Q_{i_{k+1}}$ to $Q_i$. Now consider the dispute wheel $\Pi' = (\vec{U}', \vec{Q}', \vec{R}')$, where $\vec{U}' = (u_i u_{i_k+1} \ldots u_{i-1})$, $\vec{Q}' = (Q_i Q_{i_{k+1}} \ldots Q_{i-1})$, and $\vec{R}' = (R_{i_k}[u_i] R_{i_{k+1}} \ldots R_{i-1})$. Intuitively, $\Pi'$ is obtained by "chopping" $\Pi$, using path $R_{i_k}[u_i]$ as the new rim path associated with vertex $u_i$. Observe that every spoke path in $\Pi'$ is a also spoke path in $\Pi$. Moreover, every rim path in $\Pi'$ except $R_{i_k}[u_i]$ is also a rim path in $\Pi$, and $R_{i_k}[u_i]$ is a subpath of $R_{i_k}$. Therefore, $\Pi'$ trivially satisfies all the conditions of Lemma 2.5. Moreover, by the definition of index $i_k$, we know that $\Pi'$ is such that $u_i$ only appears in $Q_i$. 
Figure 2.13: A dispute wheel where pivot vertex $u_i$ appears in rim paths other than $R_i$ and $R_{i-1}$. By Lemma 2.6, another dispute wheel can be constructed such that $u_i$ appears in exactly 3 paths.

$R_{ik} [u_i]$ and $R_{i-1}$. By applying this construction, we force one pivot vertex at a time to satisfy Condition (i) of Definition 2.5, even if $R_{ik}$ contains other pivot vertices than $u_i$. Hence, after iterating the construction at most $|\mathcal{U}|$ times, we eventually end up with a dispute reel.

We stress that Condition (iii) of Lemma 2.5 is strictly necessary to apply the construction in Lemma 2.6. As a counterexample, consider again the instance in Fig. 2.6. The DW $((1 2 3), ((1 0) (2 0) (3 0)), ((1 3 2) (2 1 3) (3 2 1)))$ only violates Condition (iii) of Lemma 2.5. 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, however, no DR can be constructed starting from that DW.

**Theorem 2.3** If an SPP instance $S$ is not safe under filtering, then it contains a dispute reel.

**Proof:** Lemma 2.5 ensures that $S$ contains a dispute wheel satisfying some particular constraints. We can then apply Lemma 2.6 to find a dispute reel in $S$. □

By combining Theorems 2.1 and 2.3, 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 SPP instance is safe under filtering (Corollary 1 of [FJB07]) and has a unique stable state (Theorem V.4 of [GSW02]). Obviously, as safety can be viewed as a special case of safety under filtering, the absence of a dispute wheel also implies that an SPP instance is safe (also stated in Theorem V.9 of [GSW02]). Fig. 2.5a effectively displays those implications.

As a side effect of our findings, we argue that the “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 SPP instance is safe, SUF, and has a unique stable state (solvable). Moreover, dispute reels still reflect the structure of routing policies only. In particular, the following theorem holds.

**Corollary 2.1** The absence of a DR in an SPP instance $S$ implies that $S$ has a unique stable state, and is safe.

**Proof:** From Theorem 2.2 we know that $S$ must have a unique stable state. Since safety is a special case of safety under filtering, Theorem 2.3 proves the rest of the statement. 

\[ \square \]

## 2.5 Robustness

Safety under filtering is a useful concept to study the impact of route filters on routing stability. An interesting related problem is the impact 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 SPP instance is referred to as robustness (see Section 2.3).

As pointed out in [FJB05, FJB07], the removal of edges and vertices has the same effect as filtering all the paths that traverse those edges and vertices. As a consequence, an instance that is SUF is also robust. Following the findings of Section 2.4, we now show that the class of robust SPP instances is strictly larger than the class of instances that are SUF. Consider instance FILTHY-GADGET in Fig. 2.14.

**Lemma 2.7** FILTHY-GADGET is not safe under filtering.
CHAPTER 2. A MODEL FOR BGP STABILITY PROBLEMS

Figure 2.14: Filthy-Gadget: an instance which is robust but not safe under filtering. Vertex 0 is omitted for brevity.

Proof: The statement follows by Theorem 2.1, since Filthy-Gadget contains a DR $\Pi = (\bar{U}, \bar{Q}, \bar{R})$, where $\bar{U} = (1234)$, $\bar{Q} = ((10)(20)(30)(40))$, and $\bar{R} = ((1XY2)(23)(3ZX4)(4YZ1))$. □

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 2.8 Instance Filthy-Gadget is safe.

Proof: Let $\sigma$ be any fair activation sequence. Given that $\pi_t(0) = (0)$ for all $t$, by the fairness of $\sigma$ each neighbor of 0 is prevented from selecting path $\epsilon$. In particular, after some time vertex 2 can only use paths $(230)$ or $(20)$. Since $Y$ accepts both paths from vertex 2, $Y$ is prevented from selecting path $(YZ10)$, which is less preferred. Vertex 4 is therefore prevented from selecting path $(4YZ10)$. Since 4 is a neighbor of 0, it is also prevented from selecting $\epsilon$. Hence, by the fairness of $\sigma$, vertex 4 will end up selecting path $(40)$ permanently, in turn forcing vertex $X$ to permanently choose path $(X40)$. Since path $(XY20)$ will not be advertised by $X$, vertex 1 is prevented from selecting path $(1XY20)$. Also, being 1 a neighbor of 0, it will end up selecting path $(10)$ permanently. Vertex $Z$, in turn, will be forced to select path $(Z10)$, preventing vertex 3 from selecting $(3ZX40)$. 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 \( \sigma \) is stable. Since we did not make any hypothesis on \( \sigma \), we conclude that Filthy-Gadget is guaranteed to reach this stable path assignment for any fair activation sequences, that is, Filthy-Gadget is safe. \( \square \)

Lemma 2.9 Instance Filthy-Gadget is robust.

Proof: By Lemma 2.8, 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 \( \Pi \) we described above. It is easy to see that its supporting instance \( S[\Pi] \) is built on the same graph as Filthy-Gadget. Hence, removing one or more links forcedly creates an instance where \( \Pi \) does not exist anymore. In order to complete the proof, we need to demonstrate that \( \Pi \) 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 \( \Pi' = (\vec{U}', \vec{Q}', \vec{R}') \) exists having X, Y, or Z as a pivot vertex.

(i) Assume that \( X \) is a pivot vertex of \( \Pi' \). Without loss of generality, we say \( X = u'_0 \). Then \( \vec{Q}'_0 = (X Y 2 0) \) and \( \vec{R}'_0 = (X 4) \), 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 \( \vec{Q}'_0 \) contains a pivot node \( u'_2 = Y \). The latter case results in a DW where spoke path \( \vec{Q}'_0 \) shares vertex Y with rim path \( \vec{R}'_1 \). In both cases, \( \Pi' \) cannot be a DR.

(ii) Consider vertex Y instead and assume it is a pivot vertex of \( \Pi' \). Without loss of generality, we say \( Y = u'_i \). We have two cases, namely \( \vec{Q}'_i = (Y Z 1 0) \) or \( \vec{Q}'_i = (Y 2 0) \).

- if \( \vec{Q}'_i = (Y Z 1 0) \), then \( u'_{i-1} = 4 \). We now have either \( u'_{i-2} = X \) or \( u'_{i-2} = 3 \). The former case implies that \( \vec{Q}'_{i-2} \) contains pivot vertex Y. The latter case implies that \( \vec{R}'_{i-2} \) intersects \( \vec{Q}'_i \) at vertex Z. Hence, \( \Pi' \) cannot be a DR.

- if \( \vec{Q}'_i = (Y 2 0) \), then \( u'_{i-1} = 1 \). We now have either \( u'_{i-2} = Z \) or \( u'_{i-2} = 4 \). The former case implies that \( \vec{Q}'_{i-2} \) and \( \vec{R}'_{i-1} \) share vertex X. The latter case implies that pivot vertex Y also appears in \( \vec{R}'_{i-2} \). In both cases, \( \Pi' \) cannot be a DR.
(iii) Last, for vertex $Z$ we can apply a similar argument to that used for vertex $X$. Assume that $Z$ is a pivot vertex of $\Pi'$, namely $Z = u'_0$. 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 $\Pi'$ cannot be a DR. The only other possibility is $u'_2 = Y$, i.e., $Y$ is also a pivot vertex, which has already been ruled out by the previous case.

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

We performed an exhaustive analysis which independently confirmed the result of Lemma 2.9. Namely, we generated all the possible combinations of failures of one or more links, and then ran the greedy algorithm introduced in [GSW02] on the resulting instance. That algorithm is correct, that is, it never misreports an instance as safe. Our brute force analysis confirmed that removing one or more links from Filthy-Gadget results in a safe instance. Unfortunately, the greedy algorithm is not smart enough to also prove the safety of the original instance, because it does not fully exploit vertices that are prevented from selecting specific paths.

Due to the existence of Filthy-Gadget, we can conclude that robustness relates to other classes of SPP instances as shown in Fig. 2.15. In particular, the following theorem holds.

**Theorem 2.4** SUF is strictly contained in ROBUSTNESS.

**Proof:** It has been already shown [FJB07] that any safe under filtering SPP instance is also robust. Strict inclusion of SUF in ROBUSTNESS is proved by the existence of Filthy-Gadget, which is robust but not safe under filtering, as shown by Lemmas 2.7 and 2.9.

In a sense, Theorem 2.4 involves that improper handling of route filters can impact the stability of BGP routing more than network faults could do.

Also, due to strict inclusion of SUF in ROBUSTNESS, the absence of DRs is a sufficient condition for an SPP instance to be robust.

**Corollary 2.2** The absence of a DR in an SPP instance $S$ implies that $S$ is robust.

**Proof:** The statement follows by Theorem 2.3, noting that ROBUSTNESS is a special case of SUF, as shown by Theorem 2.4.
2.6 Related Work

The need to avoid disadvantages raised by BGP convergence has spurred significant research efforts on BGP stability, over the last decade. Varadhan et al. [VGE00] firstly 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 are due to Griffin et al. [GW99, GSW99, GW00, GSW02]. Among the results they presented, those works formalized the SPP and the SPVP models (see Section 2.2) we adopt in this thesis, and 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) 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 (see, e.g., [Gao01, DEH+07]), then safety is guaranteed. In [Cha06] 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.

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 findings, we show that dispute reels can replace dispute wheels, giving raise to less constraining sufficient conditions for all those properties (see Corollaries 2.1
and 2.2).

Feamster et al. [FJB05, FJB07] explored the impact of autonomy and expressiveness on the stability of the BGP protocol. In those works, a crucial question is posed about ranking and filtering (referring to eBGP): “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. A necessary condition for safety under filtering is the absence of a particular subclass of DWs, called dispute rings [FJB07]. However, such a condition is not sufficient for several guarantee safety not even the presence of a stable state. In this chapter, we presented a characterization of safety under filtering which closes the large gap between previously known sufficient and necessary conditions (see Fig. 2.5).

In [SSZ09] it has been shown that the existence of a unique stable state is a necessary condition for safety. Indeed, authors proved that the coexistence of two stable states implies the existence of an oscillation. Policies are modeled with SPP. Although the model for BGP dynamics is slightly different from SPVP, the result also holds in SPVP [SSZ09]. Besides independently asserting the necessity of this condition based on the presence of dispute rings in the network, we take this result one step further. Namely, we show a simple technique that, taking multiple stable states as input, pinpoints the portions of the BGP configuration which define a DR (thus making the configuration not safe under filtering).

Other models have been proposed in literature for problems related to the stability of path-vector policy-based protocols. Several variants of the SPVP algorithm where only some classes of activation sequences are allowed, have been proposed in literature (e.g., [GSW99, GR00, BOR+02, FJB07]). However, it has been shown [CDR08] that any relaxed version of the original SPVP model is only able to capture a strictly smaller set of routing oscillations. Also, a model has been recently proposed in [SFR11] to take into account “spurious” BGP announcements of less-preferred routes. We argue that such spurious announcements reflect specific implementation issues which are not closely related to the BGP protocol itself and its original design. The model can be hardly generalized to other policy-based protocols.

The safety problem has also been studied from a game theoretic perspective (e.g., in [GSW02, FP08, Cha08]), where BGP convergence is mapped to a (pure) Nash equilibrium. [NRTV07] is a good introduction to the application of game theoretical techniques to interdomain routing problems.

Algebraic approaches have been taken in [Sob05, Sob03, GS05, CGG06]. These works described convergence conditions that are based on properties of
path rankings, and showed that the no DW condition finds a counterpart also in algebraic models. In particular, in [GS05] the authors proposed a relaxation of the guidelines presented in [GR00]. We plan to extend our studies to game theoretic and algebraic models in the future.

2.7 Conclusions

In this chapter, we presented the formal models we adopt in this thesis to study BGP stability. Moreover, we recalled the definition of the most relevant BGP stability problems, and of known sufficient and necessary conditions for guaranteed convergence of BGP configurations.

Further, we enriched the state of the art by providing the “No Dispute Reel” condition that fills the large gap between currently known necessary and sufficient conditions. We proved that the No Dispute Reel condition is a characterization of the safety under filtering problem. As a side effect of our findings, we also showed that dispute reels can replace the well-known dispute wheel structure in the study of several BGP stability problems, since it provides a looser sufficient conditions for many BGP stability problems. Moreover, we deepened into the inclusion relationship between different BGP stability problems. In particular, we showed a robust BGP network that is not safe under filtering. In a sense, this means that the autonomy of adding (possibly misconfigured) filters can be more harmful than network faults. Another 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, starting from the sole stable states and with no knowledge of the routing policies and the outcome of the BGP decision process at any BGP speaker in the network.

Fig. 2.15 illustrates the inclusion relationship between different stability problems after our findings. Also, Fig. 2.5 compares the state of the art before (Fig. 2.5a) and after (Fig. 2.5b) our contributions, highlighting how dispute reels close the large gap between existing sufficient and necessary conditions.

Still, a lot of theoretical problems are left open. We believe that finding a characterization for the SAFETY problem is among the most important ones.
Chapter 3

Hard Times for Static Checks *

3.1 Introduction

As stressed in Chapter 2, side effects of BGP oscillations are serious and should be avoided, since they negatively impact both router performance and quality of offered services. For this reason, it is desirable to prevent routing oscillations by configuration design.

In this chapter, we analyze the complexity of statically checking a BGP configuration for guaranteed convergence, assessing computational complexity of BGP stability problems defined in Chapter 2. We devote Section 3.2 to prove the computational complexity of the safety problem, solving a long standing open problem, firstly formulated in [GW99]. In Section 3.3, we study the computational complexity of other stability problems, namely suf, robustness and no-dr. Unfortunately, we find that all the considered stability problems are computationally hard. In particular, we show that i) safety is \text{coNP-hard}; ii) suf is \text{coNP-complete}; iii) robustness is \text{coNP-hard}. Even worse, checking a BGP configuration for the absence of dispute reels is \text{coNP-complete}. This implies that the no-dw condition is the only known sufficient (but not necessary) condition that can be checked efficiently in SPP [GSW99].

Stimulated by the above list of negative results, we investigate whether stability problems can be made tractable by sacrificing the expressive power of routing policies, while preserving full autonomy of BGP speakers. To this purpose, in Section 3.4, we extend the SPP model to arbitrary restrictions of BGP policy expressiveness, formulating the k-SPP model. We instanciate the k-SPP model to capture the so-called Local Transit policies [GGSS09], a very common configuration paradigm adopted in eBGP, where policies are functions only of the ingress and the egress points. We obtain as a result the 3-SPP model, also proposed in [CBE10] as an effective model to capture common configurations in the Internet. The same model can be effectively applied to iBGP route reflection configurations, where route propagation depends on the previous and the next hop in the iBGP path (see Chapter 1). In the same section, we also prove that the BGP stability problems remain computationally intractable even if policies are restricted to exclusively be Local Transit Policies. Even worse, we show that the only problem which can be solved efficiently in SPP [GSW99], i.e., the no-dw problem, is coNP-complete in 3-SPP. This implies that no way currently exists to overcome the computational unfeasibility of directly solving safety in 3-SPP, since verifying if any of the known sufficient conditions for safety is satisfied is also computationally hard.

In our search for policy expressiveness restrictions that enable an efficient static assessment of BGP stability, we eventually find that safety is solvable in polynomial time in 2-SPP. However, policies are so restricted in 2-SPP to be completely unsuitable for practical uses. We describe an efficient algorithm to check safety in the 2-SPP model in Section 3.5.

Related work is discussed in Section 3.6, and conclusions are drawn in Section 3.7.

3.2 Safety is coNP-Hard

We recall that the safety problem is defined as follows (see Chapter 2): given an SPP instance, is it safe, i.e., is it guaranteed to always converge to a stable state?

We now prove that safety is coNP-hard in the SPP model using a reduction from sat complement [Pap94]. In order to prove such a result, we first need to show some technical properties regarding the SPP instance of Fig. 3.1, which we call Twisted gadget. Twisted has vertex set $V = \{0, x, \bar{x}, a, b, c_1, \ldots, c_m\}$ and edge set $E = \{(0, a), (0, b), (a, x), (b, \bar{x}), (x, \bar{x})\} \cup \{(c_1, x), (c_1, \bar{x}), \ldots, (c_m, x), (c_m, \bar{x})\}$. Policies are as described in Fig. 3.1. Ver-
3.2. SAFETY IS CONP-HARD

![Twisted gadget diagram](image)

Figure 3.1: Twisted gadget.

Vertices $c_i$, with $i = 1, \ldots, m$, also have links to another portion of the network not explicitly shown in Fig. 3.1. Each path $P^c_i$ passes through the portion of the network that is not shown and is ranked better than $(c_i 0)$.

We now prove two important properties of Twisted.

**Lemma 3.1** For each activation sequence, there do not exist two instants $t'$ and $t''$ such that $\pi_{t'}(x) = (x \bar{x} b 0)$ and $\pi_{t''}(\bar{x}) = (\bar{x} x a 0)$.

**Proof:** Suppose, for a contradiction, that there exists an activation sequence such that $\pi_{t'}(x) = (x \bar{x} b 0)$ and $\pi_{t''}(\bar{x}) = (\bar{x} x a 0)$. Denote by $t_P$ the first time when path $P = (v \ldots 0)$ is selected by vertex $v$. By definition of SPVP, we have that $t_{a0} < t_{xa0} < t_{\bar{x}xa0}$ and $t_{b0} < t_{\bar{x}b0} < t_{x\bar{x}b0}$. Since vertex 0 cannot withdraw path (0), vertex $a$ ($b$, resp.) cannot select the empty path after $t_{a0}$ ($t_{b0}$, resp.).

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

Then it must be $t_{x\bar{x}b0} < t_{xa0}$ and, by symmetry, $t_{\bar{x}xa0} < t_{x\bar{x}b0}$. Hence, $t_{xa0} < t_{\bar{x}xa0} < t_{\bar{x}b0} < t_{x\bar{x}b0} < t_{xa0}$, that is a contradiction. **□**
**Lemma 3.2** For each fair activation sequence, if a vertex $c_j$ and a time $t'$ exist such that $\forall t > t' \ \pi_t(c_j) = (c_j 0)$, then a time $t''$ exists such that $\forall t > t'' \ \pi_t(x) = (x a 0)$ and $\pi_t(\bar{x}) = (\bar{x} b 0)$.

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

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

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

We now use the TWISTED gadget and the results from Lemmas 3.1 and 3.2 to reduce the opposite of the SAT problem, namely SAT COMPLEMENT, to SAFETY. Let $F$ be a logical formula in conjunctive normal form with variables $X_1 \ldots X_n$ and clauses $C_1 \ldots C_m$. We construct an SPP instance $S$ in polynomial time with respect to the size of the SAT COMPLEMENT instance as follows (see Figure 3.2).

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

**Theorem 3.1** SAFETY is $\mathsf{CONP}$-hard in the SPP model.

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

If $F$ is unsatisfiable then $S$ is safe.

Consider any fair activation sequence and assume that all vertices $c_j$ select a path $P \neq (c_j 0)$ infinite times. Let $W = \{x_i \in V \mid \exists c_j, \exists t : \pi_t(c_j) = (c_j x_i \bar{x}_i a_i 0)\}$ and $Z = \{\bar{x}_i \in V \mid \exists c_j, \exists t : \pi_t(c_j) = (c_j \bar{x}_i x_i b_i 0)\}$. Consider the boolean assignment $M$ such that $X_i$ is assigned to TRUE if $x_i \in W$, and $X_i$ is assigned to FALSE if $\bar{x}_i \in Z$. Lemma 3.1 ensures that $Z \cap W = \emptyset$. By
3.2. SAFETY IS CONP-HARD

construction of $S$, each clause in $F$ is satisfied by at least a variable in $M$, that is a contradiction.

Then there must exist a time $t'$ and a vertex $c_k$ such that $\forall t > t' \, \pi_t(c_k) = (c_k 0)$. By Lemma 3.2, this implies that there exists a time $t'' > t'$ after which each vertex $x_i$ always selects path $(x_i, a_i 0)$ and each vertex $\bar{x}_i$ always selects path $(\bar{x}_i, b_i 0)$. The fairness of the activation sequence guarantees that, eventually, each vertex $c_j$ permanently selects $(c_j 0)$, each vertex $a_i$ permanently selects $(a_i 0)$, and each vertex $b_i$ permanently selects $(b_i 0)$. It is easy to check that such a path assignment is stable. Since any fair activation sequence leads to a stable path assignment, if $F$ is unsatisfiable then $S$ is safe.

If $F$ is satisfiable then $S$ is not safe.

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

Let $\pi'$ be a path assignment such that $\pi'(x_i) = (x_i, a_i 0)$, $\pi'(\bar{x}_i) = (\bar{x}_i, b_i 0)$, $\pi'(a_i) = (a_i 0)$, $\pi'(b_i) = (b_i 0)$, and $\pi'(c_j) = (c_j 0)$, where $i = 1, \ldots, n$ and

![Figure 3.2: Reduction from SAT COMPLEMENT to SAFETY.](image-url)
j = 1, \ldots, m. It is easy to check that $\pi'$ is a stable path assignment.

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

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

We conclude that, if $F$ is satisfiable, then $S$ has two stable path assignments. The statement follows by Theorem 3.1 of [SSZ09], which proves that any SPP instance with two distinct stable path assignments is not safe. 

\[ \square \]

### 3.3 No-DR, Safety Under Filtering and Robustness are Computationally Hard

In the following, we first study the computational complexity of checking a BGP configuration for the absence of dispute reels. Then, we exploit the result of such a study for proving the computational intractability of SUF and ROBUSTNESS.

We now prove that NO-DR is \textsc{coNP}-complete by reducing 3-SAT complement to SAT in polynomial time. Refer to Fig. 3.3 for an example of the reduction.

Let $F$ be a logical formula, with variables $X_1, \ldots, X_n$ and clauses $C_1, \ldots, C_m$. For each variable $X_i$, we add to the SPP instance a gadget consisting of three vertices, namely $a_i, x_i,$ and $\bar{x}_i$, and four edges, namely $(x_i 0), (\bar{x}_i 0), (a_i x_i)$ and $(a_i \bar{x}_i)$. Vertices $x_i$ and $\bar{x}_i$ have no permitted paths other than $(x_i 0)$ and $(\bar{x}_i 0)$, respectively. Permitted paths at vertex $a_i$ are $\mathcal{P}^{a_i} = \{(a_i x_i 0), (a_i \bar{x}_i 0)\}$ and the ranking among them is not significant. Intuitively, $a_i$ represents variable $X_i$. Gadgets corresponding to variables are at the bottom of Fig. 3.3.

For each clause $C_j$, we add to the SPP instance a gadget containing vertices $c_j, c_{j,i}$, and edges $(c_j, c_{j,i})$ and $(c_{j,i}, c_{j+1})$, where $i = 1, 2, 3$. Intuitively,
3.3. NO-DR, SAFETY UNDER FILTERING AND ROBUSTNESS ARE COMPUTATIONALLY HARD

vertex $c_j$ (clause vertex) represents clause $C_j$ while vertex $c_{j,i}$ (literal vertex) represents the $i$-th literal in $C_j$. Further, if $X_i$ appears in the $i$-th literal in $C_j$, then we add an edge $(a_i, c_{j,i})$, and we set $P^{c_{j,i}} = \{(c_{j,i} c_{j+1} 0), (c_{j,i} a_i x_i 0)\}$ if literal represented by $c_{j,i}$ is $X_i$, $P^{c_{j,i}} = \{(c_{j,i} c_{j+1} 0), (c_{j,i} a_i \bar{x}_i 0)\}$ otherwise. Among the two paths in $P^{c_{j,i}}$, $(c_{j,i} c_{j+1} 0)$ is the most preferred. The permitted paths at vertex $c_j$ are $(c_j 0)$ plus the extension of the longest path permitted at each vertex $c_{j,i}$, $i = 1, 2, 3$. Path $(c_j 0)$ is the least preferred path, while the ranking of other paths can be arbitrary. Gadgets corresponding to clauses are placed at the top of Fig. 3.3.

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

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

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

**Theorem 3.2** NO-DR is coNP-complete in the SPP model.

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

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

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

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

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

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

Co-$NP$ completeness follows from noting that a DR on $S$ is a succinct disqualification for NO-DR. □

Since the absence of a DR is a characterization of $\text{suf}$ in the SPP model (see Chapter 2), we can state the following theorems.

**Theorem 3.3** $\text{suf}$ is co-$NP$-complete in the SPP model.

**Proof**: The statement directly follows from Theorem 3.2 considering that the absence of a DR is a necessary and sufficient condition for $\text{suf}$ in the SPP model. □

Also, observe that the SPP instance in Fig. 3.3 is safe under filtering iff it is also robust. In fact, filtering a path $P = (u v \ldots 0)$ at vertex $u$ is equivalent to removing edge $(u, v)$ from the graph. This property allows us to reduce 3-SAT complement to ROBUSTNESS using the same reduction used in Theorem 3.3.

**Theorem 3.4** ROBUSTNESS is co-$NP$-hard in the SPP model.

**Proof**: Observe that the SPP instance in Fig. 3.3 is safe under filtering if and only if it is also robust. In fact, filtering a path $P = (u v \ldots 0)$ at vertex $u$ is equivalent to removing edge $(u, v)$ from the graph. As a consequence, it is possible to reduce 3-SAT complement to ROBUSTNESS using the same reduction used in Theorem 3.3, hence the statement. □
3.4 Restricting to Local Transit Policies

In this section, we assess whether BGP stability can be efficiently checked with static analysis when policy expressiveness is restricted to Local Transit Policies only. First, we introduce the variation of the SPP model, namely 3-SPP, that we use for capture the Local Transit Policies. Then, we assess the complexity of the BGP stability problems introduced in the previous chapter. Observe that, the negative results we prove in this section can be extended to the model in [GW99], since any 3-SPP configuration can be expressed in the model proposed in [GW99] without changing the size of the input.
Extending the SPP Model: 3-SPP and k-SPP

SPP can model every possible BGP policy specification, but it requires explicit listing and ranking of all paths. As a consequence, the size of an SPP instance is bound to the size of \( \mathcal{P} \), which can be exponential in \(|V|\). However, in practice, BGP policies are not always specified on a per-path granularity level, as imposed in the SPP model. For example, in eBGP network operators configure BGP policies without knowing the entire network topology.

We now describe 3-SPP, a variant of SPP in which vertices are forced to rank and filter announcements on the basis of the first 3 hops in the path. This model allows BGP actors to specify Local Transit Policies [GGSS09], i.e., policies based on their neighbor pairs (e.g., paths received from neighbor \( x \) should not be exported to neighbor \( y \)).

We now formally define the 3-SPP model. Let \( G = (V, E) \) be defined as for standard SPP instances. Each vertex \( u \in V \), with \( u \neq 0 \), is assigned a set of permitted path fragments \( \tilde{\mathcal{P}}^u \) such that \( (u 0) \in \tilde{\mathcal{P}}^u \) and paths \( (u v w) \) can be in \( \tilde{\mathcal{P}}^u \) if \( u, v, \) and \( w \) are distinct vertices in \( V \) and \( (u, v), (v, w) \in E \). The only permitted path fragment at vertex 0 is \( \tilde{\mathcal{P}}^0 = \{ (0) \} \).

To reach 0, a vertex \( u \in V - \{0\} \) can use any path \( P = (u v_1 \ldots v_n 0) \), starting with a fragment in \( \tilde{\mathcal{P}}^u \) and obtained by concatenating any permitted fragment at each vertex \( v_i \), with \( i = 1, \ldots, n \). Path fragments contain exactly 3 vertices except for the case of 0 and of its neighbors, which can reach 0 directly. Let \( \tilde{\mathcal{P}} = \bigcup_{u \in V} \tilde{\mathcal{P}}^u \).

Each vertex \( u \in V - \{0\} \) ranks path fragments in \( \tilde{\mathcal{P}}^u \) according to a function \( \tilde{\lambda}^u : \tilde{\mathcal{P}}^u \rightarrow \mathbb{N} \) which assigns a level of preference to paths starting with a fragment in \( \tilde{\mathcal{P}}^u \). Namely, if \( \tilde{\lambda}^u((u v w)) < \tilde{\lambda}^u((u x y)) \) then any path starting with \( (u v w) \) is preferred to any path starting with \( (u x y) \). Similarly to the SPP model, the empty path is always permitted, i.e., \( \epsilon \in \tilde{\mathcal{P}}^u, \forall u \in V - \{0\} \), and unreachability is the last resort, i.e., \( \forall P \in \tilde{\mathcal{P}}^u, P \neq \epsilon: \tilde{\lambda}^u(P) < \tilde{\lambda}^u(\epsilon) \).

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

An instance \( \tilde{S} \) of 3-SPP is a triple \( (G, \tilde{\mathcal{P}}, \tilde{\lambda}) \). An example 3-SPP instance is depicted in Fig. 3.4 using a graphical convention similar to that used for SPP instances (see Chapter 2). The list beside each vertex \( u \) represents the permitted path fragments in \( \tilde{\mathcal{P}}^u \) sorted by increasing values of \( \tilde{\lambda}^u \). For example, vertex 2 can use path fragments in \( \tilde{\mathcal{P}}^2 = \{ (2 1 0), (2 0) \} \) to reach 0 and prefers...
3.4. RESTRICTING TO LOCAL TRANSIT POLICIES

Figure 3.4: An instance of the 3-SPP model.

(2 0). The empty path and \( \mathcal{P}^0 \) are omitted for brevity. Vertex 3 decides not to propagate the path received from 0 to 2, and permitted path fragments at vertex 2 result from filtering action performed by 3 and ranking configured at 2. Observe that path fragment 432 at vertex 4 models two distinct paths from 4 to 0, namely 4320 and 43210, that have the same rank.

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

Safety, Safety Under Filtering and Robustness are still Computationally Hard

We now state the complexity of the BGP stability problems in 3-SPP. First, we consider SAFETY.

**Theorem 3.5** SAFETY is \( \text{coNP-hard} \) in the 3-SPP model.

**Proof:** We can use the same reduction from SAT COMPLEMENT to SAFETY applied in Theorem 3.1. In fact, the SPP instance constructed in the reduction can be easily translated into a 3-SPP instance, since every permitted path at each vertex is uniquely identified by the first three hops in the path. The reduction proves the statement. \( \square \)

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

**Theorem 3.6** suf is coNP-hard in the 3-SPP model.

**Proof:** Let $S$ be the SPP instance in Fig. 3.3 and construct the 3-SPP instance $S'$ by truncating all paths in $S$ with length greater than 3. Since each permitted path in $S$ is identified by its first three hops, there is a one-to-one mapping between permitted paths in $S$ and permitted paths in $S'$. This implies that each filter in $S$ can be mapped to a unique filter in $S'$. We conclude that $S'$ is suf if and only if $S$ is suf, hence a construction analogous to that described in Section 3.3 can be applied to reduce from 3-sat complement to suf in 3-SPP. □

We now state the complexity of no-dr in 3-SPP.

**Theorem 3.7** no-dr is coNP-complete in the 3-SPP model.

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

Since a 3-SPP instance is robust if and only if the corresponding SPP instance is robust, we can directly extend Theorem 3.4.

**Theorem 3.8** robustness is coNP-hard in the 3-SPP model.

**Complexity of No-DW**

Since an instance of k-SPP can be uniquely translated into an SPP instance, we can extend the definition of DW as follows: we say that an instance of k-SPP contains a DW if its translation to SPP contains a DW.
3.4. Restricting to Local Transit Policies

In the SPP model NO-DW can be solved in polynomial time [GSW99] by looking for a cycle in an auxiliary graph called dispute digraph, whose construction takes polynomial time. In the following, we analyze the computational complexity of NO-DW in the 3-SPP model. We do it in two steps. First, we deal with the basic problem of deciding whether a given vertex of a given 3-SPP instance can establish a path to 0. We call this problem PATH and we show that it is \( \mathcal{NP} \)-complete. Second, we exploit such a result to prove that NO-DW in the 3-SPP model is \( \text{coNP-complete} \).

PATH is \( \mathcal{NP} \)-complete since it is possible to reduce 3-SAT to PATH. Let \( F \) be a 3-SAT formula with variables \( X_1, \ldots, X_n \) and clauses \( C_1, \ldots, C_m \). We construct a 3-SPP instance as follows. For each variable \( X_i \) we insert vertices \( v_i, x_i, \) and \( \bar{x}_i \), and we build a gadget having edges \((v_i, x_i)\) and \((v_i, \bar{x}_i)\). For each clause \( C_j \) we build a gadget consisting of vertices \( c_j \) and \( c_{j,k} \) and edges \((c_j, c_{j,k})\), \((c_{j,k}, c_{j,k+1})\) with \( k = 1, 2, 3 \). Also, we add to the instance vertices \( v_{n+1}, c_{m+1} \)

Figure 3.5: Reduction from 3-SAT to PATH.
CHAPTER 3. HARD TIMES FOR STATIC CHECKS

and 0, and edges \((c_{m+1}, 0)\) and \((v_{n+1}, c_1)\). Fig. 3.5 shows an example of the construction, where variable gadgets are on the left side while clause gadgets are on the right side.

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

Consider literal \(X_i\), with \(i = 1, \ldots, n\). Let \(P = (v_i \ x_i \ c_{j_1,k_1} \ \ldots \ c_{j_n,k_n} \ v_{i+1})\) be the path from vertex \(v_i\) to vertex \(v_{i+1}\) that traverses all the literal vertices \(c_{j_p,k_p}\) such that the corresponding literal in clause \(C_{j_p}\) is \(X_i\). If there are no such literals, then path \(P\) simply consists of edges \((v_i, x_i)\) and \((x_i, v_{i+1})\). We add to the graph constructed so far all the edges of \(P\). We apply exactly the same procedure for literal \(\bar{X}_i\). We then get from path \(P\) all the ordered triples of consecutive vertices and add each triple \((u \ v \ w)\) to \(\bar{P}^u\). For example, in Fig. 3.5 there is a path \((v_1 \ x_1 \ c_{1,1} \ c_{m,1} \ v_2)\) because we assume, without loss of generality, that the first literal both in \(C_1\) and in \(C_m\) is \(X_1\). For each vertex \(c_j\), set \(\bar{P}^{c_j}\) only contains paths \((c_{j,k} \ c_{j+1} \ c_{j+1,l})\), with \(l = 1, 2, 3\) and for each vertex \(c_{j,k}\), we add to \(\bar{P}^{c_{j,k}}\) paths \((c_{j,k} \ c_{j+1} \ c_{j+1,l})\), with \(l = 1, 2, 3\). This construction ensures that if vertex \(v_i\) attempts to establish a path to 0 via \(x_i\) (\(\bar{x}_i\)), it cannot use a path including \(c_{j,k}\) if and only if \(\bar{X}_i\) (\(X_i\)) is the \(k\)-th literal in \(C_j\), representing the fact that clause \(C_j\) cannot be satisfied by literal \(c_{j,k}\).

We define \(\bar{P}^{n+1} = \{(v_{n+1} \ c_1 | \forall k = 1, 2, 3)\}\) and \(\bar{P}^{m+1} = \{(c_{m+1} | 0)\}\).

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

**Lemma 3.3** Path is \(\mathcal{NP}\)-complete in the 3-SPP model.

**Proof:** Consider the construction depicted in Fig. 3.5. We now show that vertex \(v_i\) can establish a path to 0 if and only if the corresponding 3-SAT formula \(F\) is satisfiable.

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

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

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

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

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

**Lemma 3.4** The 3-SPP instance $S$ constructed in the reduction from 3-SAT to PATH (see Fig. 3.5) contains no DW.

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

By construction, permitted paths of all the vertices in $S$ are subpaths of $P = P_1 \ldots P_n (v_{n+1} c_1) Q_1 \ldots Q_m (c_{m+1} 0)$. Paths $Q_i$ are such that $Q_i = \{ c_i c_{i,j} c_{i+1} \}$, where $j$ is either 1, 2, or 3. Each path $P_i$ starts at $v_i$, ends in $v_{i+1}$, and traverses $x_i (\bar{x}_i)$ and all the vertices $c_{j,k}$ such that the corresponding literal in clause $C_j$ is $\bar{x}_i$ ($x_i$). This implies that $P_j \cap P_{j+1} = \{ v_{j+1} \}$ for each $j$, and $P_j \cap P_k = \emptyset$, if $k \not\in \{ j, j+1 \}$. Since no rim path can contain a node $c_i$, all the rim paths must be subpaths of $P_1 P_2 \ldots P_n$. However, since vertices $v_i$ are ordered and all paths $P_i$ intersects only at vertices $v_i$, no cycle among rim paths can be built, yielding a contradiction.

The proof is completed by showing that no vertex $c_i$ can appear in any rim path of any DW II. In fact, suppose that there exists a non empty set of vertices $Z = \{ c_j, \ldots, c_k \}$ such that each vertex $c_i \in Z$ appears in one or more rim paths. Obviously, $c_{m+1}$ cannot belong to $Z$. Consider, among all the vertices in $Z$, the vertex $c_h$ with the highest index. Let $R$ be a rim path in which appears
CHAPTER 3. HARD TIMES FOR STATIC CHECKS

Let $\mathcal{R}[c_h]$ be the subpath of $\mathcal{R}$ starting from $c_h$. By definition of $c_h$ and by construction of $S$, $\mathcal{R}[c_h]$ can only be $(c_h, c_{h, h'})$, with $h' = 1, 2, 3$. In fact, all permitted paths at $c_h$ are sequences of vertices $c_i$ and $c_{i,j}$, such that $i > h$ and $c_{h+1}$ cannot appear in $\mathcal{R}[c_h]$ by definition of $c_h$. Hence, vertex $c_{h, h'}$ must be a pivot vertex of $\Pi$, and its spoke path must be a path $(c_{h, h'} c_{h+1} \ldots 0)$ since it must be extended by a permitted path of $c_h$. By definition of DW, the rim path of $c_{h, h'}$ should be one among paths $(c_{h, h'} c_{h+1} \ldots 0)$, that is, $c_{h+1}$ is also on a rim path. This leads to a contradiction, because $c_h$ is defined to be the vertex with the highest index among those appearing in a rim path. □

**Theorem 3.9** NO-DW is coNP-complete in the 3-SPP model.

**Proof:** We prove the statement by reducing 3-SAT COMPLEMENT to NO-DW. Let $F$ be a logical formula with variables $X_1, \ldots, X_n$ and clauses $C_1, \ldots, C_m$. We construct an instance $\tilde{S} = ((V, E), \tilde{\mathcal{P}}, \tilde{\mathcal{L}})$ of 3-SPP as follows. Let $\tilde{S}' = ((V', E'), \tilde{\mathcal{P}}', \tilde{\mathcal{L}}')$ be the 3-SPP instance constructed as above (see Fig. 3.5). Then, $V = V' \cup \{1, 2\}$, let $E = E' \cup \{(1, v_1), (1, 2), (2, 0)\}$, let $\tilde{\mathcal{P}} = \tilde{\mathcal{P}}' \cup \{(1 v_1 x_1), (1 v_1 \bar{x}_1), (2 0), (2 1 0), (2 1 v_1)\}$ and let $\tilde{\mathcal{L}} = \tilde{\mathcal{L}}' \cup \{\lambda^1, \lambda^2\}$, where $\lambda^1$ and $\lambda^2$ are such that paths $(1 2 0)$ and $(2 1 0)$ are the most preferred permitted paths at vertices 1 and 2 respectively.

Intuitively, we added two extra vertices 1 and 2, and defined policies such that a DW exists in $\tilde{S}$ only if 1 can establish a path to 0. By applying the same arguments as in the proof of Lemma 3.3 we therefore have that $\tilde{S}$ has no dispute wheel if and only if $F$ is unsatisfiable. This implies that NO-DW is coNP-hard in the 3-SPP model. The proof is completed by noting that a DW on $\tilde{S}$ is a succinct disqualification for NO-DW, that is, a succinct proof that $\tilde{S}$ is a negative instance. □

### 3.5 Achieving Efficient Checking

The 2-SPP model is an instance of the k-SPP framework (see Section 3.4) that allows ASes to only specify path fragments of length 2. In other words, policies can be specified on a per-neighbor basis only: all paths from the same neighbor are either accepted or filtered and are equally preferred. As in 3-SPP, any arbitrary deterministic criterion can break ties.

2-SPP is similar to the model adopted in [FSS06]. By applying the technique in [FSS06], it can be shown that every 2-SPP instance has at least a stable path assignment $\pi$ and $\pi$ can be computed in polynomial time. Observe,
3.5. ACHIEVING EFFICIENT CHECKING

however, that 2-SPP allows configurations that are not safe, e.g., the famous SPP instance DISAGREE [GW99] (see Fig. 2.1) can be represented in 2-SPP. Indeed, in the DISAGREE instance, all vertices do not filter any route, and rank paths towards 0 on a per-neighbor basis only.

In the following, we show that BGP stability can be also checked in polynomial time. Given a 2-SPP instance $\hat{S} = (G = (V, E), \hat{P}, \hat{\Lambda})$, a path fragment $(u v)$, with $u, v \in V$, is consistent if there exists a sequence of permitted path fragments $P_1, P_2, \ldots, P_n$ in $\hat{P}$ such that $(u v)P_1P_2 \ldots P_n(0)$ is a simple path on $G$. Consistency of a given path fragment can be trivially checked in polynomial time, since it boils down to check for the existence of a path in a directed graph. For this reason, we consider in the following only 2-SPP instances in which all permitted path fragments are consistent.

We now present an algorithm, which we called NH-GREEDY, that efficiently solves SAFETY in 2-SPP. NH-GREEDY is an adaptation of the greedy algorithm in [GSW02]. NH-GREEDY incrementally grows a set of stable vertices for which convergence is guaranteed. The set of stable vertices at iteration $i$ of NH-GREEDY is denoted by $V_i$. At iteration $i$ NH-GREEDY also computes a partial path assignment $\pi_i^\ast$, that is, a path assignment where $\forall u \notin V_i, \pi_i^\ast(u) = \epsilon$. At the beginning, $V_0 = \{0\}$ and $\pi_0^\ast(0) = (0)$. Let $H_i$ be the set of vertices $u \notin V_i$ such that the most preferred path fragment is either $B^u = \epsilon$ or $B^u = (u v)$, where $v \in V_i$. If $H_i$ is not empty, then $V_{i+1} = V_i \cup H_i$, $\pi_{i+1}^\ast(u) = \pi_i^\ast(u)$ if $u \in V_i$, and $\pi_{i+1}^\ast(u) = B^u \pi_i^\ast(u)$ for each $u \in H_i$. Otherwise, if $H_i$ is empty, NH-GREEDY terminates. At each iteration, NH-GREEDY either inserts at least one vertex in $V_i$ or terminates, hence it terminates after at most $|V|$ iterations. If NH-GREEDY terminates after $k$ iterations with $V_k = V$ then we say that it succeeds, otherwise it fails. Being derived from the algorithm in [GSW02], NH-GREEDY inherits the correctness property from the original algorithm. This implies that, after $k$ iterations, each vertex $v \in V_k$ is guaranteed to eventually select path $\pi_k^\ast(v)$ in any fair activation sequence. As a consequence, if NH-GREEDY succeeds, then the 2-SPP instance is safe. In the following, we show that if NH-GREEDY fails then the instance is not safe.

Let $G' = (V, E')$ be the directed graph such that $(u, v) \in E'$ if and only if $(u, v) \in \hat{P}_x$. Given a partial path assignment $\pi$ and a vertex $u$ such that $\hat{P}_u \neq \{\epsilon\}$ and $\pi(u) = \epsilon$, the ideal path $P_u^\pi$ of $u$ in $\pi$ is the simple path from $u$ to 0 obtained by performing a depth-first visit on $G'$ starting from $u$. Vertices are visited according to $\hat{\Lambda}$, i.e., the neighbor with the highest preference is visited first. By definition, $P_u^\pi = (w_1 \ldots w_n v_1 \ldots v_m)$, where $w_1 = u$, $v_m = 0$, $n \geq 1$, $m \geq 1$, $(u w_2)$ is the most preferred fragment in $P_u$, $\pi(w_i) = \epsilon$ for $i = 1, \ldots, n$, and $\pi(v_j) = (v_j \ldots v_m)$ for $j = 1, \ldots, m$. Intuitively, the ideal
path $P^\pi_u$ from $u$ to 0 which traverses the best-ranked neighbor of $u$; also, all the vertices $w_i \in P^\pi_u$ select the best-ranked simple path that extends a path in $\pi$. Observe that such a path must exist because all path fragments are assumed to be consistent, i.e., $(u \, w_1)$ generates at least a path on $G$.

**Lemma 3.5** Assume that NH-GREEDY fails on a 2-SPP instance $S$ after $k$ iterations with partial path assignment $\pi^*_k$. Then, there exists a stable path assignment $\bar{\pi}$ on $S$ such that $u$ selects its ideal path, i.e., $\bar{\pi}(u) = P^\pi_k u$.

**Proof:** We construct a sequence of partial path assignments $\pi_1, \pi_2, \ldots, \bar{\pi}$ by iteratively growing $\pi^*_k$. In particular, we iteratively take any vertex in $V - V_k$ and we progressively extend $\pi^*_k$ by adding the ideal path of considered vertex, along with all the subpath of that ideal path. More formally, let $u$ be any vertex in $V - V_k$ and let $P^\pi_k u = (u \, w_1 \ldots w_n \, v_1 \ldots v_m)$ be the ideal path of vertex $u$ in $\pi^*_k$. We construct $\pi_1$ as follows. $\pi_1(u) = P^\pi_k u$. Also, for each $w_i \in P^\pi_k$ let $\pi_1(w_i) = (w_i \ldots w_n \, v_1 \ldots v_m)$ and for each $v \in V_k$ let $\pi_1(v) = \pi^*_k(v)$. Then, we consider any other vertex $z$ such that $\pi_1(z) = \epsilon$ and $z \in V - V_k$ (if one exists, otherwise stop). Given $P^\pi_1$ the ideal path of $z$, we construct the (partial) path assignment $\pi_2$ by extending $\pi_1$ with the same technique we used for constructing $\pi_1$. Since $V$ is finite, we eventually find a path assignment $\bar{\pi}$ defined for each $v \in V$.

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

**Theorem 3.10** SAFETY can be solved in polynomial time in the 2-SPP model.

**Proof:** Given a 2-SPP instance $S$, $S$ is safe if and only if NH-GREEDY succeeds. We have already discussed that if NH-GREEDY succeeds $S$ is safe. On the other hand, if NH-GREEDY fails after $k$ iterations, it is possible to build two distinct stable path assignments. In fact, let $u$ be any vertex in $V - V_k$. Lemma 3.5 ensures that there exists a stable path assignment $\pi'$ such that $\pi'(u) = P^\pi_k u$. Path $P^\pi_k u$ must be in the form $P^\pi_k u = P'(z \, v)P''$ where $z \not\in V_k$ and $v \in V_k$. 

CHAPTER 3. HARD TIMES FOR STATIC CHECKS
Observe that $z \neq u$, since $u \not\in V_k$. Consider the stable path assignment $\pi''$ such that $\pi''(z) = P_z^k$, constructed as in Lemma 3.5. Obviously, $\pi' \neq \pi''$ at least for vertex $z$ since $z \not\in V_k$. Since two distinct stable path assignments exist, $S$ is not safe as proved by Theorem 3.1 of [SSZ09]. □

3.6 Related Work

In [GW99], a BGP model is proposed where policies are described by means of functions that implement import and export filters, similarly to real-world BGP configuration languages. Several important complexity results are proved:
(i) checking if a BGP network has a stable routing (solvability) is \( \mathcal{NP} \)-complete,
(ii) deciding whether a BGP network can be trapped in a permanent oscillation is \( \mathcal{NP} \)-hard, and
(iii) solving solvability under any combination of link failure is \( \mathcal{NP} \)-hard. The complexity of safety in that model is left open.

In [GSW02], the SPP model is introduced, and it is shown that solvability is \( \mathcal{NP} \)-hard also in SPP. This result could not be evinced from [GW99], as the translation from one model to the other might take exponential time. The complexity of safety and robustness in SPP is left open.

In the SPP model, no-dw can be solved in polynomial time [GSW99] by checking for a cycle in an auxiliary graph called dispute digraph, whose construction takes polynomial time.

In [JR04], a model is used in which BGP policies are applied consistently network-wide based on a classification of neighbors into groups. In this setting, a polynomial time algorithm is given to check whether the structure of the classes can lead to specific BGP policies in which oscillations are possible. The 3-SPP model is similar to the one used in [JR04] in that it also limits the expressiveness of BGP policies. However, the 3-SPP model is more general since it allows each AS to preserve its autonomy, while a common filtering policy that is applied consistently network-wide is assumed in [JR04].

In [FP08], safety is claimed to be PSPACE-complete. However, the adopted model assumes that ASes are omniscient, that is, upon activation they can immediately know the AS-paths that are being used by every other AS, without the need to exchange BGP messages. This assumption makes it very hard to apply the results to any realistic model of BGP.

In [FSS06], an economic-based model is proposed. solvability is studied both in the general case of arbitrary BGP policies and in the restricted case in which BGP policies are based on the next-hop only. The latter model is similar
CHAPTER 3. HARD TIMES FOR STATIC CHECKS

<table>
<thead>
<tr>
<th></th>
<th>2-SPP</th>
<th>3-SPP</th>
<th>SPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFETY</td>
<td>P</td>
<td>\text{coNP-hard}</td>
<td>\text{coNP-hard}</td>
</tr>
<tr>
<td>NO-DW</td>
<td>?</td>
<td>\text{coNP-complete}</td>
<td>P \cite{GSW99}</td>
</tr>
<tr>
<td>NO-DR</td>
<td>?</td>
<td>\text{coNP-complete}</td>
<td>\text{coNP-complete}</td>
</tr>
<tr>
<td>SUF</td>
<td>?</td>
<td>\text{coNP-hard}</td>
<td>\text{coNP-complete}</td>
</tr>
<tr>
<td>ROBUSTNESS</td>
<td>?</td>
<td>\text{coNP-hard}</td>
<td>\text{coNP-hard}</td>
</tr>
</tbody>
</table>

Table 3.1: Complexity of BGP stability problems in different models. P stands for Polynomial time solvable.

to the 2-SPP model we consider in this chapter. Beyond proving NP-hardness of solvability in the general case, authors provide a polynomial algorithm that can always find a stable state in the restricted case. The safety problem is again left open.

3.7 Conclusions

The Border Gateway Protocol was originally designed to accommodate routing preferences autonomously expressed by different subjects participating in the protocol. In this chapter, we studied the computational complexity of several BGP stability testing problems, under the assumption that all the BGP speakers preserve full autonomy in setting their routing preferences. Also, we investigated how expressive routing policies can be in order to allow static assessment of BGP stability to be performed efficiently.

Unfortunately, we found that the most interesting problems about BGP stability are computationally intractable if BGP speakers preserve full autonomy and are allowed to specify policies as expressive as Local Transit Policies. We stress that Local Transit Policies effectively models a common paradigm used for configuring policies in eBGP as well as route propagation rules in iBGP route reflection configurations. Table 3.1 summarizes our results and highlights problems that remain open.

While such results are primarily related to BGP, they can be generalized to any policy-based path vector routing protocol. Moreover, our findings suggest that computational tractability of BGP stability can be achieved by restricting the expressiveness of the policies alone (e.g., in the 2-SPP model). Determining whether there exist restrictions that keep the policies expressive enough for practical uses remains an interesting open problem.
3.7. CONCLUSIONS

Our results on the complexity of BGP stability problems and computational intractability of checking for sufficient conditions motivate a research effort towards a heuristic approach for BGP stability testing. We will adopt this approach in the next chapter, proposing a heuristic and a tool to efficiently solve BGP stability problems.
Chapter 4

A Heuristic-Based Approach to Stability Testing

4.1 Introduction

Started about one decade ago and still ongoing, lots of research efforts have focused on the BGP convergence problem. However, a systematic methodology that stands on the shoulders of this research to build techniques and tools for practically checking BGP configurations is still missing. As a result, operators are often forced to “tweak and pray”, possibly discovering flaws in BGP configurations during operation (e.g., after a misbehavior).

In this chapter we introduce a technique that enables operators to statically check routing configurations for guaranteed convergence. Our approach may be exploited to validate configurations before deployment and to assist operators for improving the quality of routing. A relevant use case consists in performing what-if analyses on the impact of route preference changes on the stability of the BGP network. This especially makes sense in iBGP, where all the routers are typically under control of the same administrative entity (see Chapter 1)

and simple configuration changes (e.g., on the cost of one IGP link) can give raise to BGP instabilities.

Because of the computational complexity of all BGP stability problems (see Chapter 3), the technique we propose is based on a heuristic algorithm that checks SPP instances for their safety. 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 huge BGP configurations, (iii) it is free from false positives, meaning that configurations are only reported as stable if they are not prone to routing oscillations regardless of message timing, and (iv) it can help in spotting troublesome points in configurations that are not stable.

Further, we propose an architecture for a modular tool that exploits our heuristic algorithm to process BGP configurations and return information about the presence of sources of potential instabilities. The design of our tool is general enough to support stability checking of both interdomain (eBGP) and intradomain (iBGP) configurations. We validate the practical applicability of our architecture using a prototype implementation. Although we are mainly interested in the iBGP case, we report results of experiments on Internet-scale eBGP networks [CAI]. This allows us to test the scalability of our approach in extreme cases of thousands of BGP speakers. We find that both configuration to SPP translation and convergence check algorithm run can be implemented efficiently enough to analyze those huge BGP configurations.

The rest of the chapter is organized as follows. Our convergence check algorithm is presented in Section 4.2. We discuss the architecture of a convergence checker tool, together with the optimization techniques that make it scalable in Section 4.3. A complete example of how the tool verifies a BGP configuration is analyzed in Section 4.4. Results of Internet-scale experiments performed using a prototype implementation are discussed in Section 4.5. Related work is reviewed in Section 4.6, and conclusions are drawn in Section 4.7.

4.2 A New Greedy Algorithm

The inherent complexity of BGP stability problems pushed researchers to introduce efficient heuristic algorithms.

In this section, we first briefly recall a greedy algorithm that we call GREEDY, proposed in [GSW02] to solve the SAFETY problem. Second, we define a new greedy algorithm, called GREEDY⁺, which is able to correctly report as stable
4.2. **A NEW GREEDY ALGORITHM**

more configurations than **GREEDY** and can be efficiently implemented. This section also presents a qualitative comparison between **GREEDY** and **GREEDY**$^+$. 

**A Known Greedy Algorithm**

Given an **SPP** instance, algorithm **GREEDY** proceeds by iteratively growing a stable path assignment on it. If the algorithm terminates successfully, the path assignment defines a spanning tree that is a solution for the **SAFETY** problem on the given **SPP** instance. Otherwise, **GREEDY** 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, i.e., $V_0 = \{0\}$. As the stable set grows, a path assignment $\pi$ defined on the vertices in $V_i$ is iteratively built. We say that a path $P$ is compatible with a path assignment $\pi$ at iteration $i$ if $P = P'(u,v)\pi(v)$, where $P'$ does not contain vertices in $V_{i-1}$, $(u,v) \in E$, and $v \in V_i$.

Algorithm **GREEDY** is as follows. At any iteration $i > 0$, let $P_v$ be the path compatible with $\pi$ with minimum $\lambda^v(P)$ among the paths at any vertex $v \notin V_{i-1}$. If $P_v = \epsilon$ or $P_v = (v u \ldots 0)$ with $u \in V_{i-1}$, 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 a vertex $v$ is stabilized because either its best compatible path directly reaches an already stabilized vertex or has no path to 0. The algorithm terminates after at most $|V|$ iterations. A solution to the **SPP** instance exists if, after $k$ iterations, **GREEDY** ends with $V_k = V$. The solution is given by the stable path assignment $\pi$. On the contrary, if $V_k \neq V$, **GREEDY** returns a partial path assignment, that is, a path assignment defined on the vertices in $V_k$ only.

Note that the description of **GREEDY** we propose here slightly differs from the one in [GSW02], since we admit only one vertex to enter the stable set at each iteration. We explain in the following that this modified version is equivalent to the original algorithm. We choose to apply this slight modification of the original algorithm in order to better introduce the **GREEDY**$^+$ heuristic and highlight the differences between the two algorithms.

**GREEDY** can fail to find a solution even if the **SPP** instance under consideration is guaranteed to converge. Consider, for example, the instance **DI-SAFEGREE** in Fig. 4.1. As usual, permitted paths are listed next to each vertex in decreasing order of preference. It can be easily verified that any fair activation sequence of **SPVP** on this instance is finite. In fact, in any fair activation sequence vertices 1, 2, and 3 learn about the direct path to 0. After that, edge
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

(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 edge (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 SPVP cannot oscillate on this instance.

We 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 prematurely stops. In fact, path $P = (2 3 0)$ is compatible with $\pi$ because $2, 3 \not\in V_1$, $0 \in V_1$, and $(3, 0) \in E$. However, even if $P = (2 3 0)$ 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 $V_1 \neq V$.

Improving Greedy: the Greedy$^+$ Algorithm

We now describe the Greedy$^+$ algorithm, and we show how it is able to successfully solve the Di-safe-gree instance.

Given an SPP instance $(G = (V, E), P, \Lambda)$, we say that a path $P$ belonging to a set $S$ of paths is consistent with $S$ if either $P = \epsilon$, $P = (0)$, or $P = (v u)P'$ where $(v, u) \in E$, $P' \in S$, and $P'$ is consistent with $S$. For example, let $V = \{0, 1, 2, 3\}$, $E = \{(1, 0), (2, 1)\}$, and $S = \{(0), (1 0), (2 1 3 0)\}$. Path $(0)$ is consistent with $S$ by definition. Path $(1 0)$ is also consistent with $S$, since $(1, 0) \in E$ and $(0)$ is consistent with $S$. On the contrary, $(2 1 3 0)$ is not consistent with $S$. In fact, even if $(2, 1) \in E$, the subpath $(1 3 0)$ is not in $S$ and cannot therefore be consistent with $S$.

Further, for each vertex $v$ we define a set $\bar{P}^v$ of paths called useful set. The useful set $\bar{P}^v$ is initialized with the paths in $P^v$ that are consistent with $P$. Let $\bar{P} = \bigcup_{v \in V} \bar{P}^v$. 
4.2. A NEW GREEDY ALGORITHM

What follows is a description of Greedy+. As Greedy, the Greedy+ algorithm iteratively grows a set of stable vertices \( V_i \) and a path assignment \( \pi \). Let \( V_0 = \{ \} \). At iteration \( i > 0 \), Greedy+ performs the following steps:

\( (i) \) Prune paths that cannot be selected because a better ranked path is 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 \( \mathcal{P}^u = \{P'\} \), remove from \( \mathcal{P}^v \) all the paths \( Q \) such that \( \lambda^v(Q) > \lambda^v(P) \). Intuitively, since \( P \) will be always available at \( v \), all the paths that \( v \) ranks worse than \( P \) can be pruned from the instance since they will never be selected by \( v \).

\( (ii) \) Enforce consistency. For each vertex \( v \not\in V_{i-1} \), remove from \( \mathcal{P}^v \) all the paths that are not consistent with \( \mathcal{P} \). In a sense, this step extends the pruning performed at Step \( (i) \) to vertices which are farthest from any vertex in \( V_{i-1} \).

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

If Greedy+ stops after \( k \) iterations, its output consists of a stable set \( V_k \) and sets \( \mathcal{P}^u \forall v \in V \), with \( |\mathcal{P}^v| = 1 \forall v \in V_k \). If \( V_k = V \), Greedy+ computes a stable path assignment \( \pi \) for the input instance such that \( \mathcal{P}^u = \{\pi(v)\} \forall v \in V \). Otherwise, Greedy+ returns a partial path assignment \( \pi \) on \( V_k \), such that for every \( v \in V_k \), \( \mathcal{P}^u = \{\pi(v)\} \), every vertex in \( \pi(v) \) is in \( V_k \), and \( \pi \) is undefined for vertices \( v \not\in V_k \). As for Greedy, this partial path assignment is stable in the sense that every vertex \( v \in V_k \) is guaranteed to steadily select the path in \( \mathcal{P}^v \).

Intuitively, Greedy+ differs from Greedy because it exploits the useful set to prune those paths that, starting from a certain iteration, become permanently unavailable. This operation is encoded in Step \( (i) \) of Greedy+. Indeed, if we skip this step, the set \( \mathcal{P} \) is only used to filter out inconsistent paths, and at each iteration \( j \) both Greedy and Greedy+ algorithms select the best path among the consistent ones having a next hop in \( V_j \).

Contrary to Greedy, Greedy+ is able to solve the Di-safe-gree instance. A successful execution of Greedy+ on Di-safe-gree is shown in Table 4.1. Note that, at iteration 2, path \((2 \, 0)\) is evicted from \( \mathcal{P}^2 \) because \((2 \, 1 \, 0)\) is preferred and permanently available (Step \( (i) \)). This action puts in


CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

<table>
<thead>
<tr>
<th>$i$</th>
<th>$V_i$</th>
<th>$C_i$</th>
<th>$\mathcal{P}^1$</th>
<th>$\mathcal{P}^2$</th>
<th>$\mathcal{P}^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>${0}$</td>
<td>$\emptyset$</td>
<td>${(1,0)}$</td>
<td>${(2,3,0),(2,1,0),(2,0)}$</td>
<td>${(3,2,0),(3,0)}$</td>
</tr>
<tr>
<td>1</td>
<td>${0,1}$</td>
<td>${1}$</td>
<td>${(1,0)}$</td>
<td>${(2,3,0),(2,1,0),(2,0)}$</td>
<td>${(3,2,0),(3,0)}$</td>
</tr>
<tr>
<td>2</td>
<td>${0,1,3}$</td>
<td>${3}$</td>
<td>${(1,0)}$</td>
<td>${(2,3,0),(2,1,0)}$</td>
<td>${(3,0)}$</td>
</tr>
<tr>
<td>3</td>
<td>$V$</td>
<td>${2}$</td>
<td>${(1,0)}$</td>
<td>${(2,3,0)}$</td>
<td>${(3,0)}$</td>
</tr>
<tr>
<td>4</td>
<td>$V$</td>
<td>$\emptyset$</td>
<td>${(1,0)}$</td>
<td>${(2,3,0)}$</td>
<td>${(3,0)}$</td>
</tr>
</tbody>
</table>

Table 4.1: A successful execution of $\text{Greedy}^+$ on $\text{Di-safe-gree}$ (Fig. 4.1).

evidence the difference between $\text{Greedy}^+$ and $\text{Greedy}$ (recall that $\text{Greedy}$ stops after the first iteration). Step $(ii)$ then removes $(3,2,0)$ from $\mathcal{P}^3$ since it is inconsistent with $\mathcal{P}$. This allows vertex 3 to enter the stable set, and $\text{Greedy}^+$ to solve the instance in four iterations.

**Properties of Greedy**

We now prove that $\text{Greedy}^+$ exhibits the following highly desirable properties: (i) it exceeds $\text{Greedy}$ in that it is able to correctly report more configurations as stable, (ii) it can be implemented efficiently enough to enable static analysis of otherwise intractable Internet-scale BGP configurations, (iii) it is free from false positives, meaning that configurations are only reported as stable if they are guaranteed to converge to a stable routing, and (iv) it can help in spotting the troublesome points in a detected oscillation. Properties (i) and (ii) are unique to $\text{Greedy}^+$. On the contrary, $\text{Greedy}$ satisfies properties (iii) and (iv), which are also inherited by $\text{Greedy}^+$. In particular, Property (iv) is a direct consequence of the fact that both algorithms return a partial path assignment in case they fail.

We now prove Property 4.1 ensuring that algorithm $\text{Greedy}^+$ can be efficiently implemented, and is therefore suitable to be adopted in a tool that checks BGP configurations for routing stability.

**Property 4.1** Let $n = |\mathcal{P}|$ be the size of an SPP instance $S$. $\text{Greedy}^+$ can be implemented to terminate on $S$ in time that is polynomial in $n$.

**Proof:** A trivial bound follows. Step (i) of $\text{Greedy}^+$ 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 $\lambda^v$ for all the paths in each $\mathcal{P}^e$ 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) enforces consistency, and can be accomplished by
4.2. A NEW GREEDY ALGORITHM

comparing each path in \( \bar{P} \) with all the others, taking \( O(n^3) \). At Step (iii), candidate vertices can be found in \( O(n^3) \) time. Since GREEDY+ executes at most \(|V| \) iterations and an instance of SPP can have \( O(n) \) vertices, GREEDY+ can be implemented to run in \( O(n^4) \) time.

We now prove Property 4.2 which ensures that GREEDY+ is deterministic, namely at any time when multiple choices are possible, performing any of them does not alter the output. To this purpose, we first prove the following four Lemmas.

**Lemma 4.1** If GREEDY+ terminates after \( k \) iterations, its output is completely defined by sets \( V_k \) and \( \bar{P}^v \ \forall v \in V_k \).

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

Now, initialize the stable set \( V_0 \) to \( V_k \), execute Steps (i) and (ii) of GREEDY+ on \( S' \). We now show that, after doing so, \( \bar{P}'v = \bar{P}^v \), \( \forall v \in V \). This is trivially true for vertices \( u \in V_k \), as no path is ever removed from \( \bar{P}'u \). Observe that the outcome of Step (i) of GREEDY+ only depends on the topology of the graph \( G' \), the ranking functions \( \Lambda' \), and the sets of useful paths \( \bar{P}^v \), with \( v \in V_k \). By the definition of \( S' \), at Step (i), a path is removed from \( \bar{P}^v \) if it is removed from \( \bar{P}'v \). 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 \( \bar{P}'v \neq \bar{P}^v \). Then, there exists a path \( P \) such that either \( P \notin \bar{P}'v \land P \in \bar{P}^v \) or \( P \in \bar{P}'v \land P \notin \bar{P}^v \). In the first case, the execution of Step (ii) on \( S' \) has removed from \( \bar{P}'v \) a path that the execution of GREEDY+ on \( S \) regarded as consistent. But this is impossible, since \( \forall v \in V \), \( \bar{P}^v \subseteq \bar{P}'v \), so there can be no path that is consistent with \( \bar{P} \) and is not consistent with \( \bar{P}' \). In the second case, the execution on \( S \) has removed from \( \bar{P}v \) a path \( P \) that the execution on \( S' \) considered as consistent. Since it cannot be \( P \notin \bar{P}v \), then for \( P \) to be inconsistent with \( \bar{P} \), it may only be the case that \( P = (v \ldots u)P_u \), where \( P_u \notin \bar{P}u \) and \( P_u \in \bar{P}u \). In turn, this is only possible if there exists a path \( P_w \) such that \( P_u = (u \ldots w)P_w \), with \( P_w \notin \bar{P}w \) and \( P_w \in \bar{P}w \). By proceeding this way, we must eventually end up on a vertex \( x \) in \( V_k \), possibly 0. By recalling that \( \bar{P}'v = \bar{P}^v \ \forall v \in V_k \) by construction, we have a contradiction in that it should be \( P_x \notin \bar{P}x \) and \( P_x \in \bar{P}'x \). \( \square \)
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

Lemma 4.2 Consider a path \( P \) that is inconsistent with \( \bar{P} \) at iteration \( i \) of \textsc{Greedy}\(^+\). Then, \( P \) is inconsistent at any iteration \( j > i \).

Proof: The property follows by observing that \textsc{Greedy}\(^+\) never adds new paths to \( \bar{P} \). \( \square \)

Lemma 4.3 At any iteration \( i \) of \textsc{Greedy}\(^+\), \( C_i \cap V_i = V_i - V_{i-1} \).

Proof: By construction, \( C_i \cap V_{i-1} = \emptyset \). 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. \( \square \)

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

Lemma 4.4 Consider an arbitrary iteration \( i \) of \textsc{Greedy}\(^+\) and a vertex \( v \in C_i \). Then there exists an iteration \( j \) 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 \( P \in \bar{P}^v \) 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 \( \bar{P}^v \) during the execution of \textsc{Greedy}\(^+\) (Lemma 4.2) and \( P \) has the minimum value of \( \lambda^v \) among the paths in \( \bar{P}^v \) that are consistent with \( \bar{P} \), \( P \) can never be removed from \( \bar{P}^v \) at Step (i) of \textsc{Greedy}\(^+\). Moreover, if \( P = \epsilon \), by definition \( P \) is a consistent path. Otherwise, if \( P = (v \ u)Q, u \in V_{i-1}, \{Q\} = \bar{P}^u \), then \( P \) will remain consistent with \( \bar{P} \) because its next hop is \( u \in V_{i-1} \), so \( \bar{P}^u \) will not be updated after iteration \( i \). Thus, \( P \) cannot be removed from \( \bar{P}^v \) at Step (ii). Overall, starting from iteration \( i \), path \( P \) will always be available in \( \bar{P}^v \) and will always have the minimum value of \( \lambda^v \). In other words, \( v \) satisfies the conditions of Step (iii) at any iteration \( h \geq i \), i.e., \( v \in C_h \cup V_h \).

Since \( \forall h \geq i \) we have \( v \in C_h \cup V_h \), and \textsc{Greedy}\(^+\) only terminates when the candidate set is empty, by Lemma 4.3 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 never leave the stable set. \( \square \)

Finally, we can prove guaranteed determinism of \textsc{Greedy}\(^+\) algorithm.

Property 4.2 Consider a set \( C_j \) of vertices satisfying the criteria of Step (iii) at an arbitrary iteration \( j \) of \textsc{Greedy}\(^+\). The output of \textsc{Greedy}\(^+\) does not change, regardless of the choice of vertex \( v \in C_j \) performed at iteration \( j \).
4.2. A NEW GREEDY ALGORITHM

Proof: Assume that $\text{GREEDY}^+$ terminates at iteration $k$ and consider that, by Lemma 4.1, it is sufficient to prove the assertion for sets $V_k$ and $\bar{P}^u$ with $u \in V_k$. Consider an arbitrary vertex $v \in C_j$ at iteration $j < k$. By Lemma 4.4, we know that $v \in C_h$ for any iteration $h \geq j$, until $v$ eventually enters the stable set. Also, as shown in the proof of Lemma 4.4, the best path $(v w)P$, $w \in V_h$, is always in $\bar{P}^v$. Therefore, regardless of the iteration at which $v$ is actually selected, the set $\bar{P}^v$ is always updated with path $(v w)P$. Moreover, the set of paths that become inconsistent with $\bar{P}$ after setting $\bar{P}^v = \{(v w)P\}$ does not depend on the iteration either.

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

Given the similarities between $\text{GREEDY}^+$ and $\text{GREEDY}$, it is easy to check that the same property also holds for $\text{GREEDY}$. Moreover, this property confirms that the description of $\text{GREEDY}$ given in this section and the original description given in [GSW02] are equivalent.

The following two properties, namely Properties 4.3 and 4.4, state that $\text{GREEDY}^+$ can be used as a static, centralized, and deterministic algorithm to efficiently emulate the behavior of SPVP in the long term, thus dealing with non-determinism of SPVP. This feature of $\text{GREEDY}^+$ can be effectively exploited, e.g., by a network administrator that wants to analyze what are the paths steadily selected by BGP speakers in a given configuration.

We start with Property 4.3, that guarantees that every vertex that $\text{GREEDY}^+$ puts in the stable set always selects the same path in any fair activation sequence of SPVP. Therefore, if $\text{GREEDY}^+$ terminates with $V_k \neq V$, the source of potential oscillations must be searched in those vertices that are left out of the stable set.

**Property 4.3** Consider an SPP instance $S$ and run $\text{GREEDY}^+$ on $S$. Let $P \in \mathcal{P}^u$ be a path that $\text{GREEDY}^+$ deletes at iteration $j$. Then, for any fair activation sequence $\sigma$ of SPVP on $S$, there exists a time $t'$ such that $\forall t > t'$, $\pi_t(v) \neq P$.

Proof: The statement asserts that $\text{GREEDY}^+$ deletes only those paths that would be discarded by any fair activation sequence of SPVP. The proof is by induction on the iteration $j$ of $\text{GREEDY}^+$. At iteration $j = 1$, since $\bar{P}^u = \mathcal{P}^u$ for all $u \in V$, $\text{GREEDY}^+$ deletes a path $P$ from $\bar{P}^v$ 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 $\sigma$, there must exist a time $t'$ such that $(0, v)$ 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 $\mathcal{P}^v$ 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 $\{P'\} = \mathcal{P}^u$. Observe that the induction hypothesis assures that previously deleted paths are eventually discarded after time $t'$. Then, by the fairness of $\sigma$, 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 $R \notin \mathcal{P}^w$. By the induction hypothesis, there exists $t'$ such that $t > t'$ $\pi_t(w) \neq R$. Then, by the fairness of $\sigma$, $v$ must receive a message that withdraws the availability of $R$ at a time $t'' > t'$. Therefore, $\pi_t(v) \neq P \forall t > t''$. \hfill $\square$

Property 4.4 If Greedy$ terminates successfully on an instance $S$ of SPP, then $S$ is safe and has a unique solution.

Proof: Assume that Greedy$ terminates successfully on $S = (G = (V, E), \mathcal{P}, \Lambda)$ after $k$ iterations, and let $V_k = V$ be the stable set after the $k$-th iteration. By Step (iii) of Greedy$ we know that for each vertex $v \in V_k$, $\mathcal{P}^v$ only contains one path (possibly $\epsilon$), because Greedy$ deleted all the permitted paths but one. By Property 4.3, for each deleted path $P = (v\ \ldots)$ there exists a time $t'$ after which vertex $v$ is permanently unable to select $P$, i.e., $\pi_t(v) \neq P$ for each $t > t'$. This implies that there exists a time $\tilde{t}$ after which all the deleted paths are never selected. Therefore, for all $v \in V$ and for all $t > \tilde{t}$, $\pi_t(v) = P_\nu$, where $\mathcal{P}^\nu = \{P_\nu\}$. \hfill $\square$

Finally, Property 4.5 shows that Greedy$ exceeds existing algorithms that can be used for stability check. In fact, we have already shown that Greedy$ can solve instances for which state-of-the-art sufficient conditions for safety do not hold (instance Di-safe-gree in Fig. 4.1 is among these). We complete the
4.2. A NEW GREEDY ALGORITHM

Proof in the Appendix by showing that Greedy$^+$ also solves all the instances solved by Greedy. We need the following lemma to prove Property 4.5.

**Lemma 4.5** Let $S$ be an instance of SPP. If Greedy terminates on $S$ finding a path assignment $\pi^*$, then Greedy$^+$ also terminates on $S$ finding $\pi^*$.

**Proof:** By Property 4.2 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 $\pi^*$ 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 $\pi$ 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 $\lambda_u$ among those compatible with $\pi$, for some $v \in V_{j-1}$. By the induction hypothesis, $\bar{P}^v = \{\pi(v)\}$, therefore path $(u \ v)\pi(v)$ is consistent with $\bar{P}$. We show that path $(u \ v)\pi(v)$ must still be in $\bar{P}^u$ at iteration $j$. Lemma 4.2 ensures that Step (ii) did not remove path $(u \ v)\pi(v)$ from $\bar{P}^u$. That is, since path $(u \ v)\pi(v)$ is consistent with $\bar{P}$ at iteration $j$, it was always consistent during the previous iterations. By the induction hypothesis, $\forall w \in V_{j-1} \ P^w = \{\pi(w)\}$, therefore all the paths that are regarded as consistent by Greedy$^+$ are necessarily compatible with $\pi$. Hence, since $(u \ v)\pi(v)$ is consistent with $\bar{P}$ and has minimum $\lambda_u$ among the paths compatible with $\pi$, it must also have minimum $\lambda_u$ among the paths consistent with $\bar{P}$. Therefore path $(u \ v)\pi(v)$ cannot be deleted at Step (i) of Greedy$^+$, and vertex $u$ is a candidate to be inserted in the stable set by Greedy$^+$.

Since, by Property 4.2, 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 $\pi^*$.

**Property 4.5** The set of SPP instances that Greedy$^+$ can successfully solve is strictly larger than the set of instances that Greedy is able to solve.
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

Proof: Lemma 4.5 proves the inclusion. The strictness is supported by DISAFE-GREEDY, which is not solved by GREEDY, as we discussed above, while it is solved by GREEDY+ as shown in Table 4.1. □

Note that GREEDY+ is not able to solve all the instances where GREEDY fails to find a solution. For example, FILTHY-GADGET (see Fig. 2.14) cannot be solved by GREEDY+. Because of the computational complexity of the SAFETY problem (see Chapter 3), no efficient algorithm can be complete and correct at the same time. However, given the particular nature of the problem, we stress that the need to avoid false positives (i.e., networks that are mistakenly reported as safe) outweighs the risk of allowing false negatives (i.e., networks that are mistakenly reported as potentially unsafe). Characterizing the set of additional instances that GREEDY+ can solve with respect to GREEDY is still an open problem. However, in Section 4.5 we empirically assess the effectiveness of GREEDY+ compared to GREEDY by using a quantitative analysis on huge BGP topologies.

4.3 An Automated Convergence Checker

GREEDY+ exhibits several desirable properties. We now show that the benefits of these properties exceed mere theoretical speculation. We propose a design of a modular tool to automatically check whether a given set of BGP configurations leads to guaranteed convergence.

The architecture of our tool is designed to support checking arbitrary BGP configurations, regardless of whether they come from routers within a single AS (iBGP) or from different ASes (eBGP). Therefore, the description of the tool presented in this section is independent of whether we are considering eBGP or iBGP.

Architecture of the Checker

An architectural overview of the tool we realized is shown in Fig. 4.2, where sharp boxes represent data, and rounded boxes represent the main architectural components. The checker processes input data along a pipeline from BGP configurations to the final output, which tells the user whether the system is safe or not. In the latter case, the tool returns a portion of the network that can be responsible for potential oscillations.

Our architecture encompasses the following components.
4.3. AN AUTOMATED CONVERGENCE CHECKER

A Configuration parser, which parses input BGP configuration policies and translates them into a custom format. Depending on the origin of policy information, the input can be router configuration files, RPSL [AVG+99] policy descriptions, or topologies obtained by monitoring projects such as CAIDA [CAI]. This allows us to support and integrate different sources of policy information by just changing the parser component.

An SPP generator, which takes as input the BGP policies extracted by the parser and builds an SPP instance that models the configurations.

A Stability checker, which implements the GREEDY+ algorithm and runs it on the generated SPP instance in order to check it for safety.

The SPP generator is the most challenging component to design. In fact, enumerating the permitted paths of an SPP instance may take exponential space, since the number of paths in a graph is exponential with respect to the number of vertices.

The first step performed by the checker consists in the translation of the input BGP configuration policies into a custom input-independent format. Especially when the input policies are described using vendor-specific languages, isolating this step has some important benefits: (i) even though configuration languages continuously evolve and different vendors propose proprietary constructions, the only component of the checker that needs to be updated to accommodate these changes is the parser; and (ii) while most configuration languages are designed with an event-condition-action approach in mind, where specific actions are undertaken whenever a particular event takes place and a set of conditions is found to hold, by using the SPP model we rely on an explicit set of ordered paths, which allows us to disclose the network-wide semantic that is hidden behind the policy configurations.

Next, an intermediate vendor-independent representation of the input BGP configurations is built. Consider that, despite the variety of routing policies
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

1: dissemination(v)
2: while receive (P, A) from w do
3: (P', A') = F_v⇐w((P, A))
4: if (P', A') ≠ (ϵ, ⊘) then
5: \( \mathcal{R}_t(v) := \mathcal{R}_{t-1}(v) \cup \{(P', A')\} \)
6: if \( \mathcal{R}_t(v) ≠ \mathcal{R}_{t-1}(v) \) then
7: for all \( u \mid (u, v) \in E \) do
8: send \( F_v⇒u(((v)P', A')) \)
9: end for
10: end if
11: end if
12: end while

Figure 4.3: The dissemination algorithm used by the checker to generate the paths of the SPP instance.

contained in well-known data sets (e.g., the Internet Routing Registries) and the number of constructions supported by router vendors, it is easy to see that a full-blown configuration can be ultimately mapped to a set of filters. We represent a BGP announcement with a pair \((P, A)\), where \(P\) is a path and \(A\) is a set of BGP attributes. Before a received announcement is processed by a router \(u\), an import filter \(F_{u⇐v}((P, A))\) is applied to the announcement; similarly, before a router \(u\) sends an announcement, an export filter \(F_{u⇒v}((P, A))\) 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 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 \((ϵ, ⊘)\) if the BGP announcement is discarded.

While we extract filters from BGP configurations, we also collect information about the peerings established by each router. This allows us to build the logical graph \(G = (V, E)\).

Similarly to [FB05], where the processing steps of BGP are sequenced into a dissemination phase, a filtering phase, and a ranking phase, we distinguish the generation of routing paths from the actual best route selection operated
4.3. AN AUTOMATED CONVERGENCE CHECKER

by BGP.

Regarding the path dissemination, vertex 0 first starts announcing \((0), \emptyset\). We then run on the other vertices of \(G\) the distributed algorithm in Fig. 4.3. In this algorithm, vertices \(v \in G\) exchange routing messages containing the full set of attributes (including, e.g., AS-path, next-hop, community, etc.), apply all the configured filters, and store received announcements in sets \(R(v)\). Every time a new, not previously observed announcement is received by \(v\), 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 SPP model). It is easy to verify that the algorithm in Fig. 4.3 ends in finite time, and a centralized implementation can be easily realized. A set of permitted paths \(P^v\) for each \(v \in V\) can then be constructed starting from sets \(R\).

To compute path rankings, for each \(v \in V\) we apply the BGP decision process to the announcements that \(v\) has collected in \(R(v)\) during the dissemination. This operation also allows us to define the ranking functions \(\lambda^v\).

After executing the above steps, we define an SPP instance \(S = (G, P, \Lambda)\) with \(P = \bigcup_{v \in V} P^v\) and \(\Lambda = \{\lambda^v \mid v \in V\}\). We use instance \(S\) to study the convergence of the network.

**Optimizing for Scalability**

In principle, mapping real-world BGP configurations (i.e., vendor specific configurations) to a set of explicitly permitted paths is a step that requires exponential space. On the other hand, hardcoding filter applications in the path generation process allows us to avoid generating a large number of paths. However, this is still not enough to be able to efficiently process huge configurations: we need to further reduce the paths to be generated. For this reason, during the dissemination phase and before actually generating the SPP instance, we run two pre-processing steps.

First of all, vertex 0 starts marking 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, \(v\) marks the announcement as reliable and stops considering future incoming announcements (**early stabilization**). In any case, \(v\) then applies the export filter \(F_{v \rightarrow w}((P', \mathcal{A}'))\) and further propagates the announcement to each neighbor \(w \neq u\). Non-reliable paths continue to be disseminated in the standard way. This step corresponds
to precomputing a subset of the stable vertices computed by Greedy. Based
on Property 4.3, a vertex \( v \) marking an announcement \((P, A)\) as reliable is guar-
anteed to select the corresponding path \( P \). This allows us to only generate a
single path for each stabilized vertex. In order to maximize the number of early
stabilized vertices, we evaluate preferences based on the local-preference,
the AS-path length, and the router-id of the announcements.

Our experiments showed that early stabilization is not enough to make
Internet-scale configurations tractable. Therefore, we apply an additional opti-
mization step while generating the SPP instance: vertex \( v \) does not propagate
any announcement that it considers worse than a received reliable announce-
ment \((P, A)\) (early suppression). In fact, since paths from reliable announce-
ments are always available, \( v \) will be unable to select an alternative path ranked
worse than \((P, A)\). This step implements the optimizations found in Greedy\(^+\).

In order to finally generate the SPP instance, path rankings are computed
by running the complete BGP decision process.

As the last step, the SPP instance is checked for guaranteed stability, i.e.,
safety. Observe that an efficient implementation of this step can keep track of
the consistent paths in order to avoid recomputing them at each iteration.

4.4 Walking through the Operation of the Convergence
Checker

In this section we walk through a complete example showing how a BGP config-
uration is verified by our tool. Collection and parsing of the BGP configuration
are omitted.

For the sake of simplicity, we consider a simple eBGP network in which poli-
cies are specified via setting the local-preference attribute. The convergence
checker behaves in a similar way in the iBGP case. Throughout the example,
we always refer to a single network prefix originated by the AS modeled as
vertex 0.

BGP Configuration

Fig. 4.4 shows the BGP topology we consider in our example. The topology
consists of 7 ASes, plus the prefix originator 0. To make the example more
realistic, we labeled adjacencies between ASes with commercial relationships
(customer\(\rightarrow\)provider, peer\(\leftrightarrow\)peer, or sibling\(\leftarrow\)sibling) that are supposed to
be honored in the configurations (see [Gao01, GR00, DEH\(^+\)07]). For conve-
4.4. WALKING THROUGH THE OPERATION OF THE CONVERGENCE CHECKER

Figure 4.4: The example topology we discuss in Sec. 4.4. Edges are labelled according to the commercial relationships between the Autonomous Systems (customer▶provider, peer—peer, or sibling◀sibling).

nience, vertices in Fig. 4.4 are laid out according to the customer-provider hierarchy.

We assume that all the routers in each AS implement homogeneous routing policies, so that a single BGP configuration describes the routing behavior of the whole AS. In particular, ASes in Fig. 4.4 implement the policies described in Fig. 4.5 (observe that the figure contains the policy specifications for all the 7 ASes). In order to provide a specification that is easier to read and independent of a vendor’s specificities, we describe the policies using the Routing Policy Specification Language (RPSL [AVG+99]).

In Fig. 4.5, lines 1 through 18 define so-called **AS-set objects**, which are groups of ASes that will be reused later on in the policy specifications. The policies adopted by each AS are described in the **aut-num objects** in lines 19 through 57. Each **aut-num object** states the policies applied to incoming (import) and outgoing (export) announcements separately. For example, AS0 (lines 19-22) only originates a prefix and has therefore no import policies specified. On the other hand, its export policies correctly state that it must "announce AS0" to each of its neighbors AS1, AS2, and AS3. Note that the exact semantic of "announce AS0" is "announce all the prefixes originated by AS0", which is what we would expect.

Policies may be more complex than AS0’s. For example, AS1 (lines 23-26) applies different preference values to announcements that come from its providers (note the practical use of the **AS-set “AS1:PROVIDERS”**) or directly from AS0. Since in RPSL lower **pref** values correspond to higher preferences, AS1’s policies in Fig. 4.5 implement the prefer-customer rule (see [GR00]) according to the topology in Fig. 4.4. The statement “accept ANY” implies that no filter is being applied on incoming announcements. Most of the policies
**CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AS-set: AS1:PROVIDERS</td>
</tr>
<tr>
<td>2</td>
<td>members: AS2, AS3, AS4</td>
</tr>
<tr>
<td>3</td>
<td>AS-set: AS2:CUSTOMERS</td>
</tr>
<tr>
<td>4</td>
<td>members: AS0, AS1</td>
</tr>
<tr>
<td>5</td>
<td>AS-set: AS2:PROVIDERS</td>
</tr>
<tr>
<td>6</td>
<td>members: AS3, AS4</td>
</tr>
<tr>
<td>7</td>
<td>AS-set: AS3:NEIGHBORS</td>
</tr>
<tr>
<td>8</td>
<td>members: AS1, AS2, AS4, AS5, AS6, AS7</td>
</tr>
<tr>
<td>9</td>
<td>AS-set: AS3:RESTRICTED</td>
</tr>
<tr>
<td>10</td>
<td>members: AS5, AS6, AS7</td>
</tr>
<tr>
<td>11</td>
<td>AS-set: AS4:NEIGHBORS</td>
</tr>
<tr>
<td>12</td>
<td>members: AS1, AS2, AS3</td>
</tr>
<tr>
<td>13</td>
<td>AS-set: AS5:NEIGHBORS</td>
</tr>
<tr>
<td>14</td>
<td>members: AS3, AS6, AS7</td>
</tr>
<tr>
<td>15</td>
<td>AS-set: AS6:NEIGHBORS</td>
</tr>
<tr>
<td>16</td>
<td>members: AS3, AS5, AS7</td>
</tr>
<tr>
<td>17</td>
<td>AS-set: AS7:NEIGHBORS</td>
</tr>
<tr>
<td>18</td>
<td>members: AS3, AS5, AS6</td>
</tr>
<tr>
<td>19</td>
<td>aut-num: AS0</td>
</tr>
<tr>
<td>20</td>
<td>export: to AS1 announce AS0</td>
</tr>
<tr>
<td>21</td>
<td>export: to AS2 announce AS0</td>
</tr>
<tr>
<td>22</td>
<td>export: to AS3 announce AS0</td>
</tr>
<tr>
<td>23</td>
<td>aut-num: AS1</td>
</tr>
<tr>
<td>24</td>
<td>import: from AS0 action pref=50; accept ANY</td>
</tr>
<tr>
<td>25</td>
<td>import: from AS1:PROVIDERS action pref=100; accept ANY</td>
</tr>
<tr>
<td>26</td>
<td>export: to AS1:PROVIDERS announce AS0</td>
</tr>
<tr>
<td>27</td>
<td>aut-num: AS2</td>
</tr>
<tr>
<td>28</td>
<td>import: from AS2:CUSTOMERS action pref=50; accept ANY</td>
</tr>
<tr>
<td>29</td>
<td>import: from AS2:PROVIDERS action pref=100; accept ANY</td>
</tr>
<tr>
<td>30</td>
<td>export: to AS2:PROVIDERS announce !AS2:CUSTOMERS</td>
</tr>
<tr>
<td>31</td>
<td>export: to AS1 announce ANY</td>
</tr>
<tr>
<td>32</td>
<td>aut-num: AS3</td>
</tr>
<tr>
<td>33</td>
<td>import: from AS3:NEIGHBORS action pref=50; accept community.contains(4:50)</td>
</tr>
<tr>
<td>34</td>
<td>import: from AS3:NEIGHBORS action pref=100; accept NOT community.contains(4:50)</td>
</tr>
<tr>
<td>35</td>
<td>export: to AS3:RESTRICTED announce ANY AND NOT ![AS0 AS4]</td>
</tr>
<tr>
<td>36</td>
<td>export: to AS3:NEIGHBORS announce ANY</td>
</tr>
<tr>
<td>37</td>
<td>aut-num: AS4</td>
</tr>
<tr>
<td>38</td>
<td>import: from AS3 action pref=50; accept ANY</td>
</tr>
<tr>
<td>39</td>
<td>import: from AS2 action pref=100; accept ANY</td>
</tr>
<tr>
<td>40</td>
<td>import: from AS1 action pref=150; accept ANY</td>
</tr>
<tr>
<td>41</td>
<td>export: to AS3 action community.append(4:50); announce ANY</td>
</tr>
<tr>
<td>42</td>
<td>export: to AS4:NEIGHBORS announce ANY</td>
</tr>
<tr>
<td>43</td>
<td>aut-num: AS5</td>
</tr>
<tr>
<td>44</td>
<td>import: from AS6 action pref=50; accept ANY</td>
</tr>
<tr>
<td>45</td>
<td>import: from AS5:NEIGHBORS action pref=100; accept ANY</td>
</tr>
<tr>
<td>46</td>
<td>export: to AS5:NEIGHBORS announce ANY</td>
</tr>
<tr>
<td>47</td>
<td>aut-num: AS6</td>
</tr>
<tr>
<td>48</td>
<td>import: from AS7 action pref=50; accept ANY</td>
</tr>
<tr>
<td>49</td>
<td>import: from AS6:NEIGHBORS action pref=100; accept ANY</td>
</tr>
<tr>
<td>50</td>
<td>export: to AS6:NEIGHBORS announce ANY</td>
</tr>
<tr>
<td>51</td>
<td>aut-num: AS7</td>
</tr>
<tr>
<td>52</td>
<td>import: from AS5 action pref=50; accept ANY</td>
</tr>
<tr>
<td>53</td>
<td>import: from AS7:NEIGHBORS action pref=100; accept ANY</td>
</tr>
<tr>
<td>54</td>
<td>AND NOT !AS6</td>
</tr>
<tr>
<td>55</td>
<td>export: to AS7:NEIGHBORS announce ANY</td>
</tr>
</tbody>
</table>

Figure 4.5: ASes in Fig. 4.4 are assumed to implement the policies described in this fragment of BGP configuration. The fragment is described using RPSL.
4.4. WALKING THROUGH THE OPERATION OF THE CONVERGENCE CHECKER

adopted at other ASes implement the same prefer-customer rule, with a few exceptions. AS3 prefers announcements coming from its sibling AS4 (lines 33-36) because they are marked by AS4 itself with a specific community value (line 43). AS4 assigns an arbitrary ranking to its neighbors (lines 40-42). ASes 5 and 7 always prefer routes through customers (lines 46 and 54), while AS6 arbitrarily prefers one of its providers (line 50).

There are some policies that seem to have ambiguous neighbor specifications: for example, the AS-set “AS3:NEIGHBORS” is used in both the policies at lines 33 and 35; “AS3:RESTRICTED” and “AS3:NEIGHBORS” have some ASes in common (lines 37 and 38); similarly for AS6 and “AS5:NEIGHBORS” (lines 46 and 47), AS7 and “AS6:NEIGHBORS” (lines 50 and 51), AS6 and “AS7:NEIGHBORS” (lines 54 and 55). In all these cases, the specification-order rule [AVG+99] applies to disambiguate the specification. For example, if AS3 receives from a neighboring AS an announcement containing a community 4:50, it only applies the first matching import policy (lines 33-34), despite the existence of other policies that address the same set of neighbors.

Observe that all the policies in Fig. 4.5 but the one implemented by AS 7 are per-neighbor, meaning that each AS applies the same filter to all the announcements coming from a certain neighbor. Also, export policies mostly implement the selective export rules in [Gao01]: AS1 exports to its providers only the direct route to AS0 (line 26); AS2 exports everything to its customer AS1 (line 31) and only its customer routes to its providers (line 37 – see [AVG+99] for the syntax of regular expressions on AS-paths); AS3, AS4, and AS7 may avoid applying any filters, but AS3 and AS7 still do: AS3, with respect to its neighbors AS5, AS6, and AS7, filters out all the announcements that come from AS0 or AS4 (line 37); AS7, with respect to its neighbors AS3 and AS6, filters out all the announcements that have traversed AS6 (lines 55-56). Notable exceptions to the selective export rule are AS5 and AS6, since they allow routes received from a provider to be propagated to another provider.

All the violations to standard customer-provider policies discussed above have been introduced in order to build oscillatory structures in the topology of Fig. 4.4. In particular, ASes 3 and 4 mutually prefer each other, thus building a DISAGREE [GSW02], while ASes 5, 6, 7 each prefer their counter-clockwise neighbor, thus forming a BAD-GADGET [GSW02].

Policy Specifications

Following the architecture in Fig. 4.2, the next step performed by the checker is to parse the configuration in Fig. 4.5 and construct an internal representation
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY

\[
\text{set}(P, A, \text{a, val}) = (P, A') \mid A'[a'] = A[a'], \forall a' \neq a \land A'[a] = \text{val}
\]

\[
\text{append}(P, A, \text{a, val}) = (P, A') \mid A'[a'] = A[a'], \forall a' \neq a \land A'[a] = A'[a] \cup \{\text{val}\}
\]

\[
\begin{align*}
F_{1.0} & = ((P, A)) : \text{true} \Rightarrow (\epsilon, \emptyset) \\
F_{1.1} & = ((P, A)) : \text{true} \Rightarrow ((0), \emptyset) \\
F_{1.2} & = ((P, A)) : \text{true} \Rightarrow (((1)P, A))
\end{align*}
\]

\[
\begin{align*}
F_{3.0} & = ((P, A)) : \\
& \text{true} \Rightarrow \text{set}((P, A), \text{local-preference}, 100) \\
F_{3.1} & = ((P, A)) : \text{true} \Rightarrow \text{set}((P, A), \text{local-preference}, 50) \\
F_{3.2} & = ((P, A)) : \text{true} \Rightarrow ((1)P, A)
\end{align*}
\]

\[
\begin{align*}
F_{6.0} & = ((P, A)) : \\
& \text{true} \Rightarrow \text{set}((P, A), \text{local-preference}, 100) \\
F_{6.1} & = ((P, A)) : \text{true} \Rightarrow \text{set}((P, A), \text{local-preference}, 50) \\
F_{6.2} & = ((P, A)) : \text{true} \Rightarrow ((6)P, A)
\end{align*}
\]


\[
\begin{align*}
F_{7.0} & = ((P, A)) : \\
& \text{true} \Rightarrow \text{set}((P, A), \text{local-preference}, 100) \\
F_{7.1} & = ((P, A)) : \text{true} \Rightarrow \text{set}((P, A), \text{local-preference}, 50) \\
F_{7.2} & = ((P, A)) : \text{true} \Rightarrow ((7)P, A)
\end{align*}
\]

Figure 4.6: Custom representation of the policies in Fig. 4.5. Unspecified announcement attributes are assumed to have their default values.

that is independent of the BGP configuration given as input. Fig. 4.6 shows a possible representation of the policies in Fig. 4.5 in terms of the import and export filters described in Section 4.3. In Fig. 4.6 we first define two handy functions for manipulating BGP attributes: \text{set}((P, A), \text{a, val}) modifies the announcement \((P, A)\) by setting attribute \text{a} to value \text{val}; \text{append}((P, A), \text{a, val}) alters the variable length attribute \text{a} by appending \text{val} at its end (for convenience, we handle \text{a} as a set of values).

Each filter has the following structure: upon matching a predicate on the attributes of a BGP announcement \((P, A)\) (left side of the \(\Rightarrow\)), a sequence of actions is performed and a new announcement is returned (right side of the \(\Rightarrow\)). In Fig. 4.6, all the sequences consist of a single action, and the kind
of action has been annotated in italics on the right margin. Observe that \texttt{local-preference} values have been swapped because a lower \texttt{pref} in RPSL corresponds to a higher \texttt{local-preference} in BGP. If a filter returns $\left(\epsilon, \emptyset\right)$, the announcement is simply dropped.

**Generated (reduced) SPP Instance**

Once the import and export filters are known, an SPP instance that models their effect can be easily built. First of all, the filters need to be analyzed in order to extract the topology of the network under consideration. Since each filter refers to a specific neighbor, we can simply enumerate the neighbors of each vertex to accomplish this task.

After that, the filters are passed to the dissemination algorithm in Fig. 4.3 in order to generate the paths of the SPP instance. Barely running the dissemination on the topology and filters collected from the configuration in Fig. 4.6 would result in the SPP instance in Fig. 4.7a. Despite filter applications being hardcoded in the dissemination process, it is evident that an exponential number of paths still needs to be generated, leading to an overly large instance even in the simple example we are considering.

Fig. 4.7b shows the effectiveness of our dissemination-time optimizations. In particular, early stabilization allows to avoid generating and disseminating all the dimmed paths in Fig. 4.7b. For example, at vertices 1 and 2 the direct path to 0 is the overall most preferred, therefore the announcements of (1 0) and (2 0) are marked as reliable and no other paths are generated at 1 and 2, which are considered stable vertices. As a consequence, none of the paths that would be obtained by the dissemination of the worse path (2 1 0) are available any longer. In this example there are no other vertices that can be early stabilized, yet the gain in terms of generated paths is already noticeable.

On the other hand, early suppression prevents generating all the struck out paths in Fig. 4.7b. For example, vertex 4 will be prevented from disseminating any paths that are worse than (4 2 0), because this path extends the reliable path (2 0). As a consequence, path (3 4 1 0) will no longer be generated at 3. In a similar way, 3 will avoid disseminating paths that are worse than (3 0), which extends the reliable path (0). As a consequence, all the paths that extend (3 2 0) and (3 1 0) are not generated. Note that, after applying the two optimization steps, only the empty path $\epsilon$ is left at vertices 5, 6, and 7. The final SPP instance resulting after applying the optimizations is in Fig. 4.7c.
Figure 4.7: SPP instance constructed from the policies in Fig. 4.6.

(a) SPP instance obtained by running the dissemination algorithm in Fig. 4.3 on the topology and policies extracted from the policy specifications in Fig. 4.6.

(b) Reduced SPP instance obtained by applying early stabilization (dimmed paths) and early suppression (struck out paths) during the dissemination.

(c) Final SPP instance (same as (b) with removed paths).
4.4. WALKING THROUGH THE OPERATION OF THE CONVERGENCE CHECKER

The final step of our convergence checker consists in running on the generated SPP instance the \texttt{Greedy} \texttt{+} algorithm described in Section 4.2. Table 4.2 shows the steps of execution of \texttt{Greedy} \texttt{+} on the instance in Fig. 4.7c. At step \(i = 1\) the candidate vertex 1 enters the stable set because its best path (1 0) uses the stable vertex 0 as next hop. At the same time, paths (3 2 0) and (3 1 0) are removed from the useful paths at 3 because they are worse than (3 0), which will always be available at 3. In a similar way, at step \(i = 2\) vertex 2 enters the stable set because its best path (2 0) has 0 as next hop. At step \(i = 3\) path (4 1 0) is removed from the useful paths at 4 because (4 2 0) will always be available, since 2 is now stable. The algorithm then proceeds by stabilizing vertices 5, 6, and 7, since their only useful path is \(\epsilon\).

At the end we have that all the vertices but 3 and 4 have been stabilized. This means that the original BGP configuration may exhibit oscillations and the cause lies in the policies at 3 and 4. This response is correct, since we intentionally built a \texttt{Disagree} using vertices 3 and 4. Also notice that \texttt{Bad-Gadget}, the other oscillatory structure we put in the configuration, is reported as stable because the paths supporting its oscillation will never be available after the SPVP algorithm (i.e., BGP) has exchanged a few messages. This was, in some way, also suggested by the fact that all the paths at 5, 6, and 7 had been cleared by early stabilization and early suppression.

<table>
<thead>
<tr>
<th>(i)</th>
<th>(V_i)</th>
<th>(C_i)</th>
<th>(\mathcal{P}^i)</th>
<th>(\mathcal{P}^2)</th>
<th>(\mathcal{P}^3)</th>
<th>(\mathcal{P}^4)</th>
<th>(\mathcal{P}^*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0}</td>
<td>—</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0), (3 2 0), (3 1 0)}</td>
<td>{(4 3 0), (4 2 0), (4 1 0)}</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>1</td>
<td>{0, 1}</td>
<td>{1, 2, 5, 6, 7}</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0)}</td>
<td>{(4 3 0), (4 2 0), (4 1 0)}</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>2</td>
<td>{0, 1, 2}</td>
<td>{2, 5, 6, 7}</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0)}</td>
<td>{(4 3 0), (4 2 0), (4 1 0)}</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>3</td>
<td>{0, 1, 2, 5, 6, 7}</td>
<td>{6, 7}</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0)}</td>
<td>{(4 3 0), (4 2 0)}</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>4</td>
<td>{0, 1, 2, 5, 6}</td>
<td>{7}</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0)}</td>
<td>{(4 3 0), (4 2 0)}</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>5</td>
<td>{0, 1, 2, 5, 6, 7}</td>
<td>{7}</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0)}</td>
<td>{(4 3 0), (4 2 0)}</td>
<td>{\epsilon}</td>
</tr>
<tr>
<td>6</td>
<td>{0, 1, 2, 5, 6, 7}</td>
<td>(\varnothing)</td>
<td>{(1 0)}</td>
<td>{(2 0)}</td>
<td>{(3 4 2 0), (3 0)}</td>
<td>{(4 3 0), (4 2 0)}</td>
<td>{\epsilon}</td>
</tr>
</tbody>
</table>

Table 4.2: Execution of \texttt{Greedy} \texttt{+} on the SPP instance in Fig. 4.7c.

(Partial) Stable Path Assignment

The final step of our convergence checker consists in running on the generated SPP instance the \texttt{Greedy} \texttt{+} algorithm described in Section 4.2. Table 4.2 shows the steps of execution of \texttt{Greedy} \texttt{+} on the instance in Fig. 4.7c. At step \(i = 1\) the candidate vertex 1 enters the stable set because its best path (1 0) uses the stable vertex 0 as next hop. At the same time, paths (3 2 0) and (3 1 0) are removed from the useful paths at 3 because they are worse than (3 0), which will always be available at 3. In a similar way, at step \(i = 2\) vertex 2 enters the stable set because its best path (2 0) has 0 as next hop. At step \(i = 3\) path (4 1 0) is removed from the useful paths at 4 because (4 2 0) will always be available, since 2 is now stable. The algorithm then proceeds by stabilizing vertices 5, 6, and 7, since their only useful path is \(\epsilon\).

At the end we have that all the vertices but 3 and 4 have been stabilized. This means that the original BGP configuration may exhibit oscillations and the cause lies in the policies at 3 and 4. This response is correct, since we intentionally built a \texttt{Disagree} using vertices 3 and 4. Also notice that \texttt{Bad-Gadget}, the other oscillatory structure we put in the configuration, is reported as stable because the paths supporting its oscillation will never be available after the SPVP algorithm (i.e., BGP) has exchanged a few messages. This was, in some way, also suggested by the fact that all the paths at 5, 6, and 7 had been cleared by early stabilization and early suppression.
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

4.5 Experimental Results and Applicability Considerations

In Section 4.3 we described the optimization techniques we used to make our tool efficient enough to process huge BGP networks. We also showed the potential of these techniques in a realistic example in Section 4.4. However, it is difficult to analytically assess the effectiveness of these optimizations, since they strongly depend on configurations and network topology.

Hence, in order to validate our approach and to assess its practical applicability, we experimented with a prototype implementation of the convergence checker. Since we are interested in showing that our approach is extremely scalable, in our experiments we focused on eBGP configurations and Internet-scale BGP networks. Regarding the policies we chose to get the policies from the largest publicly accessible source having reasonable worldwide coverage, which is CAIDA [CAI]. Indeed, other sources, like the Internet Routing Registries, are known [DRR06] to contain partial, often inconsistent and out of date information.

In the following, we describe the prototype and discuss the results of our experiments.

Prototype Implementation

We developed a Java-based prototype implementation of the architecture in Fig. 4.2. Our prototype has some limitations that restrict the set of BGP configuration policies it can analyze. In particular: (i) a limited number of BGP attributes is supported, namely only the AS-path, the next-hop, the local-preference and the community, and (ii) filters can only be defined on a per-neighbor basis. Limitation (i) is not really constraining, as the combined use of local-preference and community leads to highly expressive policies, and previous studies show the widespread use of these attributes [DB08]. Also limitation (ii) is not constraining because, on one hand, BGP policies are often analyzed using neighbor-specific models [GR00, SF04, CAI], and, on the other hand, using a finer granularity would add little expressiveness at the expense of manageability and performance, and we argue that these factors would impact network operation, too.

Moreover, it should be considered that our architecture is designed to accommodate every kind of filter. An improved version of the tool, adapted for iBGP and able to process a wider set of BGP attributes, is presented in Chapter 8.
4.5. EXPERIMENTAL RESULTS AND APPLICABILITY CONSIDERATIONS

<table>
<thead>
<tr>
<th>Degree Threshold</th>
<th>Vertices</th>
<th>Edges</th>
<th>Degree Threshold</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10</td>
<td>33</td>
<td>25</td>
<td>592</td>
<td>7403</td>
</tr>
<tr>
<td>500</td>
<td>25</td>
<td>209</td>
<td>10</td>
<td>1657</td>
<td>15672</td>
</tr>
<tr>
<td>250</td>
<td>48</td>
<td>603</td>
<td>5</td>
<td>3735</td>
<td>24388</td>
</tr>
<tr>
<td>100</td>
<td>125</td>
<td>1956</td>
<td>4</td>
<td>5113</td>
<td>28772</td>
</tr>
<tr>
<td>50</td>
<td>268</td>
<td>3953</td>
<td>2</td>
<td>21263</td>
<td>62756</td>
</tr>
<tr>
<td>35</td>
<td>397</td>
<td>5466</td>
<td>1 (complete topology)</td>
<td>33508</td>
<td>75001</td>
</tr>
</tbody>
</table>

Table 4.3: Topologies used in our tests, obtained from CAIDA topologies by pruning vertices with degree lower than a threshold.

Performance and Scalability

We ran our experiments using the AS-level topologies from CAIDA [CAI] as input. 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 Jan 20th, 2010 a set of smaller topologies by pruning vertices with degree lower than a threshold. We picked thresholds in the values listed in Table 4.3, which also shows the number of vertices and edges of the obtained topologies. The last line (threshold 1) corresponds 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 our experiments we assumed to originate a prefix from a given AS picked from a sample of 200 ASes having very different degree values. In particular, we picked the top 100 and the bottom 100 ASes according to CAIDA’s ranking algorithm, which ranks ASes based on the size of their customer cone (number of direct and indirect customers). Since there is a high correlation between the customer cone and the position of an AS in the Internet hierarchy, essentially our sample is composed of ASes ranging from tier-1 providers to small stubs. The median degree of the ASes in our sample is 8, the average degree is 201, and 20% of ASes have a degree higher than 264.

Our testing platform was a dual Xeon 2.66GHz with 16GB of RAM. We ran the checker once for each combination of pruned topology and originator AS.
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

From the theoretical 2,400 runs we had to exclude the cases when the originator AS itself was removed by the pruning, dropping to 1,099 runs. For each run, we assigned 4GB of RAM to the Java VM hosting the checker. Observe that, despite the amount of memory reserved for the tool, there were cases in which the checker ran out of memory because of the excessive number of paths to be generated for the SPP instance. Every time this happened for a certain originator AS and a certain threshold, we avoided attempting the check with the same originator AS on topologies with lower threshold. Thus, we boiled down to 540 successful runs. Note that, in many cases, the SPP instances can only be generated using both our optimizations (Greedy+): with early stabilization alone (i.e., Greedy) we could achieve only 105 successful runs.

The convergence check took a fraction of a second to complete in 24% of the successful runs, and 16 seconds on average. The median of running times was of 2 seconds, while the maximum was 13 minutes, recorded in 1 run only. As a term of comparison, running our implementation using early stabilization alone (i.e., Greedy) resulted in up to 64 minutes of computing time. We could not find any correlations between the running time and other topological features such as the pruning threshold and the originator AS degree. These results already prove that our approach outperforms the state of the art: it can successfully check a larger number of Internet-scale topologies and achieves this in a very short time. These performance results show that the tool can be used for online convergence checks performed right after a policy change.

Figs. 4.8 and 4.9 show the effectiveness of the optimizations in Greedy+. Both figures refer to the experiments that considered the top 100 ASes as originators. In order to assess the feasibility of convergence checks for Internet-scale BGP configurations, we associated an arbitrary value of 5M paths to each path generation that ran out of memory. Fig. 4.8 plots the median number of paths in the set $P$ of the generated SPP instance. Each point corresponds to a value of the degree threshold we used to prune the CAIDA topology. We used all the values in Table 4.3 down to 2, since degree-1 vertices cannot make the network unstable. It is clear that using both our optimizations (greedy+) defeats early stabilization alone (greedy) and generation of paths without optimizations (naive). Note that, starting from threshold 100, the optimizations in Greedy+ are necessary to successfully generate the SPP instance. The plot for Greedy+ exhibits some irregularities (e.g., for degree threshold 35): we ascribe this to the fact that generated paths are highly dependent on the topology, since the presence of specific vertices can cause a large number of additional paths to be generated. Fig. 4.9 shows the cumulative distribution function of the number of generated paths, on the topology obtained with de-
4.5. EXPERIMENTAL RESULTS AND APPLICABILITY CONSIDERATIONS

Figure 4.8: Median number of generated paths, considering the top 100 ASes as originators. The plots show the values without optimizations (naive), with early stabilization (greedy) and with early stabilization and early suppression (greedy+). The X axis shows the threshold used to prune CAIDA topologies.

Figure 4.9: Cumulative distribution function of the number of generated paths, considering the top 100 ASes and fixing the degree threshold at 2.
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

Figure 4.10: This triple of vertices, on which C-BGP was unable to converge, has been pinpointed by our checker as potentially oscillating. On the left we show the commercial relationships between the ASes. On the right we show the propagation of BGP announcements and the local preference at each vertex.

gree threshold 2. Observe that early stabilization alone (greedy) allows us to successfully generate the paths for less than 20% of the cases. On the other hand, using both our optimizations (greedy+) allows us to successfully perform the check on more than 50% of the topology-originator pairs, and for 40% of them the check requires generating less than 200,000 paths. Moreover, in 50% of the successful runs our checker generated less than 1% of the paths generated using early stabilization alone (greedy).

The results obtained with the bottom 100 ASes in the CAIDA ranking show similar trends (plots are omitted for brevity) but different absolute values. We were able to successfully perform the check on 38% of the topology-originator pairs. This shows that the performance of our technique degrades when the originator AS is placed at the bottom levels of the Internet hierarchy, which is possibly due to the fact that stub ASes are reachable from the Internet through a very high number of possible paths. However, we stress that the optimizations of GREEDY+ greatly benefit from the presence of peer-to-peer links, which are typically not captured in AS-level topologies (see, e.g., [DKF+07]).

Interestingly, compared with similar experiments we performed few years ago [CRCD09], our optimizations are still effective, despite the larger topology (26% more vertices and 39% more adjacencies) we used as input.

Spotting Potential Oscillations

We finally looked at the percentage of vertices of the input topology that our checker reported as safe. Interestingly, depending on the originator AS, our checker reported up to 5% of vertices as potentially unstable.

We further investigated the portions of the network which our checker reported as potentially unstable. In a separate experiment based on data used in [CRCD09], our prototype was able to spot a triple of vertices that, if con-
4.6. RELATED WORK

Routing convergence is renowned to be a fundamental problem in network routing [FBR04]. As discussed in the previous chapter, deciding whether a BGP network is stable is a computationally hard problem, and a bunch of sufficient conditions to guarantee stable routing have been found. Based on these results, two research directions have been explored.

On one hand, several modifications to the BGP protocol have been proposed to dynamically detect and solve policy-induced oscillations (e.g., [GW00, ERC+07]). Also, in [GS05] the authors introduce formal tools for the design of inherently stable protocols. However, there are currently serious difficulties in deploying substantial changes to BGP while guaranteeing service continuity. By contrast, in this chapter we aim at checking whether a given BGP network is stable by simply analyzing static properties of its configuration, without considering protocol dynamics.

On the other hand, few techniques are available that address the convergence problem considering the current implementation of the protocol. We have already discussed and experimentally evaluated practical advantages of Greedy+ with respect to Greedy in Sections 4.2 and 4.5.

In [GSW99], a correct and complete algorithm has been proposed to check...
CHAPTER 4. A HEURISTIC-BASED APPROACH TO STABILITY TESTING

for the absence of dispute wheels in a BGP configuration. We recall that the absence of dispute wheels is a sufficient but not necessary condition for safety. Indeed, the Di-safe-gree instance (see Fig. 4.1) is safe, despite the presence of a dispute wheel in it. Although algorithm in [GSW99], Greedy\(^+\) is able to solve it. Also, the technique proposed in [GSW99] is based on a data structure called dispute digraph. Starting from an SPP instance \(S\), time complexity of constructing the corresponding dispute digraph is \(O(p^2) = O(n^4)\), where \(p\) and \(n\) are the number of paths and the number of vertices in \(S\), respectively. Greedy\(^+\) is definitely more scalable, since its time complexity is \(O(n)\), where \(n\) is the number of vertices in the SPP instance.

A heuristic that performs convergence checks of iBGP configurations is presented in [FRBS08]. However, the algorithm assumes that the network under consideration contains only one layer of route reflectors, and cannot be easily extended to the case of eBGP configurations. Our approach is more general in both respects.

A generalization of the technique in [FRBS08] is proposed in [FMS\(^+\)10]. Such a generalization is based on the reliance graph concept. A reliance graph is a data structure similar to a dispute digraph, which is built according to the iBGP route reflection rules. Basically, reliance graphs are exploited to look for dispute wheels. However, Greedy\(^+\) can solve a larger set of instances with respect to the heuristic in [FMS\(^+\)10], since the presence of a dispute wheel does not imply a convergence problem. Indeed, contrary to Greedy\(^+\), the algorithm in [FMS\(^+\)10] is not able to solve simple variations of the Di-safe-gree gadget. An example of such gadgets is depicted in Fig. 4.11. Also, adapting the technique to configurations different from route reflection (e.g., BGP confederations) is not straightforward.

Existing configuration checkers (e.g., [QN04, FB05]) typically execute syntactic checks and batch tests on BGP configurations. Our approach is complementary in that we explicitly focus on convergence, which also requires analyzing configuration semantics. We also overlap simulators [QU05], in that we are able to point out the converging portion of networks that could permanently oscillate. For these reasons, we believe our technique can effectively integrate existing checkers and visual analysis tools (e.g., [CDM\(^+\)05]) to assist operators in verifying configurations.
4.7 Conclusions

In this chapter we showed that an automated check for BGP convergence is feasible in practice. We described a heuristic algorithm (Greedy+) that can be used to check the convergence of BGP in the SPP model. We proved that this algorithm has several desirable properties, among which the ability to avoid false positives, i.e., configurations mistakenly reported as safe while they are not. We used the Greedy+ algorithm as the basis for an automated tool that is able to statically check BGP configurations (both eBGP and iBGP) for guaranteed routing convergence. The tool models the configurations using the SPP formalism. Since this choice may lead to an intractably large representation to be handled, we proposed optimizations that make it feasible to run the convergence checker, even online, on huge BGP networks. We evaluated our approach and its scalability by performing experiments on AS-level Internet topologies from CAIDA. During such experiments, we spotted potentially oscillating portions of networks that cause state-of-the-art simulators (C-BGP) to loop indefinitely. Our results show that Greedy+ outperforms the well-known Greedy algorithm [GSW02]. We plan to test Greedy+ algorithm and our prototype tool on real-world configuration as part of our future work.
Part III

Monitoring
Chapter 5

Control-Plane Monitoring

5.1 Introduction

ISP business mainly relies on forwarding paths on which customer traffic flows. For inter-domain traffic, BGP has the final say on routing decisions, and BGP messages received from neighboring ISPs can have a dramatic impact on the quality of service actually provided by an ISP. Thus, BGP monitoring enables ISPs to perform business-critical activities like troubleshooting and anomaly detection [MYC08, RGMM+04]. Recently, it has been shown that BGP data can be also exploited for business intelligence [Gao01], traffic engineering [BL08], root cause analysis [CCD+08, FMM+04], oscillation detection [FRBS08], routing table analysis [Hus01] and Service Level Agreement (SLA) 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 border 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 analyses. 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. Unfortunately, these drawbacks prevent exploiting the monitoring system for interesting applications like fine tuning of ingress policies, verification of SLAs and analysis of what-if scenarios (e.g., what if one of my providers goes down?). Recently, the BGP Monitoring Protocol [SFS10] has been proposed to overcome those limitations, but it is still experimental and requires software support on the routers.

In this chapter, we define a novel technique for building a BGP monitoring system that overcomes limitations of state of the art solutions. We start discussing, in Section 5.2, realistic scenarios demanding for an improvement in existing BGP monitoring architectures. In Section 5.3, we define a set of requirements that an ideal BGP monitoring system should satisfy so that an ISP can take the biggest possible advantage from its deployment. In Section 5.4, we describe our proposal for a BGP monitoring system, outlining its architecture and discussing the most relevant components. Our approach enables real-time, non-invasive and scalable collection of all BGP messages received by BGP border 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. After possibly reordering out-of-order segments, our collector parses the BGP messages and stores them in the standard MRT format [BKL09]. Based on the requirements defined in Section 5.3, we evaluate our technique in Section 5.5. By means of experiments performed 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, packet latency and router CPU usage. We also show that our prototype implementation can monitor hundreds of BGP routers on commodity hardware. We also check the accuracy of the collected data.

We compare our approach with existing solutions in Section 5.6, showing that our solution better fulfills the requirements we identify for an ideal monitoring system. Among the most important advantages of our proposal, we also stress its flexibility. Indeed, the same approach can be adopted to monitor other signaling protocols. In this light, we consider this work as a first step towards a centralized monitoring solution for the whole control plane.

Finally, conclusions are drawn in Section 5.7.
5.2. MOTIVATING SCENARIOS

In this section, we show simple scenarios in which existing monitoring techniques cannot be exploited to perform business-critical activities, as they do not support monitoring of all BGP routes and collection of BGP messages before ingress policy application.

What-If Analyses

We start by analyzing a case in which monitoring all BGP messages is necessary for performing what-if analyses. We refer to the routes received by at least one border router but not picked as best (hence, not propagated further inside the AS) as non-best routes.

Consider the scenario depicted in Fig. 5.1. An ISP X configures one of its border routers to peer with two different upstream providers A and B. For economic reasons, X always prefers to send traffic to ISP A, and it is willing to use the peering with B only as backup. To respect this policy, the router is configured to assign a higher value of the local-preference attribute on the route announcements received from A with respect to those received from B. In this setting, collecting non-best routes enables X to perform analyses of what-if scenarios, answering questions like the followings.

- What if the link with the upstream provider A goes down? Are all the Internet prefixes reachable through B?
• What happens to paying traffic if I set different values to some BGP attributes (e.g., local preference or MED) on routes announced by ISP A?

• What is the effective redundancy provided by the backup link? Does B provide different paths to destination prefixes with respect to those provided by A? Can a single failure on the Internet affect the capability of X to reach some destination using both links with A and B?

SLA and Quality Monitoring

Monitoring only best routes prevents the BGP monitoring system from detecting potential SLA violations and assessing the quality of the service offered by upstream providers.

Refer again to Fig. 5.1. Assume that X has a SLA with both its providers stating that a path for any prefix in the full Internet routing table must be available during 99.9% of the time. In this case, it would be impossible to verify whether B complies with signed SLA, unless non-best routes are collected. Indeed, non collecting non-best routes would provide information about provider A only.

Moreover, collection and analysis of non-best routes is necessary to monitor the quality (e.g., in terms of AS-path length) of the routes announced by both A and B. Observe that quality monitoring can also trigger refined business-intelligence activities, and help answering questions like “Is it convenient for X to keep the link with A as primary or it is better to send the traffic relative to some or all the prefixes to B?”.

Historical Data

BGP messages should be collected as they are before border routers apply input policies.

Consider the scenario in which an AS X decides to maintain historical BGP data for several months, in order to perform some analyses on the updates received from its providers over time. In this case, a single change in the policy applied by AS X on one of its border routers would invalidate the entire historical dataset, since data collected in the past would be no longer consistent with the new one. Indeed, in this case, differences in the attribute value within some BGP message could be caused by a change in the policy enforced by X or a change in the BGP announcements sent by one X’s neighbor or both.
Notice that, even if the routing policy locally applied by the collector ISP is known, the original BGP messages sent by neighboring ISPs can not be reconstructed in the most general case, since policies are not always reversible (e.g., if an attribute gets overwritten).

5.3 Requirements for a BGP Monitor

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

- **Collection of non-best routes updates.** BGP routers select a single best route among a set of candidates. Although non-best routes have no impact on where packets are forwarded, keeping track of them allows an ISP to better engineer its traffic and analyze what-if scenarios.

- **Policy independent data collection.** An ideal collection system should reconstruct the original BGP messages as sent by neighboring ISPs, without being affected by the locally configured policies. This allows ISPs to decouple BGP data from BGP policies, so that policy changes cannot affect the consistency of historical data.

- **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 applicative analysis within few seconds. This is a crucial requirement for network management applications: network administrators want to know what is going on while it is going on, not hours later.

- **Low impact on router resources.** A typical constraint on management systems is to have a small impact in terms of extra resource demand (e.g., CPU usage, throughput and bandwidth) on the network infrastructure. This is especially true for BGP monitoring, given that BGP border routers typically have to forward huge amounts of traffic.

- **Cost-efficient deployment.** To be realistically deployable in large networks, the monitoring system should be able to handle hundreds of border routers employing few machines equipped with commodity hardware.
5.4 Proposed Architecture and Implementation

We now propose an architecture for a BGP monitoring system that aims at satisfying all the requirements listed in Section 5.3. The key 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 MRT format [BKL09]. We show that this technique can be implemented using a feature commonly available on routers together with ad-hoc software employed on the collector side.

Fig. 5.2 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 peerings at public Internet eXchange Points (IXPs).

Fig. 5.2 highlights the role of the two main architectural components: the border router (BR) and the route collector (RC). We provide details on each component in the remaining of the section.
5.4. PROPOSED ARCHITECTURE AND IMPLEMENTATION

Border Routers: Cloning BGP Traffic

The majority of ISP-targeted commercial routers provides the feature to clone IP packets and send copies to a remote machine. This is mostly used for copying traffic to Intrusion Detection Systems [cisd]. Leading vendors also provide filtering capabilities that allow operators to specify which packets must be cloned. To maintain a vendor-independent terminology, we will refer to this feature as Selective Packet Cloning (SPC). An SPC-enabled BR copies the packets received from user-specified source interfaces and matching an optional filter to another interface, which we call 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 a tunnel interface (e.g., IP-in-IP encapsulation or Generic Routing Encapsulation).

We now 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 [cisd]. 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 [cisa], 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 [junb]. Traffic received via user-specified ingress interfaces can be cloned and forwarded over a VLAN or a tunnel (IP-in-IP or GRE) interface.

Enabling SPC feature on BRs requires a very small amount of extra configuration. For example, Fig. 5.3 shows how to configure RITE on Cisco routers. 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 from BRs to the RC which decodes and stores BGP messages. The RC performs the following activities, as summarized in Fig. 5.4.

- **Packet reception.** The RC receives cloned packets and buffers them
CHAPTER 5. CONTROL-PLANE MONITORING

RITE Configuration Steps

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

```bash
7201(config)#access-list 100 permit tcp any any eq bgp
```

Step (ii) - Define a destination interface

```bash
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

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

Figure 5.3: Steps for configuring SPC on Cisco routers.

Figure 5.4: Main activities performed by the route collector software.

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 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.

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

In this section, we evaluate the extent to which our proposal meets the requirements we defined in Section 5.3. In particular, we measured performance, accuracy and scalability of the proposed monitoring system.

In all the experiments we ran, we found that no cloned packet was dropped and BGP messages were always correctly reconstructed and stored on disk. Hence, we focus on the performance degradation at the BRs and on the scalability of the RC component.

Border Router Performance

We evaluate the router load in terms of frame loss (throughput), average CPU usage, and average packet latency. In our experiments, we used a Cisco 7201 router, referred to as device-under-test (DUT) in the following. 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 one of the cheapest router targeted to ISPs.

Since the performance of the DUT is highly dependant on the total amount of traffic it has to route, we evaluate the impact of SPC features in different scenarios. In the following, we describe each scenario, discussing the results of our tests.

Baseline Measurement

First of all, we measured the performance of the device without any special configuration. We use the results of this experiment as a baseline for evaluating
118

CHAPTER 5. CONTROL-PLANE MONITORING

Figure 5.5: Baseline test topology.

the impact of enabling the SPC feature on the DUT.

Fig. 5.5 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.

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 different packet transmission rates. Results are summarized in Fig. 5.7, 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 (near 60% of the maximum packet rate) experiencing a negligible frame
5.5. EVALUATION

loss (less than 0.01%). The router was not able to handle the two million packets per second that the vendor’s datasheet claims (vertical dashed line in Fig. 5.7) 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.

Single Peering Scenario

After having performed the baseline measurement described in the previous section, we evaluated router performance in a single BGP peering scenario. Namely, we set up a testbed using the topology depicted in Fig. 5.6.

The DUT was connected to the traffic generator as in the baseline experiment. Also, the DUT 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 310,000 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 5.5, 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 enabled (test “BGP-updates-mirror”) and

![](image)

Figure 5.6: Topology in the single peering scenario.
disabled (test “BGP-updates-no-mirror”). Results are presented in Fig. 5.7. 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 essentially has no impact on the throughput achieved by the router. 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 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.

### Five Peerings Scenario

To verify the impact of enabling SPC in a more realistic situation, we performed more in-depth tests in the five peerings scenario.

Fig. 5.8 shows the topology of this scenario. We interposed five BGP daemons (i.e., five Quagga [Ish] processes) between DUT and the ISP, in order to amplify the original stream five times. Each BGP daemon had a peering session with the ISP and one iBGP peering session with the DUT. This way, whenever a BGP update was sent by the ISP’s BGP router, each BGP daemon sent an update to the DUT. The configuration of all the devices is analogous to that we used in the single peering scenario, the only difference being the exploitation of the BGP third-party next-hop mechanism instead of the static routes at the DUT.
5.5. EVALUATION

In this scenario, we measured packet loss, average CPU usage and average latency when the router is solicited by the traffic generator with traffic at increasing packet rates. We ran tests both with SPC enabled and disabled and then compared the results. Since packet rates higher than 60% causes the router to drop non-negligible amounts of frames, we do not report results of the tests made for higher packet rates. We structured our tests in iterations of 5 minutes each, during which the SmartBits sends traffic to the DUT at the same packet rate. We chose this threshold estimating the interarrival time between BGP update bursts during daily hours of week days.

Fig. 5.9 reports the results of our tests. The $y$-axis shows the difference (expressed as a percentage) between the performance of the router when SPC is enabled with respect to when it is disabled. The $x$-axis represents packet rate. It is easy to see that activating the SPC feature essentially has no impact on the frame loss and on the average latency. The worst latency we recorded was 375 $\mu$s with SPC enabled and 301 $\mu$s with SPC disabled.

Differences for CPU load are small and highly dependant on the presence of BGP bursts. Anyway, activating SPC never affected CPU load for more than 2%.

Update Bursts Scenario

We set up another experiment to evaluate how SPC affects the performance of the BRs under heavy BGP update bursts. The topology of the testbed is the same of the previous experiment (see Fig. 5.10).
CHAPTER 5. CONTROL-PLANE MONITORING

---

**Figure 5.9:** Performance degradation induced by SPC in the five peering scenario.

**Figure 5.10:** Test topology in the update bursts scenario.
In this experiment, however, we tore down and restored the BGP session with the ISP’s BGP router, at regular intervals of 1 minute each. This way, we were able to generate huge amounts of BGP updates. In fact, every time the BGP session was tore down (restored), the entire Internet full routing table was withdrawn (reannounced) by each of the five BGP daemons, and the DUT received almost 1.5 million BGP route withdrawals (announcements). Moreover, we configured our traffic generator to send a considerable amount of traffic; namely the SmartBits generated traffic at the 45% of the maximum packet rate obtainable on a full-duplex Gigabit Ethernet.

We highlight that such a scenario is unrealistic. Indeed, we stressed the DUT with traffic sent at the 75% of the maximum throughput that can be achieved by the device and routers of an ISP should not be (and typically are not) so overloaded by regular traffic. Moreover, real-world routers typically 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”). Fig. 5.11 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 DUT lost a very small fraction of traffic, about 0.001%, when working with SPC disabled, as shown by the red solid line
in Fig. 5.11. 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 small extent, as it is evident by observing that packet loss never exceeds 0.01%.

We also ran a 5 minutes experiment sending traffic at the 45% of the maximum packet rate, while repeatedly tearing down and bringing up the BGP peering with the ISP’s router. We measured that, in these conditions, the router dropped less than 0.005% of packets even when SPC was enabled on it.

Performance of the Collector Software

From a theoretical point of view, scalability of the RC component should not be a problem. We now discuss the main factors that can affect the scalability of the collector software.

- **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
5.6. RELATED WORK

across 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.

We also performed some experimental tests to assess the amount of resources actually required on the RC side. During these tests, we separately measured the processing time needed for receiving the packets, reconstructing the TCP stream, decoding BGP messages and storing them in MRT format. All tests were performed 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 these experiments provides an upper bound on the performance that can be achieved by a RC, since processing times can be greatly improved by enabling pipelining and parallel processing, as all the activities are trivial to parallelize across multiple processors.

We now present results of our experiments, reporting average values. 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 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. Re-playing the capture file with tcpreplay took 3.38 seconds, while originally the BGP sessions lasted slightly less than 2 minutes and a half. A regular BGP session can reach such a high speed just sporadically. Even in this extreme experiment, we were able to capture all the 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 per second, our prototype implementation can handle hundreds of BRs on commodity hardware.

Clearly, multiple route collectors can be deployed on the same network. However, given our experimental results, we expect that even tier-1 ISPs will only need a handful of collectors.

5.6 Related Work

Two naive approaches for BGP monitoring can be enabling debug option on router devices and installing optical taps and ad-hoc filtering boxes near
each BR. However, debugging output stream “might render the system unusable” [cise]. On the other hand, installing an optical tap and a dedicated filtering box for each optical fiber of each BR would be too expensive.

Existing Solutions

Existing approaches can be broadly classified in two categories: those employing some kind of route collectors to which BGP messages are pushed 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 essentially consists in a route collector, deployed inside the network, that is configured to maintain iBGP peerings with every BR. Quagga [Ish], OpenBGPd [BJ], BIRD [Fil] and PyRT [Mor] are probably the most famous and widespread tools to set up a route collector this way. Essentially, Quagga, OpenBGPd, and BIRD simulate the behavior of real routers, but they also support dumping BGP messages in MRT [BKL09] format. The Python Routing Toolkit (PyRT) [Mor], on the other hand, only implements a minimal set of features, and is more lightweight and scalable.

BGP monitoring systems based on separate management protocols are designed to pull information from routers. In particular, 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) [SFS10]: it is based on the idea of sending received BGP messages via a TCP connection with a monitoring station.

Comparison with Related Work

Table 5.1 summarizes the main differences between our approach and existing solutions. In the following, we discuss them in more detail.

- **Collection of Non-Best Routes** Since Quagga, OpenBGPd, and PyRT rely 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. BMP and
5.6. RELATED WORK

<table>
<thead>
<tr>
<th></th>
<th>Quagga</th>
<th>PyRT</th>
<th>SNMP</th>
<th>BMP</th>
<th>SPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-best route collection</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>policy independent data collection</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>real-time data collection</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>almost</td>
<td>yes</td>
</tr>
<tr>
<td>impact on router resources</td>
<td>very low</td>
<td>very low</td>
<td>heavy</td>
<td>very low</td>
<td>very low</td>
</tr>
<tr>
<td>cost efficient deployment</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison between our solution and related work with respect to the requirements defined in Section 5.3.

the solution we present in this chapter are currently the only way to continuously monitor non-best routes.

- **Policy Independent Data Collection** Quagga, Pyrt, and OpenBGPd can only monitor routes selected as best, and they are forced to collect BGP messages after ingress policy application. On the contrary, polling-based mechanisms typically provide a way to gather BGP messages as they are before BGP filters are applied (see [HH06] for SNMP based mechanisms). Both BMP and our approach also allow an ISP to collect policy independent data.

- **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, 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: their usage is restricted to periodic snapshots of BGP routes received by BRs. The current BMP specification asserts that BMP messages “are not real time replicated messages received from a peer” [SFS10]. Section 5.5 shows that our approach can collect data in near real-time.

- **Low 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 monitor. Our experimental tests show that our approach affects the performance of the BR only minimally, see Section 5.5. We expect that also BMP has a low impact on router resources in most of the cases. See Section 5.6 for a more detailed comparison between BMP and our solution.

- **Cost efficient deployment** 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. PyRT is not affected by this problem since it only implements a minimal set of features, 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 5.5 ensures that our approach and, reasonably, also BMP can handle hundreds of BRs on a single RC.

**Comparison with BMP**

Section 5.6 highlights that only our approach and BMP can reasonably be used in a monitoring system which aims at satisfying all the requirements listed in Section 5.3. However, BMP is not yet standardized and, currently, only JunOS versions later than 9.5 support BMP.

The main technical difference between BMP and our approach is that BMP relies on TCP while our solution forwards packets from BRs to the RC over IP tunnels or VLAN. Our solution is based on enabling SPC at the BRs and does not need any additional daemon to be run. SPC involves only switching capabilities (either software or hardware) which are usually highly optimized. On the contrary, BMP is not implementable using switching mechanisms, must rely on conventional TCP implementation, and usually requires an additional daemon.

Adopting TCP, BMP guarantees reliable delivery of copied BGP messages to the collector. However, 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 congestion window. Our proposal does not
5.7. CONCLUSIONS

mandate the router to maintain any state. Observe that RCs can easily check whether some TCP segments are missing by analyzing sequence numbers of cloned traffic.

Essentially, our approach pushes as much complexity as possible to the collector. The benefits are twofold: on one hand, a simpler router-side component results in precious resource savings; on the other hand, our solution is easy to extend to monitor other control-plane protocols than just BGP without requiring changes on routers.

5.7 Conclusions

In this chapter, we defined requirements for an ideal control-plane monitoring system. We especially referred to BGP monitoring, since it can allow ISPs to take better high-level economic decisions about peerings and commercial agreements as well as to improve troubleshooting and other business-intelligence activities.

Given the limitations of existing solutions, we proposed a new technique for real-time collection of all BGP messages sent by BGP peers, before the application of input policies. Through several experiments, we showed 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.

Among the most important benefits of proposed architecture, we recall (i) leverage of efficient features, available on today’s routers, for selectively cloning packets to a central monitoring station, and (ii) the possibility to extend our approach to monitor other control-plane protocols.

As future work, we plan to improve our collector prototype (e.g., for supporting a bigger set of protocols) and deploy our solution in real networks.
Chapter 6

Leveraging Router Programmability for Traffic Matrix Computation *

6.1 Introduction

After having tackled the control-plane monitoring problem in Chapter 5, we now turn our attention to data-plane monitoring. In particular, we focus on efficient and accurate computation of traffic matrices.

A traffic matrix (TM) quantifies the amount of traffic traversing a network from every ingress point to every egress point in a given time interval. This provides operators with a measure of the actual bandwidth demand. TMs are practically used by Internet Service Providers (ISPs) as an input for several business-critical activities related to the network design, e.g., capacity planning, provisioning and traffic engineering [PTL04]. Computing TMs in real-time would also enable anomaly detection and on-line diagnosis of routing events and device failures [ZGWX06].

Computing TMs, however, is a challenging task, since direct measurement is unpractical [ZGWX06], packet sampling forces a trade-off between the overhead imposed on network devices and the accuracy of the TM, and estima-

CHAPTER 6. LEVERAGING ROUTER PROGRAMMABILITY FOR TRAFFIC MATRIX COMPUTATION

Traffic techniques rely on statistical assumptions and cannot be arbitrarily accurate [MTS+02].

In [VE04], Varghese and Estan envision a new measurement approach, which can lead to a superior trade-off, by carefully evaluating implementation costs, understanding real needs, and leveraging other system parts. We believe that this challenge can now be tackled exploiting router programmability features recently made available in commercial products. Following this idea, we propose a new architecture for directly measuring TMs, founding on the possibilities opened by router programmability. Because of their practical importance for ISPs [BDJT01], we focus on PoP-to-PoP TMs, that account of traffic between each pair of Points of Presence of an ISP. However, our solution can be extended to other types of TMs. Contrary to existing approaches, we develop a distributed solution for the TM computation. Indeed, our approach is based on the possibility to program routers to autonomously compute parts of TM and a central component is only used to trivially combine precomputed data and display the TM. This allows us to limit the total overhead [PTL04]. Our architecture can be implemented using current technologies and avoids the need for packet sampling since it leverages highly optimized packet counting features already available on commercial routers.

In order to show the feasibility of our solution, we prototypically realized our idea on a Juniper router and experimentally evaluated our prototype in a testbed. Preliminary experiments show that our architecture has the potential to enable accurate computation of TMs with a tunable time granularity and limited router load.

The rest of the chapter is organized as follows. In Section 6.2, we describe existing solutions for computing TMs along with their major limitations, further motivating our research effort. In Sections 6.3 and 6.4, we describe our proposed architecture and our prototypical implementation, respectively. In Section 6.5, we report the results of our experimental evaluation. Finally, we conclude in Section 6.6.

6.2 State of the Art

Because of their importance, TMs and TM computation attracted huge industrial and research effort. Different approaches have been proposed and evaluated over the years.

Direct measurement of traffic data on network devices is probably the most straightforward approach to compute TMs. Unfortunately, this approach
is generally considered unpractical, or even unfeasible, because of the huge amount of data to be managed and the lack of an appropriate measurement infrastructure [ZGWX06].

To mitigate these problems, sampled flow data are typically collected on routers and combined with routing information. In particular, Cisco NetFlow v9 [net] allows collection of flow records that also include some routing information (e.g., the BGP next-hop address). Techniques based on sampled flow statistics are presented in [FGL00, PTL04]. However, packet sampling forces a trade-off between the overhead imposed on network devices and the accuracy of the TM.

Estimation techniques based on indirect measures, e.g., aggregated link loads that can be collected using SNMP [CFSD90], are proposed in literature as an alternative approach. Most of these techniques are based on mathematical and statistical methods and can combine multiple data sources (e.g., SNMP and NetFlow data, BGP routing information, etc.) [MTS02, ZRDG03, ZGWX06]. However, this approach generally relies on statistical assumptions and periodic snapshots of routing information, resulting in a final estimation whose accuracy cannot be arbitrarily high (error is typically about 10%) and also depends on the absence of routing changes [MTS02, ZGWX06, VE04].

A simulation approach is proposed in [UQLB06].

All the solutions listed above are centralized in that they require a central component that gathers all necessary data and computes, or estimates, the TM. A distributed solution is advocated in [PTL04] in order to improve efficiency and lower total overhead. In this chapter, we propose an architecture that enables the TM computation to be distributed, relying on current technologies.

An approach to directly collect statistics on routers is proposed in [XHB00], but the technique can only be applied to MPLS networks. More recently, a new approach to directly measure TMs, based on accounting capabilities of switches, is proposed in [TGG10]. However, its applicability is limited to OpenFlow networks and the usage of per flow counters would not scale to large ISP networks.

Our solution philosophically conforms to the general proposal described in [VE04], where authors suggest to map prefixes into equivalence classes (according to the BGP next-hop) and set up per-class counters. However, leveraging router programmability, we do not require any support from routing protocols. This makes our solution readily deployable.
CHAPTER 6. LEVERAGING ROUTER PROGRAMMABILITY FOR TRAFFIC MATRIX COMPUTATION

6.3 Leveraging Router Programmability

Nowadays, the vast majority of commercial devices implements a counting mechanism (CM) that can be configured to selectively count traversing traffic. Such a counting mechanism allows operators to define a set of counting rules, each responsible for incrementing a specific counter when traffic satisfies a given matching condition. Being involved in packet forwarding, this mechanism is tightly integrated within the data plane and is highly optimized [SDEK]. CMs usually track both number of packets and bytes.

A CM can be activated on real-world devices through simple configurations. E.g., it can be enabled on Juniper routers using the count statement within an appropriate firewall filter. On Cisco routers, statements policy-map and class-map can be used for the same purpose. Current router architectures do not allow arbitrary matching condition to be specified. Indeed, CMs can match traffic only based on the information that is known by the data plane, e.g., the IP next-hop, the IP destination address and other fields of the IP packet. Unfortunately, computing a PoP-to-PoP TM requires to correlate, at each ingress point, the destination address inside IP packets with the BGP next-hop attribute [RLH06] as specified in the BGP Routing Information Base (RIB) of traversed routers.

Since collecting per-prefix data does not scale up in common scenarios [VE04], plain usage of counting mechanisms is not viable for computing TMs. We propose to conveniently configure the CM on routers located at each PoP, grouping IP flows on the basis of the BGP next-hop. Router programmability is exploited to dynamically adapt counting rules to routing changes.

Architecture

Fig. 6.1 shows our proposed architecture and the interaction between components. Router programmability allows us to add new modules to the standard modules of a router.

Among standard modules, the BGP routing daemon computes the association between each IP prefix \( p \) and the BGP next-hop to which traffic to \( p \) has to be forwarded. Also, the counting mechanism (CM) is used as a primitive in order to install appropriate counting rules on the router.

Counting rules to be installed are defined by the counting rules computing module (CRCM), which is new with respect to the standard router modules. The CRCM configures the CM as follows. One counter per BGP next-hop is set on the router. The matching condition of each counting rule \( R \) matches the
6.3. LEVERAGING ROUTER PROGRAMMABILITY

IP destination address of traversing packets with the IP prefixes associated to a specific BGP next-hop $NH(R)$ in the RIB. The action performed by each counting rule $R$ in case of a match is to increment the counter associated to $NH(R)$. Note that the CM typically ensures that only the rule matching the most specific IP prefix is applied, hence it is coherent with the longest prefix match rule applied for packet forwarding.

The most important task of the CRCM is to guarantee consistency between counting rules and BGP routing information over time. For this purpose, it interacts with both the CM and the BGP routing daemon. Interaction with the BGP routing daemon is performed using standard protocols, since currently available API does not provide special hooks for BGP events. In our architecture, CRCM maintains an iBGP session with the BGP routing daemon in which the daemon is configured as route reflector [BCC06]. Over that iBGP session, the CRCM receives all the BGP best routes chosen by the router. After the reception of a BGP update, the CRCM modifies the set of counting rules accordingly, and forwards information about the changes that has to be performed to the CM.

A counters access interface is also added to standard router modules. It
enables external systems to easily access the values of counters. In practice, the counters access interface can be either designed from scratch or based on other standard protocols like SNMP.

**Practical Issues**

A few issues must be handled when implementing the architecture depicted in Fig. 6.1.

Firstly, the CM might take some time to activate a new counting rule or to modify an existing one, since compilation and optimization steps may be required. Moreover, routing events, like routing instabilities or BGP path explorations, can generate many short-lived routing updates that can force the system to update rules at an unnecessarily high rate. Depending on implementative aspects, the continuous modification of single counting rules can be time consuming and can introduce errors in the TM computation. For these reasons, the CRCM can gather BGP updates and require the CM to update the counting rules at tunable time intervals, e.g., periodically. This allows the CM to use a single transaction for updating many rules and helps limiting the impact of transient routing events. Time granularity at which counting rules are updated on the device can be conveniently tuned according to specific needs and device performance. Clearly, postponing rule updates can, in turn, introduce errors, since the CM is not always aligned with the routing control plane. To minimize such errors, the CRCM can assign a priority value to each counting rule according to the amount of traffic handled by that rule, making the system more responsive to updates that are more likely to introduce a larger error, if postponed. Effectiveness of the priority mechanism is increased by the fact that traffic distribution across the Internet is becoming more and more skewed \[LIJM+10\]. To prioritize rule updates, the CRCM may also take into account the values of counters stored by the CM (dotted line in Fig. 6.1).

Also, the CRCM can be implemented either totally inside routers or, at least partially, outside them. Both options allow a distributed solution to be deployed, mandating each router to autonomously compute a part of the TM, thus significantly reducing the total overhead \[PTL04\]. On one hand, the first option minimizes the communication overhead while forwarding capabilities of routers should not be affected, since the CRCM only needs to use the control plane CPU and memory (see Section 6.4). On the other hand, the latter option would make the architecture suitable for more flexible measurement operations. For instance, moving the CRCM outside routers allows the rule computation logic to be centralized and easily modified, e.g., providing support for different
kinds of what-if analyses. For example, an operator could compute the traffic matrix resulting from applying a different BGP configuration to routers, using a fictitious BGP RIB to feed the external CRCM. Such a flexibility comes at the cost of re-introducing some communication overhead.

Key Benefits and Major Limitations

In order to efficiently manage huge volumes of traffic, the counting mechanisms already available on routers are typically implemented in a highly optimized way (e.g., using ASICs) [SDEK]. One of the major benefits of our architecture is that it exploits these optimized mechanisms, making packet sampling unnecessary. It consumes a limited amount of memory, since no data on IP traffic flows is stored but only a counter per BGP next-hop and a set of counting rules. Also, the way counting rules are actually stored on devices depends on the implementation of the CM, that, however, must be designed to be scalable since packet forwarding is concerned.

Usage of highly optimized mechanisms can allow our architecture to accurately measure TMs in near real-time. Indeed, results of our preliminary experiments (see Section 6.5) are quite promising. A complete performance study, however, is subject for future work. Our evaluation is based on a prototype deployed on a Juniper device. This also shows practical feasibility of our architecture. Observe that such an architecture can, in principle, be deployed at any router that allows to configure the CM using programmability features. In this sense, our approach is vendor independent.

Nevertheless, our architecture relies on few reasonable assumptions concerning the routing control plane. First of all, routing information is assumed to be correct, i.e., network configuration is assumed not to be subject to routing or forwarding anomalies [GW02b]. If such anomalies can arise in the network, the computation of the TM cannot rely only on local BGP routing information owned by single routers. We also assume that all destination network prefixes are advertised in BGP. Finally, the association between each BGP next-hop and the corresponding PoP is assumed to be known or to be easily derived from known topological information.

6.4 Implementation

We developed a prototypical implementation of the proposed architecture. A high-level diagram of our prototype is depicted in Fig. 6.2. In order to facilitate the installation of counting rules and the access to the values of counters,
we deployed a router application inside a programmable router. The router application communicates with an external system that implements the core functionalities of the CRCM. The choice of implementing the CRCM outside the router is motivated by the possibility of rapidly prototyping the module and easily experimenting different strategies. Of course, deploying the CRCM inside the router is also possible and it is part of our future work. All the operations performed by the external system are CPU-bound and communications between submodules happen over pipes. Thus, implementing the whole architecture inside the router should only affect the control plane CPU and memory usage. Being forwarding and control plane typically separated in commercial devices, we expect that device capability to route packets should not be concerned by our application.

To realize the router application, we used the Junos SDK, which allows third party developers to install custom applications inside Juniper routers [KAB09]. To define and modify counting rules, our prototype exploits the firewall filters manipulation facilities provided by the Junos SDK API.

The counting rules installer component allows the external system to specify simple commands for adding and removing counting rules, using an ad-hoc protocol based on TCP. Right after receiving a command, the counting rules installer uses the Junos SDK API to instruct Junos to perform the operation specified by the command. Access to counters is provided by the counter reader component, which can print current values of counters on a plain text file.
The external system realizing the core of CRCM consists of two submodules: a BGP receiver and a commands generator.

The **BGP receiver** is realized as a customly modified version of the Python Routeing Toolkit (PyRT) open-source software [Mor]. It maintains a BGP session with a given BGP peer and dumps the association between each IP prefix and the corresponding BGP **next-hop** as communicated by its peer. The BGP receiver separately dumps data concerning the BGP RIB and successive BGP updates.

The **commands generator** processes data dumped by the BGP receiver in order to generate the commands to be sent to the router application. We use a single firewall filter containing several terms to implement the counting rules.

To avoid redundancy and lower the number of terms as much as possible, the commands generator reduces the size of the BGP RIB using a Patricia Trie data structure and an algorithm similar to that described in [DKVZ99]. Among optimizations performed, only one counting rule, matching the 0/0 IP prefix, is added for the BGP **next-hop** associated to the highest number of IP prefixes. Since that rule is installed on the router as the last-evaluated counting rule, it matches only the traffic that is not destined to any other BGP **next-hop**. Notice that correctness of the traffic count is not affected by the usage of the 0/0 IP prefix since we apply counting rules on output traffic, i.e., traffic that is not routed by the device does not increment any counter.

The Patricia Trie data structure is also exploited to elaborate successive BGP updates, avoiding the generation of unnecessary commands. For instance, a command is not generated for a BGP update that specifies an association between an IP prefix and a BGP **next-hop** if the association between a less specific IP prefix and the same BGP **next-hop** is already known. However, from an algorithmic point of view, no guarantee is provided on the optimality of the data structure (and of the resulting counting rules) after arbitrary sequences of BGP updates. To avoid progressive degradation of the data structure, the counting rules computation algorithm can be periodically re-run (i.e., once per day).

Generated commands are finally collected for a tunable time interval, in order to send them to the router application in groups and to eliminate redundant ones (e.g., sequences of add and delete operations for the same term). We implemented the commands generator module using few Perl and Bash scripts that run in pipeline.
6.5 Evaluation

In this section, we evaluate the viability of our approach, discussing the results of experiments we run. In particular, we performed a preliminary study on the load our solution imposes on routers, and we estimated the accuracy our prototype is able to achieve. We repeated each experiment several times in order for the results to be statistically significant. In the following, we present average data.

The topology of the testbed we used in all the experiments is depicted in Fig. 6.3. We deployed the router application presented in Section 6.4 on an M7i Juniper router. The device is equipped with an old generation of Forwarding Engine Board (FEB) which is not designed to support a high rate of rule updates. In the following, we refer to the M7i router as the device-under-test (DUT). We connected the DUT to a traffic generator, a Smartbits 600B. The traffic generator generated and collected IP traffic traversing the DUT, which was configured to route all packets back to the traffic generator. Since the DUT is equipped with only two FastEthernet interfaces, we were not able to measure performance (e.g., switching throughput) degradation when the device is stressed with huge amounts of traffic. However, since the CM is implemented in hardware in the DUT and the majority of the operations performed by the router application are CPU-bound, we expect the throughput of the DUT to be marginally affected by our solution. Our prototypical CRCM was installed on commodity hardware (a desktop PC equipped with a dual-core 2.4 GHz CPU and 4G of RAM). It was configured to receive the full BGP RIB and a stream of real-world BGP updates from the border router of a medium-sized
6.5. EVALUATION

Italian ISP and to send the corresponding commands to the router application inside the DUT.

In the first experiment, the CRCM generated the commands associated to the full BGP RIB (consisting of more than 310,000 IP prefixes) of the medium-sized ISP and the corresponding counting rules were installed inside the DUT, sending all the commands in a single group. Using our optimizations, we needed only 408 counting rules, each matching 2000 IP prefixes at most, to represent the entire BGP RIB. The total number of counters is 370. The generation of all the commands took about 8 seconds and the installation of the whole set of counting rules on the device took less than 40 seconds, during which the control plane CPU usage at the DUT was about 90%. Observe that, since the BGP RIB must be processed only once (for each reboot), this computation is independent from the number of peerings the DUT has, and depends on the RIB entries only.

We then checked that counters were correctly installed and properly worked, by configuring the traffic generator to send traffic (at top speed for a FastEthernet interface) destined to several IP prefixes and comparing collected data with the values of the counters on the DUT. In all the tests we run, we found that every packet sent by the traffic generator incremented the right counter, i.e., the counter associated to the BGP next-hop matching the IP destination address of the packet (as specified in the RIB).

As a second experiment, we configured the CRCM to run for 5 consecutive minutes, sending commands to the DUT every 10 seconds. Such commands corresponded to the processing of both the initial RIB and the stream of BGP updates received from the ISP’s border router. The CRCM always spent few millisecond to process BGP updates. The DUT control plane used no more than 3% additional memory and the control plane CPU usage was always less than 45%, with spikes of 35% - 40% typically for about 7 seconds.

Finally, we measured the responsiveness of our prototype to rule updates, i.e., the time took by our system to update a single counting rule. We configured the traffic generator to generate only traffic destined to specific IP prefixes. While traffic traversed the DUT, at a certain time we sent a group of 17 ad-hoc commands, which represents one of the bigger groups practically generated by the CRCM in the previous experiment. Each command in the group modified a term containing several IP prefixes. Among those modifications, commands replaced the counters associated to the IP prefixes traffic is destined to, simulating changes in the BGP next-hop associated to such prefixes. By comparing actual values of the counters with their expected values, we found that the DUT took about 8 seconds for replacing counters. Indeed, old counters, i.e., counters
CHAPTER 6. LEVERAGING ROUTER PROGRAMMABILITY FOR TRAFFIC MATRIX COMPUTATION

associated to monitored prefixes before changes required by ad-hoc commands, are incremented for about 5 seconds more than expected, while a number of packets corresponding to about 3 seconds is not accounted by any counter. In our tests, the time taken by the DUT for replacing counters was independent both on the number of commands in the group and on the amount of traffic the DUT was loaded with.

We finally estimated the accuracy our prototype can achieve, computing the maximum error made on single counters 
\[ E_{max} = \frac{t_{inst} + t_{\text{group}}}{t_{\text{inter}}} \]
where \( t_{\text{inst}} \) is the time took for installing rules at the DUT, \( t_{\text{group}} \) is the time interval configured at the CRCM, and \( t_{\text{inter}} \) is the inter-arrival time between consecutive BGP updates that changed the BGP next-hop associated to the counter. Exploiting the BGP peering with the medium-sized ISP, we collected data in different days between July 20th, 2009 and August 11th, 2009. We found that for the 98% of IP prefixes, the 90th percentile of BGP updates that contain a change of the BGP next-hop has an inter-arrival time always higher than 15 minutes, resulting in a maximum error of 2% if commands are grouped for 10 seconds. Such a maximum error falls to 0.89% if commands are sent to the DUT as soon as they are generated. Since it is shown [RWXZ02] that routing changes rarely affect large amounts of traffic, we expect average errors on TM to be very limited in practice.

We consider experimental results on our prototype promising. Also, we expect the responsiveness of our solution to be improved by usage of a new generation FEB and optimization of our prototypical software.

6.6 Conclusions

We believe that router programmability has the potential to provide fundamental support for network management in the near future, paving the way for a new generation of innovative monitoring solutions.

In this chapter, we showed the feasibility of a distributed architecture for accurate computation of traffic matrices. The architecture is based on programmable routers which autonomously compute different parts of the matrix. We realized a prototypical implementation of our architecture using current technologies. Preliminary experiments we performed are promising.

We plan to improve the algorithms we adopted in our prototype and deeper evaluate our solution through performance tests on up-to-date hardware. We also plan to investigate opportunities opened by router programmability to solve other monitoring and management problems as future work.
Part IV

Evolving Routing
Chapter 7

Enabling Network-Wide IGP Reconfigurations *

7.1 Introduction

Network-wide reconfigurations of routing within a running network, such as routing protocol replacement or the modification of its configuration, can improve performance, scalability, manageability, and security of the entire network. Also, as the network grows or when new services have to be deployed, network operators often need to perform large-scale reconfigurations [Hv10].

However, reconfigurations are an important source of concern for network operators as they can lead to long, service-disrupting outages. Reconfiguring a routing protocol is a complex process since all the routers have to be reconfigured in a proper manner. Restarting the network with the new configurations do not work since most of the networks carry traffic 24/7. Therefore, reconfigurations have to be performed gradually, while the network is running. Such operations can lead to significant traffic losses if they are not handled with care. Unfortunately, network operators typically lack appropriate tools and techniques to seamlessly perform large, highly distributed changes to the configuration of their networks. They also experience difficulties in understanding what is happening during a migration since complex interactions may arise between upgraded and non-upgraded routers. Consequently, as confirmed by

*Part of the material presented in this chapter is based on the following publications L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, O. Bonaventure. Seamless Network-Wide IGP Migrations. In Proc. SIGCOMM, ACM, 2011.
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

many private communications with operators, large-scale configuration changes are often avoided until they are absolutely necessary, thus hampering network evolvability and innovation.

In this chapter, we aim at enabling **seamless IGP migrations**, that is, progressive modifications of the commonly used link-state IGP configuration of a running network without losing packets. Our contribution is manifold. After introducing an abstract model for link-state IGPs in Section 7.2, we discuss the most common migration scenarios ISPs encounter in Section 7.3. In the same section, we also formalize the IGP migration problem we tackle. In particular, we show that long-lasting forwarding loops can appear, both theoretically and practically, when changes are made to the IGP hierarchy and when route summarization is introduced or removed.

In Section 7.4, we introduce a methodology that enables seamless IGP migrations while minimizing the number of reconfigurations per router. Our methodology leverages the fact that several IGP processes can run at the same time on a router. Each IGP process is assigned with a **administrative distance** (AD), and the IGP process with the lowest distance controls the forwarding. The reconfiguration consists in introducing the final IGP configuration at each router with a high AD, wait for its convergence, and lower the distance router by router so that it is used for forwarding. The crucial point of our methodology consists in identifying the order in which to reconfigure the routers while guaranteeing no packet loss. In particular, we focus on avoiding forwarding loops throughout the reconfiguration process.

In Section 7.5, we show that finding such an ordering is an $NP$-complete problem. Also, we propose an exponential algorithm and a heuristic which can be used to find an lossless migration ordering. The exponential algorithm always find the ordering when it exists, but completeness comes at the cost of high time complexity. On the contrary, the heuristic trades completeness for time efficiency. Indeed, it is correct (i.e., never outputs an ordering which is not lossless), but it is not always able to find a lossless ordering even if it exists.

Based on these algorithms, we built a prototype provisioning system which we describe in Section 7.6. The provisioning system automates the whole migration process according to our methodology: it generates router configurations, assesses the proper state of the network and updates all the routers in an appropriate sequence, while monitoring the entire reconfiguration process. As shown in our evaluation and case study, such a provisioning system enables faster and seamless IGP migrations, while avoiding human errors due to manual design and application of new configurations on routers.

In Section 7.7, we evaluate algorithms and our system implementation on
both inferred and real-world topologies. Our results suggest that, in real-world networks, it is possible to find an ordering in which to reconfigure the routers while guaranteeing no forwarding loop. We also show effectiveness of our provisioning system by simulating a full reconfiguration of the GEANT network \cite{gea10} in a virtual environment.

In Section 7.8, we explain how to deal with network failures that can arise during the reconfiguration. Indeed, we extend our algorithms to find a migration ordering which provides additional guarantees in case of network failures.

In Section 7.9 we present design guidelines that facilitate IGP migrations. We review related work in Section 7.10, and we discuss limitations of our techniques and future directions in Section 7.11. Finally, we conclude in Section 7.12.

7.2 An Abstract Model for Link State IGPs

In this section, we aim at capturing IGP configurations and forwarding behavior of routers in a model that abstracts protocol-specific details. Transient IGP behavior is not modeled since we ensure that both the initial and the final IGPs have converged before starting the migration process (see Section 7.4).

We formally define an IGP configuration as a tuple \((p, G, D, w, m)\). The tuple reflects configuration knobs available to operators. In such a tuple, \(p\) is the identifier of an IGP protocol, e.g., OSPF or IS-IS, and \(m\) is the mode in which the protocol is configured, namely flat or hierarchical. \(G = (V, E)\) is the logical graph, i.e., a directed graph that represents the IGP adjacencies among routers participating in \(p\). Each node in \(V\) represents an IGP router, and each edge in \(E\) represents an adjacency between the two routers. Edges are labeled with the name of the zones to which they belong. Moreover, the function \(w : E \rightarrow \mathbb{N}\) associates a positive integer, called weight, to each edge in \(G\). Finally, \(D \subseteq V\) is the set of IGP destinations for traffic that flows in the network. We associate each destination to a single node in \(G\), assuming that each IP prefix is announced by one router only. This assumption is without loss of generality, as we can use virtual nodes and \(G\) can be transformed in a multi-graph, in order to model peculiarities of the considered IGP. For example, consider how to model the binding of each interface to a given area or the redistribution of external prefixes in OSPF. For each router \(r\) that coincides with a traffic destination, we can add a virtual node \(r_j\) for each OSPF area \(j\) in which \(r\) participates, and a virtual node \(r_{ext}\) for external destinations injected by \(r\) in the IGP. For each \(r_j\), only one edge \((r_{ext} r_j)\), belonging to zone \(Z_j\) and
weighted 1, is added to the graph. One edge $e_j$ for each OSPF area $j$ is also added between $r$ and $r_{ext}$. Each $e_j$ is such that it is labeled as belonging to zone $Z_j$ and $w(e_j) = 1$. The destination set $D$ will contain virtual nodes only. Similarly, virtual nodes can be used to model IP prefixes announced by more than one IGP router.

Packets destined to one router $d \in D$ follow forwarding paths. A forwarding path, or simply path, $P$ from $s$ to $d$ is a path $P = (s \ r_1 \ldots \ r_k \ d)$ on $G$ in which $r_i$, with $i = 1, \ldots, k$, are routers traversed by the traffic flow. Several forwarding paths can simultaneously be used for the same pair $(s, d)$, e.g., in case of Equal Cost Multi-Path (ECMP). The weight of a path is the sum of the weights of all the links in the path.

According to the IGP configuration, each router chooses its preferred path towards each destination and forwards packets to the next-hops in such preferred paths. To model this behavior, we define the next-hop function $nh$, and the actual path function $\pi(u, d, t)$. Both functions are derived from the IGP configuration of a network, i.e., an IGP configuration univocally determines the next-hop and the actual path functions. We denote the set of successors (next-hops) of $u$ in the paths router $u$ uses at time $t$ to send traffic destined to destination $d$ by $nh(u, d, t)$. Notice that $|nh(u, d, t)|$ is not guaranteed to be equal to 1, since routers can use multiple paths to reach the same destination (e.g., in presence of ECMP). The paths actually followed by packets sent by $u$ towards $d$ at time $t$ can be computed as a function $\pi$: $\pi(u, d, t)$ is the set of paths resulting from a recursive concatenation of next-hops. More formally, $\pi(u, d, t)$ is a function that associates to each router $u$ the set of paths $\{(v_0 \ v_1 \ldots \ v_k)\}$, such that $v_0 = u$, $v_k = d$ and $\forall i \in \{0, \ldots, k - 1\}$ $v_{i+1} \in nh(v_i, d, t)$. Note that the actual path function does not always coincide with the preferred path of each router, since deflections can happen in the middle of a path [IFU05]. A series of deflections can even build a forwarding loop, as shown in different examples described in Section 7.3. More formally, there exists a forwarding loop, or simply a loop, for a given destination $d$ at a given time $t$ if $\exists r$: $\pi(r, d, t) = (r \ v_0 \ldots \ v_j \ r)$, with $j \geq 0$.

By appropriately tuning the next-hop function, our model is able to capture specific details of IGP configurations such as the corresponding forwarding rules in hierarchical and flat mode, and route summarization. In Section 7.3, we provide some examples of next-hop functions, actual path functions, and migration loops in different migration scenarios.
7.3 The IGP Migration Problem

In this section, we study the problem of seamlessly migrating a network from one IGP configuration to another. Both configurations are provided as input (i.e., by network operators) and, by definition, are loop-free.

**Problem 7.1** Given a unicast IP network, how can we replace an initial IGP configuration with a final IGP configuration quickly, with minimal configuration changes and without causing any forwarding loop?

Assuming no network failure and no congestion, solving this problem leads to seamless migrations. Observe that our approach reduces the opportunities for failures during the migration process, because of its time efficiency. Further, in Section 7.8, we show how to extend our techniques to provide guarantees even in case of network failures. Similar extensions may be used to avoid congestion during the migration. We plan to fully investigate congestion-free migration techniques in future work. However, we argue that congestion issues are less critical, as they can be strongly mitigated by performing the migration during time slots in which traffic is low. Also, large ISPs are normally overprovisioned [ICBD04], further reducing the risk of congestion.

In this chapter, we focus on issues generated by the IGPs themselves, while leaving migration issues due to the presence of additional routing protocols in the network (e.g., BGP) to future work. In the rest of the chapter, we call router migration the replacement of the initial next-hop function $nh_{init}$ with the final next-hop function $nh_{final}$ on a given router. Formally, we define the operation of migrating a router $r$ at a certain time $\bar{t}$ as the act of configuring the router such that $nh(r, d, t) = nh_{final}(r, d), \forall d \in D$ and $\forall t > \bar{t}$. We call router migration ordering the ordering in which routers are migrated. A network migration is completed when all routers have been migrated. In this work, we focus on per-router migrations in which all the destinations are migrated at the same time in order to limit the number of configuration changes and to minimize the migration duration. However, such an ordering might not always exist as described in Section 7.3. Only in such cases, we compute and apply separate migration orderings for the troublesome destinations (see Section 7.4). Results of our evaluation (see Section 7.7) suggest that troublesome destinations are zero or few in realistic topologies.

Throughout the chapter, we focus on migration loops, that is, loops arising during an IGP migration because of a non-safe router migration ordering. Migration loops are not protocol-dependent, and are more harmful than loops that arise during protocol convergence as they last until specific routers are...
migrated (e.g., see Section 7.3). Observe that, if \( nh_{init} = nh_{final} \), the \( \pi \) function does not change either, hence any router migration ordering is ensured to be loop-free.

**IGP migration scenarios**

Most of the time, network operators target three aspects of the IGP when they perform large-scale migrations. First, they may want to replace the current protocol with another. For example, operators may want to migrate to an IGP that provides more guarantees against security attacks [GM03, nan08], or that allows to integrate new equipments which are not compliant with the adopted one [nan05], or that is not dependent on the address family (e.g., OSPFv3, IS-IS), e.g., to run only one IGP to route both IPv4 and IPv6 traffic [gea03, nan08]. Second, when the number of routers exceeds a certain critical mass, operators often introduce a hierarchy within their IGP to limit the control-plane stress [DRM08, Tho03]. Another reason operators introduce hierarchy is to have more control on route propagation by tuning the way routes are propagated from one portion of the hierarchy to another [Hv10]. On the contrary, removing a hierarchy might be needed to better support some traffic engineering extensions [RVB05]. Third, network operators also modify the way the IGP learns or announces the prefixes by introducing or removing route summarization. Route summarization is an efficient way to reduce the number of entries in the routing tables of the routers as IGP networks can currently track as many as 10,000 prefixes [LDF+11]. Route summarization also helps improving the stability by limiting the visibility of local events. Actually, some IGP migrations combine several of these scenarios, such as the migration from a hierarchical OSPF to a flat IS-IS [GM03]. Finally, operators may be forced to revert back to a previous IGP configuration to meet technical requirements [Tem09].
7.3. THE IGP MIGRATION PROBLEM

In this chapter, we address all those three kinds of IGP reconfiguration scenarios, which are summarized in Table 7.1. Observe that each scenario concerns the modification of a specific feature of the IGP configuration. Moreover, different scenarios can be combined if more than one feature of the IGP configuration have to be changed. We do not consider the change of link weights as a network-wide migration. Indeed, traffic matrices tend to be almost stable over time [RWXZ02], and ISPs typically change the weights of a few links at a time. Moreover, effective techniques have already been proposed for the graceful change link weights [RZC11, FB07, FSB07, FSK08, SFF09]. Nevertheless, our generalized model and the techniques we present in Section 7.5 are also applicable to reconfigure link weights. Furthermore, since the addition and the removal of links and devices can be modeled as a change of some link weights from an infinite to a finite value or vice versa, our approach can also be used to guarantee no packet loss during topological changes.

We now describe the issues that must be addressed in each migration scenario we target, using the notation introduced in Section 7.2.

**Protocol replacement**

This migration scenario consists of replacing the running IGP protocol, but keeping the same \( nh \) function in the initial and in the final configurations. A classical example of such a scenario is the replacement of an OSPF configuration with the corresponding IS-IS configuration [Hv10]. Since the \( nh \) function is the same in both IGPs, routers can be migrated in any order without creating loops.

**Hierarchy modification**

Three migration scenarios are encompassed by the modification of the IGP hierarchy. First, a flat IGP can be replaced by a hierarchical IGP by introducing several zones. Second, a hierarchical IGP can be migrated into a flat IGP by removing peripheral zones and keeping only one zone. Third, the structure of the zone in a hierarchical IGP can be changed, e.g., making the backbone bigger or smaller. We refer to these scenarios as flat2hier, hier2flat and hier2hier, respectively.

Unlike protocol replacement, changing the mode of the IGP configuration can require a specific router migration ordering. Indeed, the \( nh \) function can change in hierarchy modification scenarios because of the intra-zone over inter-zone path preference rule applied by routers in hierarchical IGPs (see Chapter 1). Hence, forwarding loops can arise due to inconsistencies between already
migrated routers and routers that are not migrated yet. Consider for example the topology depicted on the left side of Fig. 7.1. In a flat2hier scenario, some routers change their next-hop towards destinations E1 and E2. In particular, the right side of Fig. 7.1 shows the next-hop function for all the routers when the destination is E2. During the migration process, a forwarding loop arises for traffic destined to E2 if B1 is migrated before E1. Indeed, B1 reaches E2 via E1 in hierarchical mode, and E1 reaches E2 via B1 in flat mode. Hence, for each time t where B1 is already migrated and E1 is not, the forwarding path used by B1 is $\pi(B1, E2, t) = \{(B1 E1 B1)\}$, since $nh_{final}(B1, E2) = \{E1\}$ and $nh_{init}(E1, E2) = \{B1\}$. Notice that such a loop lasts until E1 is migrated. A symmetric constraint holds between routers B2 and E2 for traffic destined to E1. A loop-free migration can be achieved by migrating E1 and E2 before B1 and B2.

Nevertheless, there are also cases in which it is not possible to avoid loops during the migration. Consider, for example, the topology represented in Fig. 7.2. In this topology, symmetric constraints between B1 and B2 for traffic destined to E2 and E3 imply the impossibility of finding a loop-free ordering. We refer the reader to the central and the right parts of Fig. 7.2 to visualize the next-hop functions in flat and hierarchical modes.

Similar examples can be found for hier2flat and hier2hier migrations. They are omitted for brevity. Observe that problems in hierarchy modification scenarios are mitigated in protocols such as IS-IS that natively support multiple adjacencies [Ora90]. In fact, multiple adjacencies belonging to different zones decrease the number of cases in which the $nh$ function changes during the migration. However, migration loops can still arise, depending on the initial and the final configurations.
7.3. THE IGP MIGRATION PROBLEM

Figure 7.2: Loop Gadget. No migration ordering is loop-free for flat2hier and hier2flat scenarios because of contradictory constraints between $B_1$ and $B_2$.

Figure 7.3: Route summarization gadget. When summarization is introduced or removed, a specific migration ordering is needed between $B_3$ and $B_4$ to avoid forwarding loops.

Route summarization

Introducing or removing route summarization (i.e., summarization scenarios) in a network can lead to forwarding loops. For example, consider the topology represented in the left part of Fig. 7.3. The right part of the figure visualizes the $nh$ functions before and after the introduction of route summarization. In this case, the introduction of route summarization on $B_1$ and $B_2$ can lead to a
forwarding loop between $B_3$ and $B_4$ for traffic destined to $E_2$. Indeed, before summarizing routes, $B_3$ and $B_4$ prefer to send traffic destined to $E_2$ via $B_2$. On the other hand, when summarization is introduced, $B_1$ and $B_2$ propagate one aggregate for both $E_1$ and $E_2$ with the same weight. Hence, $B_3$ and $B_4$ change their next-hop since the path to $B_1$ has a lower weight than the path to $B_2$.

As for hierarchy modifications, no loop-free ordering exists in some cases. An example of such a situation can be built by simply replicating the topology in Fig. 7.3 so that symmetric constraints on the migration order hold between $B_3$ and $B_4$.

### 7.4 Methodology

Fig. 7.4 illustrates the main steps of our methodology. In the first step, we pre-compute an ordering in which to seamlessly migrate routers with no packet loss (Section 7.5). Migrating all the routers at once is not a viable solution in practice as it can generate protocol-dependent loops and control-plane traffic storms concerning all the protocols (BGP, LDP, PIM, etc.) that rely on the IGP. Moreover, this approach prevents operators from controlling the migration process and from falling back to a previous working state when a problem is detected, e.g., when a router does not receive an intended command. All the discussions that we had with network operators further confirmed that they prefer to gradually migrate their network to have full-control of the process. In the same step, when a per-router ordering does not exist, we identify the set of problematic destinations for which contradictory ordering constraints exist, and we compute a per-destination ordering for each of them. Then, we compute a per-router ordering for the rest of the destinations.

The actual migration process begins in the second step. As basic operation, we exploit a known migration technique called *ships-in-the-night* [Hv10, GM03, gea03], in which both the initial and the final IGP configurations are running at the same time on each router in separate routing processes. Routing processes are ranked on the basis of their priority, the Administrative Distance (AD). When a route for a given prefix is available in multiple processes, the one with the lowest AD is installed in the Forwarding Information Base (FIB). In this step, we set the AD of the routing process running the final IGP configuration to 255, since this setting ensures that no route coming from that process is installed in the FIB [BFCW09]. All ISP routers typically support this feature.

In the third step of the migration, we wait for network-wide convergence of
7.4. METHODOLOGY

Seamless IGP Migration Methodology

(i) Compute a lossless router migration order. In case no per-router ordering exists, compute a per-destination ordering for the troublesome destinations.

(ii) Introduce the final IGP configuration. The final IGP configuration is introduced on all the routers in the network. However, routers continue to forward packets according to the initial IGP configuration.

(iii) Monitor the final IGP status. Wait for the convergence of the final IGP configuration.

(iv) Progressively migrate routers. The pre-computed lossless router migration order is followed. In case no per-router migration ordering exists, a per-destination ordering is also applied for the troublesome destinations.

(v) Remove the initial IGP configuration. The initial IGP is removed from all the routers in the network.

Figure 7.4: Proposed methodology for seamless IGP migrations.

the final IGP configuration. After this step, both IGPs are in a stable routing state.

In the fourth step, we progressively migrate routers following the ordering pre-computed in the first step of the methodology. For this purpose, we lower the AD of the routing process running the final IGP such that it is smaller than the AD of the process running the initial configuration. Doing so, the router installs the final routes in its FIB. If a per-destination ordering is required for some destinations, we prevent them from being routed according to the final IGP by keeping the AD of these destinations to a high value. This could be done by using tailored route-maps matching the problematic destinations (see [cisc, juna]). After, we migrate the problematic destinations one by one, by lowering their AD following the pre-computed per-destination orderings. Since a routing entry change could take about 200ms before being reflected in the FIB [FFEB05], we wait for a given amount time (typically a few seconds) before migrating the next router in the ordering. This step ensures a loop-free migration of the network. Notice that switching the AD and updating the FIB are lossless operations on ISP routers [FMB+07].

In the last step, we remove, in any order, the initial IGP configuration from the routers. This is safe since all of the routers are now using the final IGP to
forward traffic.

7.5 Loop-Free Migrations

In this section, we study the problem of migrating a network from one link-state IGP configuration to another without creating any loop. Firstly, we prove that the problem is \(NP\)-complete. Then, we present the algorithms we use to compute a loop-free router migration ordering. Finally, we describe how to adapt the algorithms to compute a per-destination ordering to use as fallback when a per-router ordering does not exist.

Router Migration Ordering Problem

We now study the following problem from an algorithmic perspective.

**Problem 7.2** Given an initial and a final next-hop functions, a logical graph \(G\), and a set of destinations \(D\), compute a router migration ordering, if any, such that no forwarding loop arises in \(G\) for any \(d \in D\).

Even the problem of deciding if a loop-free router migration ordering exists, that we call Router Migration Ordering Problem (RMOP), is an \(NP\)-complete problem. In order to prove the complexity of the RMOP problem, we use a reduction from 3-SAT [Pap94]. In the following, we denote the fact that \(u\) is migrated before \(v\) with \(u \prec v\). Consider a logical formula \(F\) in conjunctive normal form. Let \(C_1, \ldots, C_l\) be the clauses in \(F\), \(X_1, \ldots, X_h\) be the variables, and \(X_1\) and \(\overline{X}_1\) the literals corresponding to \(X_1\). In the following, we build the RMOP instance \(S = (G = (V, E), D, nh_{init}, nh_{final})\) corresponding to \(F\).

As a basis, \(G\) contains a single vertex \(P\). For each variable \(X_i\) in \(F\), we add to \(S\) a variable gadget as depicted in Fig. 7.5a. In practice, we add two vertices \(d_{i1}, d_{i2}, x_i\) and \(\overline{x}_i\) to \(G\), along with edges \((x_i, P)\), \((\overline{x}_i, P)\), and \((x_i, \overline{x}_i)\). \(d_{i1}\) and \(d_{i2}\) are also added to \(D\). Intuitively, node \(x_i\) and \(\overline{x}_i\) represent literals \(X_i\) and \(\overline{X}_i\), respectively. In the following, we call nodes \(x_i\) and \(\overline{x}_i\) literal vertices. Assigning TRUE to \(X_i\) corresponds to migrate \(x_i\) before \(P\), while assigning FALSE to \(X_i\) implies \(\overline{x}_i < P\). For each clause \(C_j = (L_1 \lor L_2 \lor L_3)\), we add a clause gadget similar to that depicted in Fig 7.5b. For each literal in \(C_j\), we add the corresponding literal vertex, along with edges \((l_1 l_2), (l_2 l_3), (l_3 P),\) and \((P l_1)\). Moreover, a vertex \(d_j\) is added to both \(V\) and \(D\). After having added all the vertices, one edge is added to \(E\) from any vertex to any destination.

Finally, we define the next-hop functions. For each \(d_{i1}\), \(nh_{init}(u, d_{i1}) = nh_{final}(u, d_{i1}) = \{d_{i1}\} \forall u \in V\), except \(nh_{init}(x_i, d_{i1}) = \{\overline{x}_i\}, nh_{init}(\overline{x}_i, d_{i1}) = \{x_i\} \forall u \in V\), and \(nh_{final}(x_i, d_{i1}) = \{\overline{x}_i\}, nh_{final}(\overline{x}_i, d_{i1}) = \{x_i\} \forall u \in V\).
7.5. LOOP-FREE MIGRATIONS

Figure 7.5: Gadgets used in the reduction from 3-SAT to RMOP. Solid lines represent \( nh_{init} \), while dotted lines \( nh_{final} \). Edges are labeled with destinations they refer to.

\[
\begin{align*}
\{P\}, \text{ and } nh_{final}(x_i, d_{i1}) &= \{\bar{x}_i\}. \text{ Similarly, for each } d_{i2}, nh_{init}(u, d_{i2}) &= nh_{final}(u, d_{i2}) = \{d_{i2}\} \forall u \in V, \text{ except } nh_{init}(P, d_{i2}) = \{\bar{x}_i\}, nh_{final}(x_i, d_{i2}) &= \{\bar{x}_i\}, \text{ and } nh_{final}(\bar{x}_i, d_{i2}) = \{P\}. \text{ Finally, for each } \tilde{d}_j \text{ corresponding to a clause } C_j = (L_{j1} \lor L_{j2} \lor L_{j3}), nh_{init}(u, \tilde{d}_j) &= nh_{final}(u, \tilde{d}_j) = \{\tilde{d}_j\} \forall u \in V, \text{ except } nh_{final}(P, \tilde{d}_j) = \{l_{j1}\}, nh_{init}(l_{j1}, \tilde{d}_j) = \{l_{j2}\}, nh_{init}(l_{j2}, \tilde{d}_j) = \{l_{j3}\}, \text{ and } nh_{init}(l_{j3}, \tilde{d}_j) = \{P\}. \\
\text{Regarding destinations } d_{i1} \text{ and } d_{i2}, \text{ it is easy to verify that only } x_i, \bar{x}_i, \text{ and } P \text{ can be part of a loop, since the next-hop of all the other vertices is the destination in both the initial and the final next-hop functions. In particular, a loop arises toward } d_{i1} \text{ or } d_{i2} \text{ if and only if } P \text{ is the first node to be migrated or } P \text{ is the very last node to be migrated, respectively.}
\]

**Property 7.1** A router migration ordering does not create a loop towards destinations \( d_{i1} \) and \( d_{i2} \) if and only if

- \( x_i < P \rightarrow P < \bar{x}_i \); or
- \( \bar{x}_i < P \rightarrow P < x_i \)

As a consequence, only the orders \( x_i < P < \bar{x}_i \) and \( \bar{x}_i < P < x_i \) are loop-free. This prevents a variable to be TRUE and FALSE at the same time.

Analogously, for destinations \( \tilde{d}_j \), all the routers, except \( l_{j1}, l_{j2}, l_{j3}, \) and \( P \), cannot be part of a loop, since their next-hop is \( d_j \) in both the next-hop functions. The following property holds for \( l_{j1}, l_{j2}, l_{j3}, \) and \( P \).
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

Property 7.2 A loop arises toward a destination $d_j$ if and only if $P$ is migrated before all the vertices $l_i$, with $i = 1, 2, 3$, corresponding to a literal in $C_j$.

It is easy to check that the reduction can be done in polynomial time. We now use such a reduction to prove the complexity of RMOP.

Theorem 7.1 The Router Migration Ordering Problem is $NP$-complete.

Proof: Consider a logical formula $F$ in conjunctive normal form. Let $S$ be the instance of the Router Migration Ordering Problem corresponding to $F$. Then,

- if $F$ is satisfiable, then there exists a router migration order in $S$ that does not create any forwarding loop. In fact, if $F$ is satisfiable, then there exists at least one boolean assignment such that for each clause $C_j$ at least one literal $L_i$ is TRUE. This corresponds to $l_i < P$ in the migration order. Such a condition guarantees that no loop arises in the clause gadget corresponding to $C_j$, by Property 7.2. The same argument can be iterated on all the clauses in $F$. Since the boolean assignment that satisfies $F$ is a valid assignment, no variable is assigned both TRUE and FALSE at the same time, hence no loop can be generated in the variable gadget (Property 7.1).

- if $F$ is not satisfiable, then there does not exist a router migration order in $S$ that does not create any forwarding loop. In fact, if $F$ is not satisfiable, then for each valid boolean assignment at least one clause $C_n = (L_{j1} \lor L_{j2} \lor L_{j3})$ is not satisfied by the boolean assignment. However, this means that all the literals in $C_n$ are FALSE. This corresponds to migrate $P$ before all the nodes $l_{ji}$, with $i \in \{1, 2, 3\}$. Hence, a loop arises for destination $d_n$ by Property 7.2. The same argument can be iterated on all the boolean assignment on the variables in $F$. As a consequence, every router migration ordering on $S$ contains at least one loop.

The proof is completed by noting that a loop-free router migration order is a succinct certificate for $S$. □

Router Migration Ordering Algorithms

We now present a correct and complete algorithm to find a loop-free ordering. Because of the complexity of the problem, the algorithm is inefficient and can
7.5. LOOP-FREE MIGRATIONS

take several hours to run on very large ISP networks (see Section 7.7). Hence, we also propose an efficient heuristic that is correct but not complete.

We now describe our algorithms in absence of virtual nodes. However, the algorithms can be simply extended to deal with virtual nodes, by enforcing that the identified loop-free ordering contains no conflict between virtual nodes and nodes that represent physical routers. In particular, it can be always guaranteed that any ordering found by our algorithms does not contain virtual nodes, which indeed are not subject to reconfiguration as they do not represent physical routers. This is possible since no virtual node is involved in any ordering constraint generated by our algorithms. In fact, virtual nodes never change their respective next-hop function, and our algorithms do not generate constraints for nodes that keep the same next-hop function as they cannot be responsible for any loop. We now show that the next-hop function at each virtual node is guaranteed to remain the same for all the destinations. Consider the logical graph on which the algorithms run. By construction, each virtual node \( v \) has only one edge, connecting it to the node \( u \) representing the corresponding physical router. Consequently, the next-hop of \( v \) is always \( v \) when \( v \) itself is the destination and it is always \( u \) for any other destination.

**Loop Enumeration Algorithm**

The Loop Enumeration Algorithm (Fig. 7.6) enumerates all the possible migration loops that can arise during a migration. Then, it outputs the sufficient and necessary constraints that ensure that no loop arises. To identify all possible migration loops, for each destination \( d \), the algorithm builds the graph \( G_d \) (line 4) as the union of the actual paths in the initial and in the final configuration. \( G_d \) contains all the possible combinations of paths followed by traffic destined to \( d \) for any migration order. Then, all the cycles are enumerated and for each cycle, the algorithm outputs the constraint (line 8) of migrating at least one router that participates in the loop in the initial configuration before at least one router that is part of the loop in the final configuration (lines 5-8). In the example of Fig. 7.7, indeed, migrating \( c_1 \) before at least one among \( c_2 \) and \( c_3 \) avoids the loop. In the algorithm, \( V_{\text{init},L} \) represents the set of routers that participate in the loop when they are in the initial configuration (line 6), and \( V_{\text{final},L} \) contains only routers that participate in the loop when they are in the final configuration (line 7). The constraints identified by the algorithm are encoded in an Integer Linear Program (lines 12-22), where the variables \( t_u \) represent the migration steps at which routers can be safely migrated (lines 14-19). Finally, the algorithm tries to solve the linear program and returns a
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

160

1: \textit{loop\_enumeration\_run}(G = (V, E), D, nh\textsubscript{init}, nh\textsubscript{final})
2: \textit{CS} \leftarrow \emptyset
3: \textbf{for} \ d \in D \ \textbf{do}
4: \quad \hat{G}_d = (V, \hat{E}), \text{ with } \hat{E} = \{(u, v)\} \text{ such that } v \in nh\textsubscript{init}(u, d) \text{ or } v \in nh\textsubscript{final}(u, d)
5: \quad \textbf{for} \ each \ cycle \ L \ in \hat{G}_d \ \textbf{do}
6: \quad \quad \textit{V\_init},L = \{u \in L : \exists v, (u, v) \in L, v \in nh\textsubscript{init}(u, d) \text{ but } v \notin nh\textsubscript{final}(u, d) \}
7: \quad \quad \textit{V\_final},L = \{u \in L : \exists v, (u, v) \in L, v \in nh\textsubscript{final}(u, d) \text{ but } v \notin nh\textsubscript{init}(u, d) \}
8: \quad \quad \textit{CS} \leftarrow \textit{CS} \cup \{u_0 \lor \cdots \lor u_k < v_0 \lor \cdots \lor v_l\}, \text{ where } u_i \in \textit{V\_init},L \ \forall i = 0, \ldots, k, \text{ and } v_j \in \textit{V\_final},L \ \forall j = 0, \ldots, l.
9: \quad \textbf{end for}
10: \textbf{end for}
11: LP \leftarrow \text{new LP problem}
12: \textbf{for} u_0 \lor \cdots \lor u_k < v_0 \lor \cdots \lor v_l \in \textit{CS} \ \textbf{do}
13: \quad \text{add to } LP \text{ the following constraints}
14: \quad t_{u_0} - \text{MAX\_INT} \times Y_1 < t_{v_0}
15: \quad \ldots
16: \quad t_{u_0} - \text{MAX\_INT} \times Y_l < t_{v_l}
17: \quad t_{u_1} - \text{MAX\_INT} \times Y_{l+1} < t_{v_0}
18: \quad \ldots
19: \quad t_{u_k} - \text{MAX\_INT} \times Y_{l \times k} < t_{v_l}
20: \quad t_{u_0}, \ldots, t_{u_k}, t_{v_0}, \ldots, t_{v_l} \ \text{integer}
21: \quad Y_1, \ldots, Y_{l \times k} \ \text{binary}
22: \quad \sum_{1 \leq i \leq l \times k} Y_i < l \times k
23: \quad \textbf{end for}
24: 
25: \textbf{return} \ \text{solve\_lp\_problem}(LP)

Figure 7.6: Loop Enumeration Algorithm.

loop-free ordering, if one exists (line 24).

We now show correctness and completeness of the Loop Enumeration Algorithm.

**Theorem 7.2** The Loop Enumeration Algorithm is correct and complete.

**Proof:** We prove the statement by showing that the linear program solved in the Loop Enumeration Algorithm encodes all the sufficient and necessary conditions for any migration loop to not arise. Indeed, let \( u_0 \lor \cdots \lor u_k < v_0 \lor \cdots \lor v_l \) be the ordering constraint that the Loop Enumeration Algorithm identifies for a given loop \( L = (c_0 \ c_1 \ldots \ c_k \ c_0) \) concerning traffic destined to
7.5. LOOP-FREE MIGRATIONS

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{migration_loop.png}
\caption{Abstract representation of a migration loop.}
\end{figure}

\textbf{LEGEND}
- \( \text{nh}_{\text{init}} \)
- \( \text{nh}_{\text{final}} \)
- \( c_1 \in V_{\text{init},L} \)
- \( c_2, c_3 \in V_{\text{final},L} \)

\( d \in D \). We now show that \( L \) does not arise at any migration step if and only if the constraint is satisfied.

\textbf{If the loop does not arise then the constraint is satisfied.} Suppose by contradiction that the constraint is not satisfied. Then, there exists a time \( t \) such that all the routers in \( V_{\text{final},L} \) are migrated while all the routers in \( V_{\text{init},L} \) are not migrated. Consider \( c_0 \). If \( c_0 \in V_{\text{final},L} \), then it is already migrated, i.e., \( \text{nh}(c_0, d, \bar{t}) = \text{nh}_{\text{final}}(c_0, d) \), hence \( c_1 \in \text{nh}(c_0, d, \bar{t}) \), by definition of \( V_{\text{final},L} \). If \( c_0 \in V_{\text{init},L} \), then \( \text{nh}(c_0, d, \bar{t}) = \text{nh}_{\text{init}}(c_0, d) \) and \( c_1 \in \text{nh}(c_0, d, \bar{t}) \). Finally, if \( c_0 \not\in V_{\text{init},L} \) and \( c_0 \not\in V_{\text{final},L} \), then \( c_1 \in \text{nh}(c_0, d, t) \) \( \forall t \). In any case, \( c_1 \in \text{nh}(c_0, d, \bar{t}) \). Iterating the same argument for all the routers in \( L \), we conclude that \( c_{i+1} \in \text{nh}(c_i, d, \bar{t}) \), with \( i = 0, \ldots, k \) and \( c_{k+1} = c_0 \). Thus, \( L \) arises at time \( \bar{t} \).

\textbf{If the constraint is satisfied then the loop does not arise.} Assume, without loss of generality, that \( c_u \in V_{\text{init},L} \) is migrated at time \( t' \), while at least one router \( c_v \in V_{\text{final},L} \) is migrated at \( t'' > t' \). Then, \( L \) cannot arise \( \forall t < t'' \), since \( \text{nh}(c_v, d, t) = \text{nh}_{\text{init}}(c_v, d) \) implies that \( c_{v+1} \not\in \text{nh}(c_v, d, t) \) by definition of \( V_{\text{final},L} \). Moreover, \( L \) cannot arise \( \forall t > t' \), since \( \text{nh}(c_u, d, t) = \text{nh}_{\text{final}}(c_u, d) \) implies that \( c_{u+1} \not\in \text{nh}(c_u, d, t) \) by definition of \( V_{\text{init},L} \). Since \( t'' > t' \), no time exists such that \( L \) arises during the migration. \( \square \)

It is easy to verify that the algorithm requires exponential time. Indeed, the algorithm is based on the enumeration of all the cycles in a graph, and the number of cycles in a graph can be exponential with respect to the number of nodes.
Routing Trees Heuristic

The Routing Tree Heuristic is illustrated in Fig. 7.8. Intuitively, it computes ordering constraints separately for each destination, so that next-hop changing routers are not migrated before their final forwarding path to each destination is established (similarly to what proposed in [FSK08, FB07]). A router ordering that satisfies all per-destination constraints is then computed. As the first step, for each destination \( d \in D \), the heuristic exploits a greedy procedure to compute a set \( S_d \) of nodes that are guaranteed not to be part of any loop (line 4). The greedy procedure (lines 20-32) incrementally (and greedily) grows the set \( S_d \), adding a node to \( S_d \) at each iteration if and only if all the next-hops of the node in the initial and in the final configurations are already in \( S_d \) (lines 27-28). After this step, the Routing Trees Heuristic builds directed graph \( G_d \), which is guaranteed to be acyclic since the final configuration is loop-free. \( G_d \) contains only the actual paths followed by packets to reach \( d \) in the final configuration (line 6). Then, it generates a constraint for each pair of routers \( (u, v) \) such that \( (u \ldots v \ldots d) \in \pi_{final}(u, d) \), and both \( u \) and \( v \) do not belong to \( S_d \) and change at least one next-hop between the initial and the final configuration (lines 7-15). In particular, among the routers that change one or more next-hops during the migration (set \( \bar{V}_d \) at line 5), each router is forced to migrate after all its successors in the actual path towards \( d \) (line 11). In the final step, the heuristic tries to compute an ordering compliant with the union of the constraints generated for all the destinations (lines 17-18).

It is easy to check that the algorithm is polynomial with respect to the size of the input. We now prove that the algorithm is correct. First, we show that the routers in \( S_d \) can be migrated in any order without creating loops towards \( d \), hence it is possible not to consider them in the generation of the ordering constraints. Then, we prove that the constraints are sufficient to guarantee that the ordering is loop-free.

**Lemma 7.1** If the greedy procedure adds a router \( u \) to \( S_d \), then \( u \) cannot be part of any migration loop towards destination \( d \in D \).

**Proof:** Suppose, by contradiction, that there exists a router \( u \) added to \( S_d \) by the greedy procedure at a given iteration \( i \), such that \( (u \ldots v \ldots d) \in \pi(u, d, t) \), with \( k \geq 0 \), at a given time \( t \) and for a given migration ordering. By definition of the algorithm, one router is added to \( S_d \) if and only if all its next-hops \( w_0, \ldots, w_n \) (in both the initial and final IGP configurations) are already in \( S_d \), since each node in \( \{w_0, \ldots, w_n\} \) is added to \( S_d \) at a given iteration before \( i \). Hence, \( v_k \not\in S_d \) at iteration \( i \), because \( u \) is one of the next-hops of \( v_k \) and it...
7.5. LOOP-FREE MIGRATIONS

Figure 7.8: Routing Trees Heuristic.

1: \textit{routing\_trees\_run}(G = (V, E), D, nh_{init}, nh_{final})
2: \textbf{C} \leftarrow \emptyset
3: \textbf{for} \textbf{d} \in D \textbf{do}
4: \textbf{S}_d \leftarrow \textit{greedy\_run}(V, d, nh_{init}, nh_{final})
5: \bar{V}_d \leftarrow \{v_i : nh_{init}(v_i, d) \neq nh_{final}(v_i, d)\}
6: G_d = (V, E'), with E' = \{(u, v) : v \in nh_{final}(u, d)\}
7: \textbf{for} P = (v_0 \ldots v_k), with v_k = d, (v_i, v_{i+1}) \in E', and predecessors(v_0) = \emptyset \textbf{do}
8: \textbf{last} \leftarrow Null
9: \textbf{for} u \in P \cap \bar{V}_d \textbf{and} u \notin S_d \textbf{do}
10: \textbf{if} last \neq Null \textbf{then}
11: \textbf{C} \leftarrow \textbf{C} \cup \{(u, last)\}
12: \textbf{end if}
13: \textbf{last} \leftarrow u
14: \textbf{end for}
15: \textbf{end for}
16: G_c \leftarrow (V, C)
17: \textbf{return} \textit{topological\_sort}(G_c)
18:
19: \textbf{return} \textit{greedy\_run}(V, d, nh_{init}, nh_{final})
20: \textbf{S}_d \leftarrow \emptyset
21: N \leftarrow \{d\}
22: \textbf{while} N \neq \emptyset \textbf{do}
23: \textbf{S}_d = \textbf{S}_d \cup N
24: N = \emptyset
25: \textbf{for} u \in V, u \notin S_d \textbf{do}
26: \textbf{if} nh_{init}(u, d) \cup nh_{final}(u, d) \subseteq S_d \textbf{then}
27: N = N \cup \{u\}
28: \textbf{end if}
29: \textbf{end for}
30: \textbf{end while}
31: \textbf{return} S_d
is added to $S_d$ at iteration $i$ by hypothesis. Iterating the same argument, all routers $v_h \not\in S_d$ at iteration $i$, $\forall h = 0, \ldots, k$. As a consequence, Greedy$^+$ does not add $u$ to $S_d$ at iteration $i$, which is a contradiction. 

**Theorem 7.3** Let $S = x_1, \ldots, x_n$ be the sequence computed by the Routing Tree Heuristic. If the routers are migrated according to $S$, then no migration loop arises.

**Proof:** Suppose by contradiction that migration is performed according to $S$ but migrating a router $u$ creates a loop for at least one destination $d$. In that case, there exists a set of routers $V = \{v_1, \ldots, v_k\}$, such that $C = (u v_0 \ldots v_k u) \in \pi(u, d, t)$, at a certain time $t$. By Lemma 7.1, all $v_i \not\in S_d$. By definition of the heuristic, all routers $v_i$ are such that $nh(v_i, d, t) = nh_{final}(v_i, d)$, with $i = 0, \ldots, k$, because either they do not change their next-hop between the initial and the final configuration or they precede $u$ in $S$. Hence, at time $t$, both $u$ and all the routers $v_i \in V$ are in the final configuration. This is a contradiction, since we assumed that the final IGP configuration is loop-free. 

Note that the heuristic is not complete; while the constraints it generates are sufficient to guarantee no forwarding loops, they are not necessary. Indeed, for each destination $d$, it imposes specific orderings between all the routers (not belonging to $S_d$) that change one of their next-hops towards $d$, even if it is not needed. For instance, in the scenario of Fig. 7.9, the heuristic mandates $v$ to be migrated before $u$ and $u$ before $z$. However, no loop arises also if $v$ is migrated before $z$ and $z$ before $u$. Generating unnecessary constraints prevents the heuristic from identifying a loop-free migration ordering every time it exists. Nonetheless, in carefully designed networks [FFS$^+$11], such cases are rare. In Section 7.7, we show that the heuristic found an ordering in most of our experiments on realistic topologies.

**Per-Destination Ordering**

If a per-router ordering does not exist or the Routing Tree Heuristic does not find an ordering, a per-destination ordering can be computed for the problematic destinations, that is, destinations for which contradictory constraints (i.e., a cycle in the constraint graph) exist. Per-destination orderings can be computed directly from the per-destination ordering constraints identified by our per-router ordering algorithms. Note that it may not be necessary to compute an ordering for every problematic destination, as excluding a destination from
7.6 The Provisioning System

We implemented a software system which is able to compute and automate all the required steps for a seamless migration. The main architectural components of our system are represented in Fig. 7.10. In the following, we describe how data flows through the system (dashed lines in the figure), by stressing the role of each component.

The main purpose of the monitoring component is to assess properties of intermediate configurations, that is, checking that given routers are correctly migrated and the expected routing state is reached. The monitoring mechanisms also enables failure detection. However, while we discuss how to prevent packet loss due to network failures in Section 7.8, we plan to study effective reactive strategies to unexpected failures in future work. The monitoring component encompasses an IGP LSA Listener and an IGP State Asserter. The IGP LSA Listener collects and parses the IGP Link-State Advertisements (LSAs) exchanged by routers, storing IGP adjacencies, link weights, and announced IP prefixes in a database. We chose to implement the IGP LSA Listener by using packet-cloning features available on routers, as it is shown to be an effective method to collect all control-plane messages with low resource consumption on monitored routers (as described in Chapter 5). The IGP State Asserter...
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

Figure 7.10: The provisioning system automates all the reconfiguration process. It computes the ordering, monitors the network and pushes the intermediate configurations in the appropriate order.

queries the database and assesses properties of the monitored IGPs state. The current implementation of the IGP State Asserter can check an IGP for convergence completion. IGP convergence is deduced by stability, over a given (customly set) time, of the expected IGP adjacencies and of the announced prefixes. Moreover, the IGP State Asserter is able to verify the announcement of a given set of prefixes in an IGP, and the equivalence of two IGPs, i.e., the equivalence of the logical graph, and of the forwarding paths towards a given set of destinations.

The IGP State Asserter is triggered at specific moments in time by the Migration Controller, which is the central component of the system, responsible for tasks’ coordination. Before the actual migration process starts, it delegates the computation of a loop-free router migration ordering to the Ordering Component. This component implements the ordering algorithms described in Section 7.5. Then, the Migration Controller runs the IGP LSA Listener. When needed (see Section 7.4), the Migration Controller asks the IGP State Asserter to assess whether it is possible to safely modify the configuration of the devices in the network without incurring transient states. This boils down to checking the stability of the current IGP. At each step of the migration process the controller also requires the Configuration Manager to properly update the configuration on the routers as described in Section 7.4. Based on a network-wide model, the Configuration Manager generates the necessary commands to be sent to routers for each migration step. The Configuration Manager is based
on an extended version of NCGuard [VPB08].

7.7 Evaluation

In this section, we evaluate the ordering algorithms and the provisioning system. The system is evaluated on the basis of a case study in which a network is migrated from a flat to a hierarchical IGP.

Data Set and Methodology

Our data set contains both publicly available and confidential data relative to commercial ISP topologies. Concerning publicly available topologies, we used the inferred topologies provided by the Rocketfuel project [SMW02]. Rocketfuel topologies represent ISPs of different sizes, the smallest one having 79 nodes and 294 edges while the biggest one contains 315 nodes and 1944 edges. In addition, some network operators provided us with real-world IGP topologies. In this section, we discuss the result of our analyses on all the Rocketfuel data and on the anonymized topologies of three ISPs, namely tier1.A, tier1.B and tier2. tier1.A is the largest Tier1, and its IGP logical graph has more than 1000 nodes and more than 4000 edges. tier1.A currently uses a flat IGP configuration. The other two ISPs are one order of magnitude smaller but use a hierarchical IGP.

On this data set, we performed several experiments. We considered the introduction of summarization, as well as flat2hier and hier2hier scenarios. Since most of the topologies in our data set are flat, we artificially built a hierarchy (i.e., the separation in zones) in order to consider scenarios in which hierarchical configurations are needed. In particular, we grouped routers according to geographical information present in the name of the routers. Doing so, we built two hierarchical topologies out of each flat topology. In the first one, zones are defined per city. In the second one, zones are defined per-continent. In both topologies, we built the backbone by considering routers connected to more than one zone as ZBRs and routers connected only to ZBRs as pure backbone routers. To simulate a hier2hier scenario, we artificially enlarged the backbone by moving to it a fixed number (from 1 up to 32) of links. Such links were randomly chosen among the links between a ZBR and a router that does not participate in the backbone. For the summarization scenario, we aggregated all the destinations inside the same zone into a single prefix. This was done for all the zones but the backbone. Our hierarchy construction methodology and the way prefixes are summarized follow the guidelines proposed in [Yu00]. All
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

Figure 7.11: CDF of the number of loops that can arise on Rocketfuel topologies. In the worst case, up to 80 different forwarding loops can be created during the reconfiguration.

the tests were run on a Sun Fire X2250 (quad-core 3GHz CPUs with 32GB of RAM). We omit the results of some experiments due to space limitations.

Ordering Algorithms

We first evaluate the usefulness and efficiency of the Loop Enumeration Algorithm and Routing Tree Heuristic. Fig. 7.11 shows the cumulative distribution function of the number of loops that can arise in Rocketfuel topologies. Different migration scenarios are considered. Each point in the plot corresponds to a specific topology and a specific scenario. In flat2hier, up to 80 different loops can arise in the worst case and at least 30 loops can arise for 4 topologies out of 11. Other scenarios follow similar trends. Observe that, in the hier2hier scenario (curves “adding $x$ links to the backbone”), the number of possible loops significantly increases with the number of links which change zone. In all the scenarios, almost all the loops involve two routers, with a few exceptions of three routers loops. Also, the vast majority of loops concerns traffic destined
to routers that do not participate in the backbone. These routers are at the border of the network (e.g., BGP border routers or MPLS PEs) and normally attract most of the traffic in ISP networks. Hence, computing an ordering in which they are not involved in loops can be critical. The number of migration loops is topology dependent, hence it can be influenced by our design approach. However, these results clearly show that migrating routers in a random order is not a viable option in arbitrary networks. Additionally, for practical and coordination reasons, it is desirable that migrations of world-wide networks be carried out on a per-zone basis, that is, migrating all the routers in the same zone (e.g., a continent) before routers in other zones. We observe that this is indeed possible since, in both Rocketfuel and real-world topologies, all the loops arise between routers in the same zone or between backbone routers and routers in a peripheral zone. Thus, it is often possible to compute and apply per-zone orderings. These considerations further motivate our effort to find a router migration ordering which is guaranteed to be loop-free. We found slightly different results on the real ISP topologies we analyzed. For the two hierarchical ISPs, none or few migration loops can arise in the considered scenarios. This is mainly due to a sensible design of the hierarchy by the ISPs. We discuss simple design guidelines that ease IGP migrations in Section 7.9. On the other hand, we found that huge number of problems could arise in a migration from a poor design to a neat one. In the hier2flat scenario, more than 2000 loops, involving up to 10 routers, might arise within the tier1.A.

As a second group of experiments, we ran the ordering algorithms on both the real-world and the Rocketfuel topologies. In the following, we present results for the flat2hier scenario but similar results and considerations hold for the other scenarios. Fig. 7.12 shows for each Rocketfuel topology the percentage of routers that need to be migrated in a specific order according to each algorithm (implying that other routers can be migrated in any order). When a point is missing, it means that the corresponding algorithm was not able to find a loop-free ordering for the topology. The enumeration algorithm was always able to find a loop-free ordering in all situations (including the real-world topologies). In the worst case, the computed ordering involves more than 20% of the routers in the network. We believe that finding ordering constraints for such a number of routers is not practical at a glance. This stresses the importance of our algorithms. The Routing Trees Heuristic, instead, found a loop-free ordering on 9 topologies out of 11. In the remaining two cases, the heuristic was not able to find a solution because of contradictory (unnecessary) constraints relative to 4 and 6 destinations, respectively. Because of the limited number of destinations involved in contradictory constraints, we propose
to apply a per-destination ordering in these cases. Fig. 7.12 also highlights the gain of relying on the greedy sub-procedure, as the heuristic could find a solution for only 6 topologies without it.

Finally, we evaluated the time taken by our ordering algorithms. Typically, time efficiency of ordering algorithms is not critical in our approach, since a loop-free router migration ordering can be computed before actually performing the migration. However, it becomes an important factor to support advanced abilities like computing router migration orderings that ensures loop-free migrations even in case of network failures (see Section 7.8). Fig. 7.13 plots the median of the computation time taken by each algorithm over 50 separated runs. Standard deviation is always under 40 for the loop enumeration algorithm, except for the two cases corresponding to topology 1239, in which standard deviation is around 450. Moreover, the standard deviation of the time taken by the Routing Trees Heuristic is always less than 25. Even if correct and complete, the Loop Enumeration Algorithm is inefficient, especially for large topologies. The heuristic is always one order of magnitude faster. In Fig. 7.13, the low absolute value of the time taken by the Loop Enumeration Algorithm
7.7. EVALUATION

Figure 7.13: Time taken to compute an ordering in flat2hier (Rocketfuel topologies). Results for other scenarios are similar.

can be explained by the relatively small size of the Rocketfuel topologies. Nevertheless, for the tier1.A topology, the Loop Enumeration Algorithm took more than 11 hours to complete. To further evaluate the performance degradation of the complete algorithm, we enlarged tier1.B’s and tier2’s topologies. The operation consisted in replicating multiple times the structure of one peripheral zone, and attaching these additional zones to the network in order to reach a size similar to tier1.A. In such experiments, we found that the Loop Enumeration Algorithm took several hours even if routers can be migrated in any order, while the heuristics always took less than 1.5 minutes.

Provisioning System

We evaluated the performance of the main components of our provisioning system by means of a case study. In the case study, we performed a flat2hier migration of Geant, the pan-european research network, that we emulated by using a major router vendor routing operative system image. In particular, we simulated the migration from a flat IS-IS configuration to a hierarchical
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

172

OSPF. Geant’s topology is publicly available [gea10]. It is composed of 36 routers and 53 links. For the case study, we artificially built zones on the basis of the geographical location of the routers and their interconnections [com]. In addition to the backbone (12 routers), we defined three peripheral zones: the south west area (6 routers), the north east area (11 routers) and the south east area (17 routers). We defined the IGP link weights to be inversely proportional to the bandwidth of the links. By executing the Loop Enumeration Algorithm (see Section 7.5), we found that 8 different loops towards 5 different destinations could arise on that topology.

We ran two experiments. In the first experiment we relied on the ordering computed by the Loop Enumeration Algorithm, while in the second experiment we adopted an alphabetical order based on the name of the routers. The second experiment mimics a naive approach in which ordering constraints are not taken into account. In order to minimize the impact of factors beyond our control (e.g., related to the virtual environment), we repeated each experiment 50 times. To measure traffic disruptions due to the migration, we injected data probes (i.e., ICMP echo request) from each router towards the 5 troublesome destinations. Fig. 7.14 reports the median, the 5th and the 95th percentiles of ICMP packets lost that arose after each migration step.

The case study showed the ability of our provisioning system to perform seamless IGP migrations. Following the ordering computed by the Loop Enumeration Algorithm, we were able to achieve no packet loss during the migration (the few losses reported in Fig. 7.14 should be ascribed to the virtual environment). On the other hand, adopting the naive approach of migrating routers in the random order, forwarding loops arose at step 6 and are only solved at step 34. Thus, the network suffered traffic losses during more than 80% of the migration process. Finally, we observe that, even migrations on a per-zone basis require the use of an ordering algorithm because all the ordering constraints are among routers in the same zone.

Our system also enables faster migrations than known migration [GM03, gea03]. The IGP LSA Listener is able to process IGP messages in a few milliseconds. The performance of the module is confirmed by a separate experiment we ran. We forced the Listener to process messages from a pcap file containing 204 LSAs (both OSPF and IS-IS). On 50 runs, the monitor was able to decode and collect each IGP message in about 14 milliseconds on average and 24 milliseconds at most. We evaluated the performance of the IGP State Asserter on the IS-IS and the OSPF DBs generated during the case study. The DBs contained information about 106 directed links and 96 IP prefixes. The IGP State Asserter took about 40 milliseconds to assess equivalence of the logical
7.8. DEALING WITH NETWORK FAILURES

In this section, we show how to extend the algorithms described in Section 7.3 to deal with network failures.

IGP link and node failures modify the IGP topology which in turn could affect both the $nh$ function and the migration ordering to be followed. Consequently, it may be necessary to adapt the migration ordering to be followed when a failure has been detected in order to not incur long-lasting migration failures.
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

Figure 7.15: Link failures can change the reconfiguration ordering to be followed. In a flat2hier scenario on this topology, a forwarding loop can appear between E1 and B1 if B1 is migrated before E1 and link (B1 E3) fails.

loops. Consider, for example, the topology in Fig. 7.15 and assume a flat2hier migration. The figure shows the initial and the final nh functions towards B2, before (left side) and after (right side) the failure of the link between B1 and E3. Before the failure, any reconfiguration ordering is loop-free since \( nh_{init} = nh_{final} \). However, after the failure of the link between B1 and E3, \( nh_{init} \) is no longer equal to \( nh_{final} \), and a migration loop can be created if B1 is migrated before E1. To prevent forwarding loops exclusively due to link failures, additional constraints need to be considered during the computation of the migration ordering. For instance, in the example of Fig 7.15, E1 should be migrated before B1 to guarantee loop prevention even in case of failure of link B1 – E3.

As a paradigmatic example of how to deal with network failures, we focus on single-link failures. Other kinds of failures (e.g., node and shared risk link group failures) can be similarly addressed. Also, note that single-link failures have been shown to account for the majority of the failures typically occurring in a network [MIB+08]. In the following, we refer to a router migration ordering which prevents loop for any single-link failure in the network as a single-failure compliant ordering.

For each IGP topology, we computed the additional set of constraints for a single-link compliant ordering, by iteratively removing single links from the initial topology and running the constraint generation portion of the Loop Enumeration Algorithm or the Routing Trees Heuristic on the topology we obtained. Fig. 7.16 shows the 50, 99 and 100-percentiles of the number of additional forwarding loops that one single-link failure can trigger. Points that do not appear on the figure are meant to lay on the x axis (i.e., no additional loop due to single-link failures). Typically, very few single-link failures are responsible for the vast majority of the additional forwarding loops and ordering.
7.8. DEALING WITH NETWORK FAILURES

<table>
<thead>
<tr>
<th># of additional loops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1221, city</td>
</tr>
<tr>
<td>1239, city</td>
</tr>
<tr>
<td>1239, cont</td>
</tr>
<tr>
<td>1755, city</td>
</tr>
<tr>
<td>1755, cont</td>
</tr>
<tr>
<td>3257, city</td>
</tr>
<tr>
<td>3257, cont</td>
</tr>
<tr>
<td>3967, city</td>
</tr>
<tr>
<td>3967, cont</td>
</tr>
<tr>
<td>6461, city</td>
</tr>
<tr>
<td>6461, cont</td>
</tr>
</tbody>
</table>

Figure 7.16: Average number of additional forwarding loops created by a single-link failure in flat2hier reconfiguration scenarios on Rocketfuel topologies. Missing points are equal to 0.

constraints. Observe that, in some cases (e.g., AS1239), a single link failure is responsible for more than 500 additional loops. For every network of Fig. 7.16, we also tried to find single-failure compliant ordering by running the resolver part of either one of the two algorithms on the union of all the constraints. In 9 out of the 11 studied networks, we were able to find such a reconfiguration ordering. Also, in one of the two remaining topologies (namely, AS1239, city), we computed a migration ordering that prevents loops for 97% of the possible single-link failures. Results on the real ISP topologies are similar. For tier 1.2, we have been able to find a single-failure compliant ordering, while on tier 2.1, we were able to find an ordering preventing loops for any single-link failure but one. Given the size of the problem, we did not run our algorithm on tier 1.1.

Our results suggest that finding a single-failure compliant ordering is typically possible on small and medium-sized topologies. For huge networks, finding an ordering is harder as the probability of generating contradictory constraints is higher given the large number of links. In this case, a per-destination ordering (see Section 7.4) can be pre-computed for a subset of the destinations thanks

Note that the Routing Tree Heuristic is not usable here as it was not able to find an ordering in the simple case, i.e. without single-link failures.
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

7.9 Design Guidelines

In this section, we state simple design guidelines that make the entire IGP migration process easier, since all the router migration orderings are loop-free. In the following, we consider the design of zones in hierarchical IGPs, since the most problematic migrations involve hierarchies.

**Guideline A**  For each zone $Z$, the shortest path from each ZBR to any destination in $Z$ is an intra-zone path.

Guideline A enables easier flat2hier and hier2flat migrations. In fact, the guideline enforces sufficient conditions to guarantee that the $nh$ function does not change for any router and any destination in any zone $Z$, since an intra-zone path is preferred in both flat and hierarchical modes. Since no router in $Z$ changes its path, then $nh_{\text{init}}(v,d) = nh_{\text{final}}(v,d)$ also for all routers $v \notin Z$ and $d \in Z$. This implies that no loop can arise during the migration. Notice that Guideline A refers only to ZBRs, since if they use intra-zone paths, then non-ZBR routers cannot use inter-zone paths. Establishing multiple adjacencies (e.g., $L1L2$ adjacencies in IS-IS) between ZBRs also guarantees the $nh$ function does not change, but backbone links could be unnecessarily traversed in this case.

**Guideline B**  In each zone $Z$, the weight of the path from any ZBR to any destination in $Z$ is the same.

Practically, Guideline B can be enforced by organizing routers in each peripheral zone $Z$ in three layers: i) a core layer, containing ZBRs in $Z$; ii) an aggregation layer, connecting the access and the core layers; and iii) an access layer, containing destinations in $Z$. Each ZBR must connect to at least one router in the aggregation layer, and each router in the aggregation layer must connect to all destinations in $Z$. In addition, all core-to-aggregation links must have the same weight $w_1$; similarly, all aggregation-to-access layer link weight must be set to the same value $w_2$ (possibly $w_2 \neq w_1$). Guideline B guarantees easy IGP migrations when route summarization is introduced or removed. We assume that aggregated prefixes are announced with a cost equal to the highest weight among the destinations in the aggregate (as in OSPF, by default [Moy98]). In this case, both with and without summarization, each
backbone router chooses the closest ZBR in $Z$ as entry point for destinations in the aggregated prefix. It is easy to check that, as a consequence, the $nh$ function does not change with or without summarization, hence no specific migration ordering is needed during the migration.

7.10 Related Work

Seamless IGP operation and maintenance have been the focus of several previous studies. For example, several protocol extensions have been proposed [SDV06, MPEL03, SG08] to gracefully restart a routing process. However, few research effort has been specifically devoted to network-wide seamless IGP migrations, and current best practices [Pep07, Hv10] are just rules of thumb which do not apply in the general case, and do not guarantee lossless reconfiguration processes.

In [RZC09] and [RZC11], Raza et al. propose a theoretical framework to formalize the problem of minimizing a certain disruption function (e.g., link congestion) when the link weights have to be changed. The authors also propose a heuristic to find an ordering in which to modify several IGP weights within a network, so that the number of disruptions is minimal. Although their work is close in spirit to ours, the migration scenarios we analyzed cannot always be mapped to a reweighting problem. For example, in hierarchical IGP configurations, both the weight of a link and the zone to which it belongs are considered in the computation of the next-hop from a router to a destination and a unique link weight assignment that generates the same next-hop for each router-to-destination pair may not exist. A more abstract reconfiguration framework on how to transform a feasible solution of a problem into another solution of the same problem is also studied from a purely theoretical point of view (e.g., [KMM11]).

In [KCF06], Keralapura et al. study the problem of finding the optimal way in which devices and links can be added to a network to minimize disruptions. Beyond addressing topological changes, our techniques can be used to address several other migration scenarios.

In [CMMvdM09], Chen et al. describe a tool that is able to automate status acquisition and configuration change on network devices according to rules specified by domain experts. The tool can be used to automate the ships-in-the-night approach, but not to compute a loop-free ordering. The authors also provide a rule of thumb to avoid problems during IGP migrations, i.e., update edge routers before the backbone ones. However, this rule does not
CHAPTER 7. ENABLING NETWORK-WIDE IGP RECONFIGURATIONS

hold in general. For example, migrating $E_1$ before $B_1$ in Fig. 7.1 creates a forwarding loop in a hier$	o$flat scenario.

In [AWY08], Alimi et al. extend the ship-in-the-night approach by allowing multiple configurations to run simultaneously on a router. They also describe a commitment protocol to support the switch between configurations without creating forwarding loops. While the technique is promising, it cannot be exploited on current routers.

Recently, some techniques [WKB+08, KRvdM10] have been proposed to enable virtual routers or parts of the configuration of routers (e.g., BGP session) to be moved from one physical device to another. Their works differ from ours as we aim at seamlessly changing network-wide configurations.

In [RFRW11], Reitblat et al. study the problem of consistent network updates in “Software Defined Networking”. They propose a set of consistency properties and show how these properties can be preserved when changes are performed in the network. Unlike our approach, this work only applies to logically-centralized networks (e.g., OpenFlow).

Regarding the problem of avoiding forwarding loops in IGPs during transient states, some previous work has also been done. Francois et al. propose protocol extensions that allow routers to update their FIB without creating a transient loop after a link addition or removal [FSB07]. Fu et al. [FSK08] and Shi et al. [SFF09] generalize the results by defining a loop-free FIB update ordering for any change in the forwarding plane and considering traffic congestion, respectively. However, these approaches cannot be used effectively in practice to perform IGP migrations since they only consider updating the FIB on a per-destination basis. Our approach is different as it is aimed at minimizing the number of changes applied to the routers’ configurations by searching for a per-router migration ordering. We only apply a per-destination ordering when no per-router migration exists.

IGP migrations could also be performed by using route redistribution. Although new primitives have been recently proposed [LXZ10], we believe that relying on a ships-in-the-night approach (when possible) makes the entire migration process easier and more manageable.

7.11 Limitations and Future Work

In this section, we discuss limitations of our methodology, especially in terms of its application to other types of migrations. Such limitations help us in defining interesting research directions opened by this work.
7.11. LIMITATIONS AND FUTURE WORK

First of all, our methodology is adapted to link-state IGP migrations and it cannot be directly applied to other IGPs (e.g., distance-vector IGPs). Contrary to link-state protocols, where routers always have a global view of the topology and can take decisions autonomously, in distance-vector protocols a change of the next-hop of one router can affect the visibility other routers have of some destinations. This is likely to pose different problems with respect to those tackled by our techniques.

In this chapter, we have defined and evaluated extensions of our techniques that deal with network failures. Intuitively, link congestion due to the applied migration ordering can be avoided with similar extensions, i.e., adding constraints to the migration ordering research space. We plan to fully investigate and evaluate provably congestion-free migration techniques in future work.

Moreover, our methodology does not take into account the interactions between the changing IGP and the protocols relying on it. In particular, our approach is not suitable for all the possible scenarios in which BGP is deployed on the migrated network. BGP uses the IGP to both discriminate among several exit-points and to learn how to reach the preferred exit-point [RLH06]. Migrating the underlying IGP can thus cause BGP routers to change their preferred exit-point which can lead to forwarding loops. Currently, our algorithms ensure that no loop occurs during the migration towards any internal destination of an AS. This property is sufficient to guarantee loop-free forwarding towards inter-domain destinations as well in the following two cases: i) inter-domain routes are redistributed (possibly in an aggregated form) in the IGP, as may be the case in enterprise networks, and ii) forwarding is based on tunneling or encapsulation mechanisms, as for example in MPLS-based networks. Though, in the migration of a pure IP network, the exclusive presence of BGP can induce forwarding loops due to conflicting BGP decisions between updated routers and non-updated routers.

Theoretically, our ordering algorithms can be adapted to deal with BGP-induced loops due to IGP configuration changes. However, the ordering problem is much more complicated since it needs to consider: i) the fact that BGP prefixes could be reached via any combination of exit-points, ii) the iBGP topology and its relationship to the IGP [GW02b], and iii) BGP dynamism. For these reasons, we expect that a loop-free migration ordering for all the BGP prefixes does not exist in most of the cases.

We believe, however, that finding an effective technique to prevent BGP-induced loops during the IGP migration of pure IP networks is an interesting open problem raised by this work.
7.12 Conclusions

Network-wide link-state IGP migrations are a source of concerns for network operators. Unless carried with care, IGP migrations can cause long-lasting forwarding loops, hence significant packet losses.

In this chapter, we proposed a migration methodology that enables operators to perform network-wide changes on an IGP configuration seamlessly, rapidly, and without compromising routing stability. Our methodology relies on effective techniques for the computation of a router migration ordering, and on a provisioning system to automate most of the process. These techniques encompass a complete, time-consuming algorithm and a heuristic. The evaluation we performed on several ISP topologies confirms the practical effectiveness of both the heuristic and the provisioning system. We also evaluated extensions of our techniques that prevent long-lasting loops even in case of network failures.

As future work, we plan to refine our techniques in order to deal with link-state IGP migrations in advanced scenarios, like congestion-free reconfigurations and prevention of BGP-induced loops during IGP reconfigurations of pure IP networks. Also, we plan to study reconfigurations of other IGPs, like distance-vector IGPs or EIGRP.
Chapter 8

Towards an Optimized iBGP Configuration *

8.1 Introduction

iBGP configuration plays a key role in large ISP networks, as iBGP is used to disseminate the interdomain routes received from other ISPs. Unfortunately, iBGP configuration optimality depends on many factors, like IGP topology and IGP metrics. Those factors, in turn, evolve with the network. Hence, when new requirements appear or when router capabilities change, network operators often need to redesign their iBGP configuration.

In this chapter, we analyze two design proposals to tweak iBGP route reflection. Those proposals are targeted to make iBGP more flexible, and adapt it to practical needs of operators, like increasing route diversity and internally engineering inter-domain traffic. We show the impact of such proposals on configuration correctness properties, among which guaranteed convergence, and absence of traffic blackholes and forwarding loops. In Section 8.2 we describe the model for iBGP configurations we adopt in this chapter, and we formally define correctness properties as formalized in [GW02b]. In particular, signaling

correctness ensures that BGP will always converge to a single stable routing, while forwarding correctness guarantees the absence of packet deflections along the forwarding path. Also, we formally define known sufficient conditions for iBGP configuration correctness.

Then, we devote the first part of the chapter to the first design proposal, consisting in adding spurious iBGP sessions. We denote as spurious sessions those iBGP sessions that do not entirely respect the neat hierarchical structure of route reflection topologies as originally conceived in [BCC06]. Section 8.3 describes the main reasons for adding iBGP spurious sessions, among which fixing route visibility issues due to route reflection and improving route diversity [PTOS08, PUT+10]. These reasons also justify the presence of some spurious sessions in real world ISP networks as reported in [FB05, PCA+11]. We then study the impact of spurious sessions on iBGP correctness properties in Section 8.4. We show that the additional spurious sessions can negatively impact both iBGP stability and forwarding correctness. Also, we find that route propagation rules play a fundamental role in ensuring correctness of iBGP configurations with route reflection in this scenario. This contradicts the assumption, made in most previous research work, that all routers are guaranteed to receive a route to any destination prefix in iBGP topology if a stable state is reached. Through simple examples, we show that traffic blackholes can be created by the combined effect of iBGP route propagation rules and the route selection algorithm. Even worse, our examples show that distinct destination prefixes cannot be analyzed separately in the general case, as their interaction can give rise to forwarding anomalies. Hence, we define a new correctness property, called dissemination correctness, to model the absence of anomalies due to iBGP route propagation rules, and we show how dissemination correctness fills the gap between signaling and forwarding correctness. Unfortunately, checking dissemination correctness is computationally intractable even when adding a single iBGP session to an existing dissemination correct iBGP configuration, as we prove in Section 8.5.

After, we switch to study the possibility of changing the attributes carried by iBGP messages: we refer to this proposal as iBGP attribute changing (IAC). In Section 8.6, we discuss possible advantages of changing iBGP attributes. Also, 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 some ISPs, especially large transit providers. In Section 8.7, we study the impact of IAC on the iBGP stability guarantees, proving that changing iBGP attributes makes iBGP prone to new types of oscillations. Further, we re-apply techniques described
in Chapter 4 to build a prototype convergence checker. To the best of our knowledge, this is the first tool able to statically check an iBGP configuration for stability in case of iBGP attribute changing, given that state-of-the-art algorithms to detect oscillations [FRBS08] assume that iBGP messages are left untouched. Experimental 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.

Further, we study sufficient conditions and we propose configuration guidelines that overcome drawbacks and establish fundamental tradeoffs (e.g., between stability guarantees and suboptimal routing likelihood) when operators are willing to deploy the analyzed iBGP design proposals. Namely, in Section 8.8, we study sufficient conditions that ensure both signaling and dissemination correctness and we use them to define design guidelines. We find that the absence of a special type of spurious iBGP sessions (i.e., spurious OVER sessions) guarantees no iBGP route propagation anomalies. Also, we state configuration guidelines to change iBGP attributes in a profitable 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.

Finally, related work is revised in Section 8.9, and conclusions are drawn in Section 8.10.

8.2 A Model for iBGP Configuration

We now present the model we use in this chapter for iBGP route reflection configurations. We refer the reader to Chapter 1 for some background on iBGP and route reflection.

We model an IGP graph as an undirected weighted graph \( I = (V,E) \), with a weight associated to each edge \( (u,v) \in E \). Also, we denote with \( \text{dist}(u,v) \) the total weight of the shortest path from \( u \) to \( v \).

We refer to the organization of iBGP sessions in an iBGP configuration as iBGP topology. We model an iBGP topology as a directed labeled multigraph \( B = (V,E) \) where nodes in \( V \) represent routers and edges in \( E \) represent iBGP sessions. Each edge \( (u,v) \) is associated with a label which is either UP, DOWN, or OVER. We use \( u \leftarrow v \), \( u \rightarrow v \), and \( u \leftrightarrow v \) to indicate that the label of edge \( (u,v) \) is DOWN, UP or OVER, respectively. Because of the way iBGP relationships are defined, we have that \( u \leftarrow v \Leftrightarrow v \rightarrow u \), while \( u \leftrightarrow v \Leftrightarrow v \leftrightarrow u \).
Due to the iBGP route dissemination rules (see Table 1.2), not every path on B can be used to distribute a BGP message. We define a valid signaling path as a path \((u \ldots v)\) on B that can be used to advertise routes from \(u\) to \(v\) (or vice versa). A valid signaling path consists of zero or more UP sessions, followed by zero or one OVER session, followed by zero or more DOWN sessions, that is, it matches regular expression \(UP^*OVER?DOWN^*\) [BUM08]. The presence of a valid signaling path between \(u\) and \(v\) is a necessary condition for \(u\) to learn routes announced by \(v\), even if we show in Section 8.4 that it is not a sufficient condition. Throughout the chapter, we assume that the iBGP graph \(B\) is connected, that is, \(\forall u, v \in B\) there is a valid signaling path from \(u\) to \(v\), otherwise obvious forwarding anomalies can arise (routes are not propagated network-wide). Whenever it is clear from the context, we use a signaling path to refer to the route advertised over that signaling path (e.g., we say that a router receives a path, or prefers a path over another).

Signaling paths can be concatenated. The concatenation of two non-empty signaling paths \(P = (v_k v_{k-1} \ldots v_i), k \geq i,\) and \(Q = (v_i v_{i-1} \ldots v_0), i \geq 0,\) is signaling path \(PQ = (v_k v_{k-1} \ldots v_i v_{i-1} \ldots v_0).\) The concatenation of two valid signaling paths can be a non valid signaling path. In the remainder of this chapter, we use the word “path” to refer to an “iBGP signaling path”, except when explicitly specified.

Route reflection topologies are usually organized in a hierarchy where there are no cycles consisting of UP sessions only. Indeed, such cycles are a sign of bad topology design and can create routing anomalies [GW02b]. In a hierarchy, each BGP router can be assigned to a layer. We denote the set of routers in the top layer of an iBGP topology \(B\) as \(T_B.\) A router belongs to the top layer \(T_B\) if it has no route-reflectors. We also define a function \(L\) that maps a router \(u\) to a hierarchical layer:

\[
L(u) = \begin{cases} 
0 & \text{if } u \in T_B \\
\max_{\{v\mid u \rightarrow v\}} (L(v)) + 1 & \text{otherwise}
\end{cases}
\]

**iBGP Configuration Correctness**

It has been shown [GW02b] that the suboptimal route visibility introduced by route reflection can cause both routing and forwarding anomalies. Routing anomalies can prevent BGP to settle to a stable state because of routing oscillations. Moreover, inconsistent routing decisions between the forwarding plane and the control plane can create forwarding deflections and loops. A BGP
configuration is said to be signaling correct if it is free from routing anomalies, i.e., if BGP is guaranteed to converge to a single predictable stable state. A signaling correct configuration is forwarding correct if it is free from forwarding anomalies. Observe that there are no guarantees that all the routers have a route towards all the prefixes even in a signaling correct BGP configuration.

The following set of sufficient conditions guarantees that an iBGP topology \( B \) is both signaling and forwarding correct [GW02b].

(i) \( B \) has no cycles consisting of UP sessions only;

(ii) any route-reflector prefers paths propagated by its clients over paths propagated by non-clients; and

(iii) all shortest paths must also be valid signaling paths.

Conditions (i) and (ii) ensure that the iBGP configuration is signaling correct, while Condition (iii) guarantees forwarding correctness. Although interesting from a theoretical perspective, such conditions can be too constraining to be applied in real networks. For example, Condition (iii) practically forces the BGP topology to be congruent to the IGP one, in such a way that even a full-mesh of iBGP sessions [RLH06] is not compliant. We discuss the applicability of Condition (ii) in Section 8.8.

In [BMU07, BUM08] the concept of fm-optimality is introduced as a relaxed sufficient condition to ensure forwarding correctness in a signaling correct iBGP configuration. To understand fm-optimality, we need to define white routers and white paths [BUM08]. Given an iBGP topology \( B \), a router \( r \) and an egress point \( e \), a router \( r' \) is said to be a white router for pair \( (r, e) \) if there is no egress point \( e' \) in \( B \) such that \( \text{dist}(r, e') > \text{dist}(r, e) \) and \( \text{dist}(r', e') \leq \text{dist}(r', e) \). A white path between a router \( r \) and an egress point \( e \) is defined as a valid signaling path between \( r \) and \( e \) that contains only white routers for pair \( (r, e) \). An iBGP topology is fm-optimal if for each router \( r \) and for each egress point \( e \) there exists at least one white path.

8.3 Proposal 1: Adding Spurious iBGP Sessions

Consider the iBGP network depicted in Fig. 8.1. The iBGP topology consists of route-reflectors \( a, b, \) and \( r \) (drawn in grey), and border routers: \( e_1 \) and \( e_2 \) which are clients of \( b \), and \( e_3 \) which is client of \( r \). The route reflection hierarchy is organized in three levels, the top layer being composed by \( a \) and
CHAPTER 8. TOWARDS AN OPTIMIZED IBGP CONFIGURATION

Figure 8.1: A BGP network in which route-reflection can limit route visibility.

Figure 8.2: An abstract representation, drawn according to the graphical formalism we use in this chapter, of the network in Fig. 8.1.

r. Observe that the absence of a direct session between a and any egress point forces a to rely on b’s and r’s routing decision. This involves that not all the external routes received at the edge of the network, that is, a has a limited route visibility. Depending on the IGP topology, limited route visibility can also create forwarding issues.

In order to deeper understand route visibility and forwarding anomalies, consider Fig. 8.2. Fig. 8.2a depends iBGP topology according to the graphical convention we will use in the following. Circles represent clients, while diamonds represent route-refectors. UP and OVER sessions are depicted with
single and double arrow links, respectively. The dashed arrows labeled $p_1$ entering routers $e_1$, $e_2$ and $e_3$ represent the fact that $e_1$ and $e_2$ are egress points for prefix $p_1$. Similarly, $e_3$ is an egress points for both $p_1$ and $p_2$. The underlying IGP graph is depicted in Fig. 8.2b, where lines represent IGP links and labels represent the IGP weight assigned to a link. Links without labels are implicitly assumed to have IGP weight 1.

Consider prefix $p_1$. Routers $e_1$, $e_2$ and $e_3$ will select their external route $R_1$, $R_2$ and $R_3$, respectively, due to step 5 of the BGP decision process. Therefore, they will advertise their best route to all their iBGP neighbors, namely $b$ (for $e_1$ and $e_2$) and $r$ (for $e_3$). Router $b$ will collect route advertisements from its clients, select its best route, and then propagate it to every other neighbor. By step 6 of the BGP decision process, $b$ will select route $R_2$ because $e_2$ is a closer egress point than $e_1$. Therefore, by iBGP propagation rules (see Table 1.2) $b$ will advertise $R_2$ to its route-reflector $a$. Each router will keep performing route collection, route selection and route dissemination until BGP converges and no further messages are propagated. After convergence, router $r$ will select route $R_3$ and router $a$ will select route $R_2$.

Observe that router $a$ has no knowledge of route $R_1$, because it only receives route $R_2$ from $b$ and route $R_3$ from $r$. In fact, route reflection introduces suboptimal route visibility and limits the amount of route diversity available at router $a$. Another side effect induced by route reflection is the packet deflection that happens when $a$ sends traffic to prefix $p_1$. More precisely, $a$ believes that the traffic will exit from egress point $e_2$ and forwards it to $e_1$ because it is the next hop to $e_2$. However, $e_1$ is itself an egress point for prefix $p_1$, so it will deflect traffic outside the ISP. The combination of multiple deflections can result in forwarding loops [GW02b].

Whenever issues due to suboptimal route visibility arise, fixing them by adding additional iBGP sessions may look like an easy and tempting solution for a network operator. In our example, adding an iBGP session between routers $a$ and $e_1$ will provide $a$ with increased route diversity and will make it able to select its optimal egress point. The addition of OVER sessions to increase route diversity in iBGP has been already proposed in [PTOS08, PUT+10], e.g., to support recently proposed techniques for reducing iBGP convergence time [FMB+07]. Indeed, quantitative studies have already shown that route reflection leads to very poor route diversity [UT06]. This, in turn, can cause high convergence time in case of failure or interdomain routing changes. Moreover, additional sessions can provide better route visibility to routers, thus making it easier for a network operator to fix its iBGP configuration in order to comply with state of the art guidelines [BMU07].
Observe that, in general, additional iBGP sessions do not need to be OVER sessions, i.e., they could be UP sessions as well. However, network operators might prefer OVER sessions due to the fact that they incur lower memory overhead and lower update churn, because only a subset of reflected routes is announced on OVER sessions (see Table 1.2).

Unfortunately, adding OVER sessions to an existing topology may have undesirable side effects. Consider the iBGP network in Fig. 8.3 (OVER-RIDE GADGET), which is a simplified version of the one in Fig. 8.2. An additional OVER session exists between routers $a$ and $e$. Since $e$ is the only egress point available for prefix $p$, $a$ will prefer the route that it learns on the OVER session because of step 8 of Table 1.3. Then, since its best route was learned from a peer, $a$ will not propagate it to $r$, so $r$ will have no route to prefix $p$.

Now if $r$ has a route for a less specific prefix than $p$ (e.g., a default route), it will use that route for traffic destined to $p$, possibly generating forwarding deflections and loops. As a consequence, it is not safe to assume that prefixes are independent in iBGP. Otherwise, if $r$ has no route for a less specific prefix than $p$, $r$ will create a traffic blackhole. Observe that both kinds of anomalies are due to the iBGP topology alone: IGP topology is irrelevant because there is only one egress point for $p$. For this reason, the OVER-RIDE GADGET complies with the conditions of [BMU07], yet it is subject to anomalies. Even worse, such anomalies can be triggered by external events, e.g., if an egress point fails.
8.4 The Impact of iBGP Spurious Sessions

In this section, we introduce the concept of spurious OVER sessions and we show how their side effects can invalidate some simple assumptions that apparently hold in any iBGP topology, and that have been used in previous research work.

Definition 8.1 Given a BGP topology $B$, an OVER session $x \leftrightarrow y$ is spurious if one of the two routers is not in the top layer, i.e., if $x \notin T$ or $y \notin T$.

Spurious sessions are not common in Internet networks and vendor guidelines suggest to not deploy them [ZB03]. Nevertheless, spurious sessions have been proposed to solve visibility issues [PTOS08, PUT+10], and previous work shows that large ISPs sometimes use them [FB05, PCA+11]. Moreover, spurious sessions can be unintentionally created during iBGP reconfigurations. For example, current best practices to deploy route reflection from an iBGP full-mesh [Hv10] are very likely to generate spurious OVERs in intermediate configurations.

Route Dissemination Deceptions

As discussed in Section 8.3, the over-ride gadget provides an example of how a spurious OVER improves egress point visibility at some routers, but potentially worsens visibility at other routers. In the gadget, the side effect of adding a spurious OVER is counter-intuitive because it induces a change in the route dissemination process at router $r$ without affecting the egress point selected by $r$. This contradicts the intuition that a connected iBGP topology guarantees that every router eventually learns at least one route for any given prefix.

Unfortunately, some previous work is based on that intuition. In particular, [PTOS08] assumes that adding an OVER session can only improve route visibility, while [BMU07, BUM08] assume that a route-reflector $r$ can “hide” a route to a neighboring router $v$ only if it has a closer alternative egress point.

More generally, spurious OVER sessions show that the concept of valid signaling path is not a good abstraction to study the actual ability of a router to learn a route to a given prefix. In order to better understand this property, we introduce the concept of dissemination correctness.

Definition 8.2 Let $B$ be a signaling correct iBGP topology. Then, $B$ is dissemination correct if all the routers in $B$ are guaranteed to receive at least one
route for prefix p in the stable state, for any non-empty set of egress points for p.

Observe that dissemination correctness does not depend on interdomain routing nor on the set of egress points currently learning routes for given prefixes. That is, it is a topological property. Dissemination correctness differs from both signaling and forwarding correctness. Indeed, a signaling correct topology is not guaranteed to be dissemination correct. Moreover, a dissemination correct topology is not guaranteed to be forwarding correct. The three properties actually complement each other: signaling correctness deals with routing anomalies that can prevent BGP from converging; dissemination correctness deals with issues in the route propagation process; forwarding correctness deals with forwarding anomalies caused by the interaction between iBGP and IGP.

Signaling and Forwarding Correctness Deceptions

Beside affecting dissemination correctness, a single spurious OVER can even prevent an iBGP topology to be either signaling or forwarding correct, as shown in Fig. 8.4.

Consider Fig. 8.4a. Every router is equipped with a list of valid signaling paths, sorted in decreasing order of preference. Observe that $\langle u_1, e_0 \rangle$ is a spurious OVER session. We now show that iBGP cannot converge in this
8.4. THE IMPACT OF IBGP SPURIOUS SESSIONS

configuration. Assume by contradiction that a stable state exists, and consider
the routing choice at router \( u_2 \). Since \( u_2 \) receives a route directly from \( e_2 \), it
is not possible that \( u_2 \) does not select any route for prefix \( p_1 \). Hence, we have
the following cases.

- \( u_2 \) steadily selects \((u_2 e_2)\). In this case, \( u_1 \) will use its most preferred path
\((u_1 u_2 e_2)\), preventing \( u_0 \) from selecting \((u_0 u_1 e_0)\). Thus, \( u_0 \) will select
\((u_0 x e_0)\), and eventually announce it to \( u_2 \). Because of path preferences,
\( u_2 \) should switch to \((u_2 u_0 x e_0)\), yielding a contradiction.

- \( u_2 \) steadily selects \((u_2 u_1 u_0 x e_0)\). This involves that \( u_1 \) steadily selects
\((u_1 u_0 x e_0)\), leading to a contradiction, since path \((u_1 e_0)\) is always
available at \( u_1 \) and is more preferred than \((u_1 u_0 x e_0)\).

- \( u_2 \) steadily selects \((u_2 u_0 x e_0)\). This implies that \( u_0 \) steadily selects
\((u_0 x e_0)\), and \( u_1 \) is forced to select \((u_1 e_0)\), since it does not receive path
\((u_2 e_2)\) from \( u_2 \). This leads to a contradiction, since \( u_0 \) will eventually
learn and select path \((u_0 u_1 e_0)\), preventing \( u_2 \) from steadily selecting
\((u_2 u_0 x e_0)\).

All the cases lead to a contradiction, hence a stable state does not exist in the
topology in Fig. 8.4a. Observe that the path preferences highlighted in the
figure can result from the standard BGP decision process (Table 1.3) if the
IGP topology is such that \( \text{dist}(x,e_0) < \text{dist}(x,e_2) \), \( \text{dist}(u_0,e_0) < \text{dist}(u_0,e_2) \),
\( \text{dist}(u_2,e_0) < \text{dist}(u_2,e_2) \), and \( \text{dist}(u_1,e_0) = \text{dist}(u_1,e_2) \). In this case, \( x, u_0 \),
and \( u_2 \) prefer paths based on the closest egress point, while \( u_1 \) prefers eBGP
routes received from \( e_2 \) over those received from \( e_0 \) for egress-id. Ties are
broken by shorter cluster-list and peer-id criteria. Also notice that, in
such a configuration, the iBGP topology in Fig. 8.4a is fm-optimal [BMU07].
For each router, its white paths for both egress points \( e_0 \) and \( e_2 \) are marked
with an asterisk.

Forwarding correctness can also be affected by the presence of spurious
OVER sessions. Consider the topology in Fig. 8.4b, and assume that \( x \) steadily
selects path \((x e_2)\), while \( z \) steadily selects path \((z e_0)\), because of the IGP
distances. Since those paths are learned via an OVER session, \( x \) and \( z \) will not
propagate their best route to \( y \), hence \( y \) will be forced to select the route from
\( e_1 \). If \( y \) is on \( x \)’s shortest path to \( e_2 \) and \( x \) is on \( y \)’s shortest path to \( e_1 \), then
a loop for \( p_1 \) arises between \( x \) and \( y \).
8.5 Checking Dissemination Correctness is Hard

In this section, we study the computational complexity of deciding whether a given IBGP topology is dissemination correct. Unfortunately, we find that such a problem is computationally intractable. Even worse, we show that the problem of deciding if the addition of a single session can affect the dissemination correctness of an IBGP topology is also computationally intractable.

More formally, we define the two problems we consider as follows.

Dissemination Correctness Problem (DCP): Given a signaling correct IBGP topology $B$ and the underlying IGP topology $I$, decide if $B$ is dissemination correct.

One More Session Problem (OMSP): Given a dissemination correct IBGP topology $B = (V, E)$, the underlying IGP topology $I$, and a spurious OVER session $o = (x, y), x, y \in V$, decide if $B' = (V, E \cup (x, y))$ is dissemination correct.

Observe that DCP is the IBGP equivalent of the reachability problem defined on eBGP in [GW99].

We now prove that DCP is coNP-hard [Pap94]. Intuitively, computational complexity of DCP mainly depends on the fact that all the non-empty sets of egress points have to be checked in the general case. In the following, we show that the SAT COMPLEMENT problem [Pap94] can be reduced to DCP in polynomial time. Consider an instance of SAT COMPLEMENT and let $F$ be a logical formula in conjunctive normal form. Moreover, let $C_1, \ldots, C_n$ be the clauses in $F$, and let $X_1, \ldots, X_m$ be the boolean variables appearing in the clauses. Each clause $C_i$ is the logical disjunction of exactly 3 literals $L_{ij}$ with $j = 1, 2, 3$. A literal $L_{ij}$ can be either a variable $X_i$ or a negated variable $\bar{X}_i$.

The SAT COMPLEMENT problem consists in deciding if $F$ is unsatisfiable, that is, if no boolean assignment makes $F$ true.

We now build the corresponding instance of DCP (see Fig. 8.5), following an intuition similar to that used in [GW02b] for proving that signaling correctness is NP-hard. The skeleton of the IBGP topology $B = (V, E)$ consists of 4 nodes, $e, s, r$, and $b$ connected as in the OVER-RIDE GADGET. In particular, $e \to s$, $s \to r$, and $e \leftrightarrow r$. Moreover, $r \leftrightarrow b$ since $b, r \in T$. For each variable $X_i$, we add two literal nodes $x_i$ and $\bar{x}_i$ to $V$, representing the two literals associated to $X_i$. For each clause $C_j$, we add a clause node $c_j$ and three nodes $v_{j1}, v_{j2},$ and $v_{j3}$. We add OVER sessions between $c_j$ and $v_{ji}, i \in \{1, 2, 3\}$. Also, each $c_j \in T$, hence it is in the top layer full-mesh. We also add an UP session from $e$ to $c_j$. Moreover, two UP sessions $(x_k, v_{ji})$ and $(\bar{x}_k, v_{ji})$ are added to $E$ iff either $X_k$ or $\bar{X}_k$ is the $i^{th}$ literal appearing in clause $C_j$. Finally, we add an
8.5. **CHECKING DISSEMINATION CORRECTNESS IS HARD**

8.5.1. Reduction from SAT Complement to DCP

UP session between each $v_{ji}$ and $r$.

We set IGP metrics as follows. Consider any clause $C_j$. If variable $X_i$ appears unnegated in the $k^{th}$ literal of $C_j$, then we set $dist(c_j, x_i) < dist(c_j, e) < dist(c_j, \bar{x}_i)$, and $dist(v_{jk}, x_i) < dist(v_{jk}, x_i)$. For any router $n \neq e, x_i, \bar{x}_i$, we set $dist(c_j, \bar{x}_i) < dist(c_j, n)$ and $dist(v_{jk}, x_i) < dist(v_{jk}, n)$. Otherwise, if variable $X_i$ appears negated in the $k^{th}$ literal of $C_j$, we set IGP metrics such that $x_i$ is replaced with $\bar{x}_i$ and vice versa in the above inequalities. Finally, we set IGP metrics in such a way that $r$ and $s$ prefer routes announced by $e$ over all other routes. Fig. 8.5b shows an example of the IGP topology resulting from a clause $C_1$ in which $X_j$ ($X_l$, resp.) appears negated (unnegated, resp.).

Intuitively, a boolean assignment $M$ corresponds to a set $S_M$ of egress points for a given prefix $p$. Router $x_i$ ($\bar{x}_i$, resp.) belongs to $S_M$ iff $X_i$ is true (false,
A SAT complement instance can be reduced to a DCP one in polynomial time, since each clause and each variable is mapped to a polynomial number of routers and links. We now show that the reduction is correct.

**Lemma 8.1**  
*B is signaling correct. Moreover, if e is not an egress point for a given prefix p, then all routers in B are guaranteed to receive a route to p.*

**Proof:** Consider prefix *p* and let \( S \neq \emptyset \) be the set of egress points for *p*. Abusing the notation a bit, we refer to routers *x\( j \) and \( x_i \) as to *x*-routers, and similarly we refer to *v*-routers and *c*-routers. We have two cases.

First, assume \( e \notin S \). In this case, all *x*-routers in \( S \) steadily select their external route and announce it to their respective route-reflectors. The *v*-routers that have at least one client in \( S \) steadily select the route propagated by one of their client, because of IGP metrics. Router \( r \) receives routes from all *v*-routers that have at least one client in \( S \). Since \( S \neq \emptyset \), we conclude that \( r \) is able to select a route to \( p \) announced by a *v*-router. Router \( r \)'s best route is then forwarded to all \( r \)'s neighbors, because it was learned from a client. Observe that all the shortest paths from \( r \) to a router in \( S \) contain \( s \), which implies that \( s \) will select the same route as \( r \). For this reason, \( e \) will receive the same route from \( s \) and from \( r \) and will steadily select it. Every *c*-router learns a route from \( r \) and possibly additional routes from its *v*-peers. In any case, *c*-routers’ best routes can only be propagated to \( s \) due to iBGP route reflection rules. Moreover, this cannot affect the route selected by \( s \). Router \( b \) and *v*-routers having no clients in \( S \) receive a single route, i.e., the one announced by \( r \). This, in turn, implies that *x*-routers that are not in \( S \) will also receive a single route. Hence, iBGP is guaranteed to converge, and all routers in \( B \) learn at least a route to \( p \) in the steady state.

We now consider the case in which \( e \in S \). Again, all routers in \( S \) steadily select their external route and announce it to their respective route-reflectors. The *v*-routers that have at least one client in \( S \) steadily select the route propagated by one of their client, because of IGP metrics. Let \( R_e \) be the route learned by \( e \). Observe that IGP metrics imply that routers \( r \) and \( s \) will steadily select \( R_e \). Every *c*-router learns at least \( R_e \) from router \( s \), and possibly additional routes from their *v*-peers. In any case, *c*-routers’ best routes are propagated to \( s \) only, which prefers \( R_e \) due to IGP metrics. Again, *v*-routers having no clients in \( S \) receive a single route from \( r \), i.e., \( R_e \). This, in turn, implies that *x*-routers that are not in \( S \) also receive a single route. Observe that router \( b \)
may or may not learn a route to prefix $p$. Since $b$'s decision cannot influence any other router, we conclude that iBGP is guaranteed to converge. □

**Theorem 8.1** DCP is coNP-hard.

Proof: Consider a logical formula $F$ and construct the corresponding DCP instance $B = (V, E)$, as described above. By Lemma 8.1, $B$ is signaling correct. Moreover, Lemma 8.1 implies that $B$ is dissemination correct if router $e$ does not receive an external route. Hence, we focus on the case in which router $e$ receives an external route. We now prove the statement in two parts.

**If $F$ is unsatisfiable then $B$ is dissemination correct.**

Assume by contradiction that $B$ is not dissemination correct, i.e., there exists a set of egress points $S_M$ for prefix $p$ such that at least one router in $B$ does not receive any route to $p$. By Lemma 8.1, we know that such a router must be $b$. We now build a boolean assignment $M$ that satisfies $F$, yielding a contradiction.

Since $b$ does not receive any route, each $c_i$ does not select the route received by $e$, otherwise it would have propagated that route to $b$. Hence, all $c_i$ must select a route learned by one of their peers $v_{ik}$, with $k = 1, 2, 3$.

Let $C_j$ be a clause and assume that $X_i$ appears unnegated in the $k^{th}$ literal of $C_j$. Then, router $c_j$ selects a route propagated by $v_{jk}$ only if $v_{jk}$ selects the route originated by $x_i$, since $dist(c_j, x_i) < dist(c_j, e) < dist(c_j, \bar{x}_i)$. In turn, router $v_{jk}$ selects the route originated by $x_i$ only if $x_i$ is an egress point for $p_M$ and $\bar{x}_i$ is not, since $dist(v_{jk}, x_i) < dist(v_{jk}, \bar{x}_i)$. Symmetrical considerations hold if $X_i$ appears negated in the $k^{th}$ literal of $C_j$. In both cases, we are able to find a boolean assignment to variable $X_i$ that makes clause $C_j$ true.

Iterating the same argument on all the clauses, we map $p_M$ to a boolean assignment $M$ which satisfies $F$.

**If $F$ is satisfiable then $B$ is not dissemination correct.**

Let $M$ be a boolean assignment that satisfies $F$. We now show that $B$ is not dissemination correct, since there exists a set $S_M$ of egress points such that if a prefix $p$ is learned at $S_M$ then $b$ receives no route to $p$.

By definition of $M$, all clauses are satisfied in $M$, hence for any clause $C_j$ at least one literal must be true. Assume, without loss of generality, that the $k^{th}$ literal of $C_j$ is true. If the $k^{th}$ literal of $C_j$ is $X_i$, then we impose that router $x_i$ receives an eBGP route $R_k$ to prefix $p_M$, while router $\bar{x}_i$ does not receive any eBGP routes to $p_M$. Since $dist(v_{jk}, x_i) < dist(v_{jk}, e)$, router $v_{jk}$ selects route $R_k$ and propagates it to router $c_j$. Similarly, since $dist(c_j, x_i) < dist(c_j, e)$, $c_j$ selects route $R_k$. Otherwise, if the $k^{th}$ literal of $C_j$ is $\bar{X}_i$, we can apply
the same argument by replacing $x_i$ with $\bar{x}_i$. In both cases, $c_j$ selects a route propagated by an iBGP peer.

Since the above argument applies to all clauses, we have that every $c_j$ selects a route learned from an iBGP peer. Router $r$ also selects a route learned from an iBGP peer, because of the presence of OVER session $(r, e)$ (see Section 8.3). Hence, every router which is a neighbor of $b$ selects a route learned from an iBGP peer, thus $b$ receives no route for prefix $p_M$. □

A similar reduction can also be used to show that $OMSP$ is coNP-Hard. Starting from a logical formula in conjunctive normal form, we build the instance of the $OMSP$ as follows. $B$ coincides with the BGP topology in Fig. 8.5a without OVER session $(r, e)$, $I$ is as depicted in Fig. 8.5b, and $o = (r, e)$.

Using the same arguments as in the proof of Lemma 8.1, it can be shown that $B$ is dissemination correct. However, deciding if $B' = (V, E')$, with $E' = E \cup \{o\}$, is dissemination correct is coNP-hard, because of Theorem 8.1. In other words, we cannot exploit the knowledge that an input iBGP network is dissemination correct to efficiently check whether adding an arbitrary OVER session preserves dissemination correctness.

### 8.6 Proposal 2: iBGP Attribute Changing

Another possibility for achieving extended flexibility in iBGP is to conveniently change attributes in BGP messages when they are passed to iBGP neighbors. In the following we refer to the practice of changing attributes in iBGP messages as iBGP attributes changing ($IAC$).

Fig. 8.6a provides a simple example of how operators can exploit this flexibility for implementing policies which are otherwise almost impossible to enforce. 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 the 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
8.6. PROPOSAL 2: IBGP ATTRIBUTE CHANGING

(a) Outbound traffic is routed via AMS-IX only, due to the as-path attribute.

(b) By changing iBGP attributes, AS X can exploit both AMS-IX and PAIX as traffic egress points, achieving better load balancing.

Figure 8.6: A use case in which changing iBGP attribute can be leveraged for traffic engineering purposes.
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 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 Fig. 8.6b. By conditionally changing the value of the local-preference attribute (e.g., via route-maps), this configuration enforces the high-level objective regardless of what as-paths 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 135,000 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 Fig. 8.6a, 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 all traffic destined to prefix $p$ is suddenly shifted towards Palo Alto.

**A Quantitative Study**

Given that changing iBGP attributes provides some advantages to ISPs, 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 1.3) 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.

Fig. 8.7 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, 31st 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 Fig. 8.7), so they are not equally good up through Step 3 of the BGP decision process. Since the 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** lengths 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 (5.73%) 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 it is a common practice to apply policies only to eBGP sessions and then rely on the iBGP topology just to
8.7. MORE FLEXIBILITY IMPLIES MORE INSTABILITY

8.7 More Flexibility Implies More Instability

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.

We now show how to construct an instance $S(X, t, p)$ of SPP 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 a special node (labeled 0) and one node for each iBGP speaker 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 $\epsilon$ and all paths $(u \ldots v 0)$ where $(v, 0)$ is an edge and $(u \ldots v)$ is a valid signaling path (see Section 8.2) 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 [GW02b], where rankings are determined by only relying on IGP metrics, since iBGP attributes are supposed to be the same at every node.

Fig. 8.8a depicts a simple iBGP configuration, while Fig. 8.8b shows the corresponding translation to SPP. We recall that we draw each node $u$ as equipped with a list of paths representing $P^u$, sorted according to $\lambda^u$ (better paths are positioned higher in the list). For example, the list besides node
b1 specifies that b1 can use paths (b1 b2 0) and (b1 0) to reach 0, and prefers (b1 b2 0). The opposite happens at vertex b2. Observe that, by modifying the local-preference attribute, we have been able to create a circular set of preferences which cannot be satisfied at the same time: b1 prefers traversing b2 rather than using the direct route to 0, and vice versa. This kind of policy conflicts can lead to routing oscillations. In fact, the SPP instance in Fig. 8.8b is the oscillation-prone DISAGREE gadget, already presented in Chapter 2.

However, the iBGP topology in Fig. 8.8a 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 ∈ {1, 2}.

In every case, no oscillations can be generated.
8.7. MORE FLEXIBILITY IMPLIES MORE INSTABILITY

The above discussion is an informal proof of the following theorem.

**Theorem 8.2** BGP configurations that allow iBGP attribute changing can generate a larger set of oscillations than BGP configurations where iBGP attributes are not modified.

**iBGP Stability Checker**

The problem of deciding whether a given iBGP configuration and a routing state at time $t$ can lead to routing oscillations is computationally hard (see Chapter 3). However, the algorithm in [FRBS08] shows that, in practice, the complexity can still be manageable. Since this algorithm only works for two-level hierarchies and under the assumption that iBGP attributes are not changed, one might ask whether IAC prevents an operator from using smart techniques to detect routing oscillations in his network. In this section, we show that this is not the case.

We built a prototype tool according to the architecture and the technique described in Chapter 4. The tool can translate iBGP configurations to SPP instances in practice, enabling us to run a stability check on the SPP instance using the Greedy+ algorithm. As a first step in the translation process, our prototype parses BGP configuration files to extract the iBGP peering topology and encodes this topology in a graph $G$ according to the algorithm described above in this section. Now, in order to compute the set $P^u$ of permitted paths at each node $u$ in the graph, we need to know the eBGP routes injected by border routers and to enumerate all valid signaling paths. To do that, we first
CHAPTER 8. TOWARDS AN OPTIMIZED IBGP CONFIGURATION

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 this process, which we call 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 $\lambda^u$ 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 $\lambda^u$. Notice that, to perform Step 6 of the BGP decision process, we need to know the underlying IGP topology.

Fig. 8.9 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 instance $S$ of SPP which is then passed to the Greedy$^+$ algorithm. This algorithm either correctly reports the instance as stable, or pinpoints a set of nodes that might be responsible for routing oscillations (see Chapter 4). Notice that there is a number of ways to obtain the input data from a real network, including, e.g., SNMP [BCC06], screen scraping, etc.

Our prototype tool has a core Java component which performs the Dissemination phase, computes rankings, creates an SPP 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 on both in-vitro and 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. The most time-consuming activity is the Dissemination phase, whose processing time depends on the number of eBGP routes that need to be propagated. Since this number is lower than 20 even for very large networks [FRBS08], we injected 20 eBGP routes for each prefix as a worst-case analysis. Fig. 8.10 shows the
8.8. DESIGN GUIDELINES

In this section, we proposed design guidelines for adding spurious OVERs and changing iBGP attributes without impacting iBGP configuration correctness. Since dissemination correctness has been never studied before, we propose new sufficient conditions, and we base our guidelines on them. Finally, we discuss practical applicability of the guidelines.
Sufficient Conditions for Dissemination Correctness

Either of the following conditions guarantees a signaling correct iBGP topology $B$ to be dissemination correct.

(i) prefer-client: all iBGP routers in $B$ prefer routes propagated by clients (on a UP* path) to any other route.

(ii) no-spurious-OVER: $B$ contains no spurious OVER.

In order to prove our results, we need the following lemma.

**Lemma 8.2** Given a signaling correct iBGP topology $B$, if for any prefix $p$ at least one router in the top layer $T$ selects a route for $p$ that was learned over an UP* path, then $B$ is dissemination correct.

**Proof:** Consider any prefix $p$, and let $\bar{r} \in T$ be the router that selects a route $\bar{R}$ to $p$ which was learned over an UP* valid signaling path $(e \ldots \bar{r})$ (possibly $e = \bar{r}$). By iBGP route propagation rules, $\bar{r}$ propagates route $\bar{R}$ to all routers in $T$. Since $B$ is signaling correct and all routers in $T$ receive at least one route for $p$, all routers in $T$ will eventually select a route. Independent of the neighbor from which the best route was learned, routers in $T$ will propagate their best route to all their clients, which are then guaranteed to receive a route for $p$. These routers, in turn, will announce their own best route to their clients, and so on until routers in the bottom layer are reached. Then, we conclude that every router receives at least one route for prefix $p$, hence $B$ is dissemination correct. □

Now we are able to prove that prefer-client is a sufficient condition for dissemination correctness.

**Theorem 8.3** Given a signaling correct iBGP topology $B$, if $B$ complies with the prefer-client condition, then $B$ is dissemination correct.

**Proof:** We now prove that for any prefix $p$ at least one router in $T$ selects a route to $p$ over an UP* path.

Let $p$ be a prefix and $e_p$ be an egress point for $p$ receiving an eBGP route $R$. Because of step 5 of the BGP decision process, $e_p$ selects $R$. If $e_p \in T$, then the statement trivially holds. Otherwise, there must exist a router $r_1$ such that $r_1 \leftarrow e_p$, by definition of $T$. Because of iBGP dissemination rules, $r_1$ receives at least route $R$ from $e_p$. Let $R'$ (possibly $R' = R$) be the route that $r_1$ selects in the stable state. Since $r_1$ receives route $R$ from a client,
the prefer-client condition implies that route $R'$ is also received from a client. Again, if $r_1 \in T$ the statement holds. Otherwise, iBGP dissemination rules force $r_1$ to propagate route $R'$ to all its route-reflectors. Let $r_2$ be one of the route-reflectors of $r_1$, that is, $r_2 \leftarrow r_1$. Observe that $r_2$ must exist since $r_1 \notin T$. Again, $r_2$ receives at least route $R'$ from its client $r_1$, so we can apply the same argument to $r_2$. We can iterate the argument until we reach a router in $T$ that learns a route from one of its clients. Because of iBGP propagation rules, that route must be learned over an UP* path. Then, the statement follows because of Lemma 8.2.

We now show that no-spurious-OVER guarantees dissemination correctness.

**Theorem 8.4** Let $B$ be a signaling correct iBGP topology with no spurious OVER. $B$ is dissemination correct.

**Proof:** We now prove that for any prefix $p$ at least one router in $T$ selects a route to $p$ over an UP* path.

Let $e_p$ be a router that receives an eBGP route $R$ towards $p$. Because of step 5 of the BGP decision process, $e_p$ selects $R$. If $e_p \in T$, then the statement directly holds. Otherwise, there must exist a router $r_1$ such that $r_1 \leftarrow e_p$. Because of iBGP dissemination rules, $r_1$ receives at least route $R$ from $e_p$. Let $R'$ (possibly, $R' = R$) be the route that $r_1$ selects in the stable state. We have the following cases:

- $r_1 \in T$ and $r_1$ learned $R'$ from one of its clients. By the iBGP propagation rules, $R'$ must be learned over an UP* path.

- $r_1 \in T$ and $r_1$ learned $R'$ from a peer $r_2$. In this case, $r_2$ must receive $R'$ over an UP* path, otherwise it would not propagate it to $r_1$.

- $r_1 \notin T$ and $r_1$ learned $R'$ from one of its clients. Then, $r_1$ forwards route $R'$ to all its route-reflectors.

- $r_1 \notin T$ and $r_1$ learned $R'$ from one of its route-reflectors.

Observe that the no-spurious-OVER condition implies that $r_1$ cannot learn $R'$ from a peer if $r_1 \notin T$.

In the first two cases, the statement holds as a consequence of Lemma 8.2. In the last two cases, there must exist a router $r_2$, with $r_2 \leftarrow r_1$, such that $r_2$ learns a route for prefix $p$. Hence, we can iterate the same argument on $r_2$. 

□
Since the number of layers in $B$ is finite, we eventually find a router in $T$ for which one of the first two cases applies, yielding the statement.

\[ \square \]

**Applicability of the Sufficient Conditions**

We now discuss how the sufficient conditions presented in the previous section can be satisfied (or enforced) in real-world iBGP topologies.

In theory, the `prefer-client` condition can be enforced by carefully designing iBGP topologies. However, we find that this condition is too constraining for real-world topologies. In fact, in order to satisfy the `prefer-client` condition each router should rank the routes it receives according to the first hop in the iBGP signaling path, while the BGP decision process uses tie-breaking criteria that are either based on the last hop in the signaling path (i.e., `egress-id`) or on the length of the path itself (i.e., `cluster-list`). In particular, a direct consequence of condition `prefer-client` is that, if a router $r$ has a valid signaling path $P = (r \ s \ldots e)$ with $r \leftarrow s$ (possibly $s = e$), then any other valid signaling path between $r$ and $e$ must either have a client of $r$ as next-hop or be longer than $P$. Hence, satisfying the `prefer-client` condition requires a deep evaluation of all the decision steps in the iBGP decision process. For this reason, it becomes a really hard task when deploying redundant route-reflectors, even on very simple topologies. Consider, for example, the configuration in Fig. 8.11, which is the simplest redundant route reflection topology designed according to current best practices [ZB03, Smi10]. Underlined paths highlight violations of the `prefer-client` condition. Indeed, clients $e_1$ and $e_2$ are connected to both route-reflectors $r_1$ and $r_2$, and $r_1$ and $r_2$ belong to different clusters. Assume that both $e_1$ and $e_2$ are egress points for a given prefix $p_1$. Even in such a simple scenario, the `prefer-client` condition does not hold, whatever the IGP topology is. In fact, consider router $r_1$ and assume that $e_2$ is its closest egress point according to IGP metrics. In this case, $r_1$ prefers all the routes having $e_2$ as egress point to all the routes having $e_1$ as an egress point, because of step 6 in the BGP decision process. In particular, $r_1$ is forced to prefer routes propagated by its peer $r_2$ via $(r_1 \ r_2 \ e_2)$ to routes propagated by its client $e_1$ via $(r_1 \ e_1)$. Thus, the `prefer-client` condition is violated. This kind of violations of the `prefer-client` condition can be solved by a wiser design of route-reflector clusters. Indeed, if $r_1$ and $r_2$ belong to the same cluster, then $r_1$ always discards routes propagated by $r_2$ and vice versa [BCC06].

**Guideline C** In redundant iBGP configurations, in order to enforce the `prefer-client` condition, redundant route-reflectors must belong to the same cluster.
8.8. DESIGN GUIDELINES

Figure 8.11: Redundant topologies hardly satisfy the *prefer-client* condition.

Observe that current best practices for cluster design [Smi10] do not comply with Guideline C.

The *no-spurious-OVER* condition is relatively easier to enforce, since it only imposes constraints on the iBGP topology and does not require to evaluate the BGP decision process at every router. However, there might be cases in which additional (spurious) sessions are desirable to locally fix forwarding issues or to improve route diversity, as already discussed in this chapter. In such cases, UP sessions can be deployed instead of spurious OVERs, without adversely affecting the dissemination correctness of the iBGP configuration.

**Guideline D** Whenever an additional session is needed to solve visibility issues, an UP session should be deployed, in order to enforce the *no-spurious-OVER* condition.

Observe that using UP sessions is not free from possibly undesired side effects, e.g., shortening the *cluster-list* of existing signaling paths, change the layering of the hierarchy, impact router memory, etc. Some of these side effects can be mitigated, e.g., by configuring route filters that only allow route propagation in one direction.

**Profitable iBGP Attribute Changing**

Previous discussion on *IAC* pros and cons suggests 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., Fig. 8.6b), while ensuring routing stability and keeping the complexity of BGP configuration manageable.
In the following, we propose guidelines that 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 a 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 E** 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 F** 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, \( LP_{peer} > LP_{mod} > LP_{prov} \)

Fig. 8.12 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
8.8. DESIGN GUIDELINES

<table>
<thead>
<tr>
<th>Configuration for Border Routers</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Tag routes according to commercial relationships</td>
</tr>
<tr>
<td>if msg from customer</td>
</tr>
<tr>
<td>add community comm_cust</td>
</tr>
<tr>
<td>if msg from peer</td>
</tr>
<tr>
<td>add community comm_peer</td>
</tr>
<tr>
<td>if msg from provider</td>
</tr>
<tr>
<td>add community comm_prov</td>
</tr>
<tr>
<td>(ii) Prefer customers to peers, and peers to providers</td>
</tr>
<tr>
<td>if comm_cust in community</td>
</tr>
<tr>
<td>set local-preference 200</td>
</tr>
<tr>
<td>if comm_peer in community</td>
</tr>
<tr>
<td>set local-preference 100</td>
</tr>
<tr>
<td>if comm_prov in community</td>
</tr>
<tr>
<td>set local-preference 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration for Route Reflectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Tag routes announced by clients</td>
</tr>
<tr>
<td>del community comm_client</td>
</tr>
<tr>
<td>if msg from client</td>
</tr>
<tr>
<td>add community comm_client</td>
</tr>
<tr>
<td>(ii) Prefer customers to peers, and peers to providers</td>
</tr>
<tr>
<td>Prefer clients to non-clients</td>
</tr>
<tr>
<td>if comm_cust in community</td>
</tr>
<tr>
<td>set local-preference 200</td>
</tr>
<tr>
<td>if comm_cust and comm_client in community</td>
</tr>
<tr>
<td>set local-preference 220</td>
</tr>
<tr>
<td>if comm_peer in community</td>
</tr>
<tr>
<td>set local-preference 100</td>
</tr>
<tr>
<td>if comm.peer and comm_client in community</td>
</tr>
<tr>
<td>set local-preference 120</td>
</tr>
<tr>
<td>if comm_prov in community</td>
</tr>
<tr>
<td>set local-preference 50</td>
</tr>
<tr>
<td>if comm_prov and comm_client in community</td>
</tr>
<tr>
<td>set local-preference 70</td>
</tr>
</tbody>
</table>

Figure 8.12: A simple configuration complying with Guidelines E and F.
very similar technique is commonly used by ISPs to manage traffic from eBGP neighboring ASes [CR05], we argue that our guidelines do not add significant configuration complexity.

We now prove that our guidelines guarantee iBGP stability.

**Lemma 8.3** If the configurations of all iBGP speakers of an AS comply with Guidelines E and F, 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 8.2). Because of the iBGP propagation rules (see Table 1.2), 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, $x$ 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 ensure that $x$ eventually announces $r_1$ to $y$.

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

Moreover, it is easy to show that if every router configuration complies with Guidelines E and F, then the resulting iBGP configuration is free from signaling and dissemination anomalies under arbitrary link failures. In particular, Guideline F enforces the prefer-client condition which we have already shown
8.9. REVISITING THE STATE OF THE ART

BGP configuration languages allow iBGP routers to change iBGP attributes that are relevant to the route selection process. However, both theoretical and practical research contributions neglected this peculiar feature of iBGP. Indeed, it is often assumed that BGP attributes which are relevant to the BGP decision process (e.g., the $\text{local-preference}$ attribute) are not changed as the BGP message is passed to iBGP peers. Such assumption is originally made in [GW02b], where issues related to reduced route visibility due to route-reflection configuration is firstly presented along with a theoretical basis. The assumption often remained in research work focused on sufficient conditions that ensure correctness of a BGP configuration, e.g., [BMU07, BUM08]. Also, state-of-the-art algorithms and tools to detect oscillations in iBGP [FRBS08, FMS+10] assume that iBGP messages are left untouched.

Also, spurious iBGP sessions and the dissemination correctness concept were mostly overlooked in past research work, so extra conditions (like the ones shown in Section 8.8) are needed to keep the validity of the results.

Signaling and forwarding correctness have been introduced and analyzed by Griffin et al. in [GW02b]. The authors show that checking either of the two properties is $\mathcal{NP}$-hard and give sufficient conditions to enforce both of them. While the concept of dissemination correctness is not envisaged in [GW02b], we find that the proposed sufficient conditions also guarantee dissemination correctness, since they encompass the $\text{prefer-client}$ condition as formulated in Section 8.8. However, as discussed in Section 8.8, these conditions are very constraining for real-world networks.

In [RS06], Rawat and Shayman give a set of sufficient conditions that guarantee signaling and forwarding correctness and also prevent MED-induced routing oscillations. In particular, one of the conditions in [RS06] imposes that, for any router, IGP distances to clients must be shorter than IGP distances to non-clients. While this conditions is intended to be a variant of the $\text{prefer-client}$ condition, it is not enough to prevent dissemination anomalies caused by multiple valid signaling paths to the same egress point, as the OVER-RIDE GADGET demonstrates. Moreover, Fig. 8.4b shows an example which matches
the conditions of [RS06] but is not forwarding correct.

In [FR09], Flavel and Roughan propose a modified BGP decision process that evaluates the length of the cluster-list before comparing IGP weights. Such a variant of iBGP is proved to always converge. However, no guarantee is given for dissemination correctness. Actually, the OVER-RIDE GADGET is a simple example where the modified iBGP protocol cannot provide all routers with a route for every prefix.

In [BMU07, BUM08], Buob et al. introduce the concept of fm-optimality, which models the visibility issues that arise when two routers in a valid signaling path disagree on which egress point is the closest one. Fm-optimality is said to guarantee forwarding correctness. Unfortunately, the fm-optimality concept does not account for visibility issues caused by iBGP route propagation rules, e.g., in presence of spurious OVER sessions. In other words, even if all routers on the signaling path agree on which egress point is the closest one, dissemination correctness is not guaranteed. As a counterexample, the OVER-RIDE GADGET is fm-optimal but not dissemination correct.

In [PTOS08, PUT+10], Pelsser et al. propose to add spurious OVER sessions to locally fix visibility issues. Our results show that such a local fix comes at the cost of potential visibility issues on remote routers. Section 8.8 discusses alternatives to spurious OVER sessions that provides similar benefits with no impact on dissemination correctness.

A more general consequence of our work is that the presence of a valid signaling path $P$ between a router $r$ and an egress point $e$ is not sufficient to ensure that $r$ has visibility of routes announced by $e$ (e.g., see Fig. 8.3). In fact, depending on both the IGP and the iBGP topology, there might be some routers in $P$ that do not propagate to $r$ the route announced by $e$. Observe that such a counter-intuitive behavior affects Lemma 3 of [VVKB06], where the presence of an UP*DOWN* path for each pair of routers is said to guarantee full visibility. On the contrary, since only best routes are propagated, the iBGP topology design technique proposed in [VVKB06] guarantees signaling and dissemination correctness, but cannot guarantee forwarding correctness. Also, conclusions drawn in [FB05] are similarly affected. Indeed, configuring a top layer full-mesh (as prescribed by Theorem 4.1 in [FB05]) guarantees a valid signaling path for each pair of iBGP routers, but does not imply dissemination correctness.

All the research contributions analyzed so far neglected the IAC practice, since they assume that the 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. Also, state-of-the-art algorithms
8.10. CONCLUSIONS

and tools to detect oscillations in iBGP [FRBS08] assume that iBGP messages are left untouched. To the best of our knowledge, the research work presented in this chapter is the first study on pros and cons of IAC, and the prototype tool we built (see Section 8.7) is the first automated convergence checker able to deal with IAC practice.

Also, despite the concept of dissemination correctness had not been formalized so far, we find that some results in the literature guarantee it as a side effect. Modifications to the iBGP protocol as proposed in [MC04] and fine tuning of attributes of iBGP messages as we proposed in Section 8.8 can be leveraged to enforced the prefer-client condition. In both cases, however, the likelihood of incurring suboptimal routing increases, since client routes are preferred, no matter what are the IGP distances of the corresponding egress points.

Recently, BGP Add-Paths [WRCS11] has been proposed to allow routers to propagate multiple routes. It is important to note that the advertisement of multiple routes guarantees dissemination and forwarding correctness only if all the routes that are equally preferred up to and including the first four steps of the BGP decision process (so called AS dominant routes) are propagated network-wide. However, the higher number of routes handled in iBGP could cause router memory and update churn penalties [VFB10]. Raszuk et al. [RFP+11] propose to add special route-reflectors in order to distribute multiple routes. Unfortunately, since this technique relies on additional route-reflectors to propagate multiple routes, it does not guarantee the advertisement of all the AS dominant routes, and thus it is not sufficient for dissemination correctness. Moreover, some encapsulation mechanism is currently suggested for solving forwarding anomalies in case of propagation of a subset of the AS dominant routes [UvF+10]. Finally, observe that both proposals are still in the development stage.

8.10 Conclusions

iBGP route reflection trades full route visibility at all iBGP routers for better scalability. In turn, limited route visibility poses the basis for several correctness (i.e., routing and forwarding anomalies [GW02b]) and performance (e.g., poor route diversity and slower convergence) problems.

In this chapter, we studied proposals to overcome limitations of iBGP by accurately designing its configuration. Starting from simple use cases, we focused on two design proposals, namely addition of spurious iBGP sessions and
changing of iBGP attributes. We showed that the extended flexibility achieved by those design proposals generally exacerbates iBGP correctness problems, potentially creating additional routing and forwarding issues. Also, they can affect propagation of routes in the iBGP topology. In order to model anomalies due to iBGP propagation rules, we introduced the new concept of dissemination correctness. Finally, we proposed some guidelines for adding spurious sessions and changing iBGP attributes with correctness guarantees.

In our opinion, this study shows that iBGP semantics are actually more complex than what is commonly assumed, and provides new motivation to recent efforts (e.g., [WRCS11, OMU\textsuperscript{+11}]) for decoupling route propagation from route selection in iBGP. Our findings also suggest that iBGP topology design need extreme care, as even a single iBGP session or a wrong attribute setting can create unexpected and counter-intuitive side effects.
Chapter 9

Seamless BGP Reconfigurations

9.1 Introduction

As discussed in Chapter 8, network evolution requires BGP configuration to be periodically modified. During the life of a network, both iBGP and eBGP configurations evolve. New iBGP routers are introduced while older ones are either decommissioned or moved to less data- or control-plane traffic intensive areas. As the network grows, iBGP sessions may also need to be added or removed. For example, as the number of iBGP routers increase, the full-mesh of iBGP sessions dictated by the original BGP specification [RLH06] has to be replaced by a route reflection [BCC06] configuration that keeps the number of iBGP sessions manageable. Also, iBGP configuration changes may be triggered by changes to the underlying IGP. IGP changes are often performed in ISPs, e.g., to optimize the usage of network resources by fine-tuning of IGP weights [FT02] (see also Chapter 7). Unfortunately, IGP configuration adjustments can affect iBGP routing choices, possibly leading to routing and forwarding inconsistencies (see [GW02b, BMU07] and Chapter 8), as well as undesired side effects on internal and external traffic flows [BL08, CEDFQ06]. Similarly, the eBGP configuration may need to be changed from time to time. A typical use case is the provisioning of a new customer, which requires to establish new eBGP sessions on some border routers. As commercial relationships between ISPs change, operators also need to modify their eBGP routing policies. Recent examples include the so-called “peering wars” that led to the depeering of large ISPs [BHP09, BZP08].

As for IGP migrations (see Chapter 7), stringent SLAs force the reconfig-
CHAPTER 9. SEAMLESS BGP RECONFIGURATIONS

...uration process to be performed on running networks and to not disrupt user traffic and services provided by the reconfiguring ISP. However, due to the nature of BGP itself, the impact of changes to either iBGP or eBGP configuration is hard to predict, and techniques applied to IGP migrations cannot be straightforwardly applied to BGP reconfigurations. The main reason is that local changes on one BGP router can affect routing information as viewed by remote routers in different ways depending on the routing choices of intermediate routers. Hence, signaling, dissemination, and forwarding anomalies can be introduced in intermediate configurations. Moreover, outgoing traffic to single prefixes can be shifted many times from one egress point to others, invalidating load-balancing policies and generating iBGP and eBGP churn. High eBGP churn can in turn increase the likelihood of being dampened.

Despite importance and complexity of BGP reconfigurations, network administrators lack methodologies and tools to perform reconfiguration tasks with minimal impact on the traffic. Only a few best practices are available (e.g., [Smi10, Hv10]), but they typically focus on simple reconfiguration cases. Even worse, current best practices barely take into account the possibility of creating routing and forwarding anomalies during the migration process.

In this chapter, we consider the problem of changing the BGP configuration seamlessly, that is, with theoretical guarantees of experiencing no packet loss. In Section 9.2, we introduce the model we use in this chapter, and we formally state the BGP seamless reconfiguration problem. In order to model iBGP topologies and mimic the iBGP decision process, we propose a variation of the SPP model, which we call i-SPP. With respect to SPP, additional constraints on paths allowed at each node and ranking of those paths are added in i-SPP. We formalize the concept of BGP configuration, so that eBGP policies are also taken into account.

In Section 9.3, we review simple approaches and current best practices. We show that simple approaches cannot avoid, both theoretically and practically, long-lasting routing oscillations, dissemination anomalies, forwarding loops, and unintended traffic shifts during BGP reconfigurations even when the initial and the final BGP configurations are anomaly-free. To quantify such anomalies, we simulated BGP reconfigurations in a Tier-1 network observing a significant number of anomalies which persist for large parts of the reconfiguration process.

In Section 9.4, we study the BGP reconfiguration problem from an algorithmic perspective. In particular, we aim at finding a seamless operational ordering in which to perform BGP configuration changes that ensure intermediate configurations to be anomaly-free. We show that there are cases in which routing or forwarding anomalies cannot be avoided in every operational...
ordering. Even worse, the problem of deciding if a seamless operational ordering exists is computationally hard, as we prove in Section 9.5. The problem remains hard (in both eBGP and iBGP reconfigurations) even if no eBGP dynamicity is assumed, and only one prefix is considered.

Given the impossibility to algorithmically solve the reconfiguration problem, we argue that additional ad-hoc configuration is needed. In Section 9.6, we propose a framework to achieve seamless migrations in any BGP reconfiguration scenario. Our proposal is based on running two iBGP routing processes at the same time, with an approach similar of the one adopted in Chapter 7. We explain what are the limitations of current technologies and how to overcome them. We describe a prototypical implementation of our approach. Further, in Section 9.7, we show the effectiveness of our framework through a case study.

Finally, we review related work in Section 9.8, and we conclude in Section 9.9

### 9.2 Seamless BGP Reconfigurations

In this section, we formally state the seamless BGP reconfiguration problem. Intuitively, it consists in finding a way to progressively replace the BGP configuration of a network with another without impacting data-plane traffic.

It is not uncommon for network operators to face such reconfiguration problems. To illustrate the frequency of BGP configuration changes, we analyzed BGP configurations of approximately 20% of the routers of a Tier-1 ISP, from April 2010 to July 2011. Considered routers were new generation routers progressively added to the network during the considered timeframe. Among those routers, some have been replaced after their introduction by other routers of a different brand: this happened 17 times in the 15 month period. We computed 1,337 BGP configuration changes in total. Addition and removal of BGP sessions were the most common BGP changes. Note that such operations correspond to iBGP topological changes and eBGP neighbor modifications. Overall, the specification of a new BGP session was added to the configuration of a router 5,828 times, encompassing 976 additions of eBGP sessions and 4,852 additions of iBGP sessions. Session removals were less frequent but still not rare, as they happened 236 for eBGP sessions and 1,440 for iBGP sessions. At each router, eBGP sessions were typically added in groups, while iBGP sessions were mostly added in pairs of redundant client sessions with two route-reflectors. However, iBGP sessions were added on several routers in short time periods, during which we often registered the addition or, even more
frequently, the replacement of a router. Beyond session addition and removal, we accounted for 41 changes of inbound eBGP policy and 77 modifications of outbound eBGP policy, by only looking at route-map names. A cluster ID was added to a router configuration, indicating that the router has become a route-reflector, 11 times in total. Since the type of the sessions (eBGP, iBGP peer/client) never changed, this also means that clients were added to the router, and a layer is added at the bottom of the route reflection hierarchy. Finally, we collected less frequent miscellaneous changes, encompassing AS number modification on an eBGP peer (8 times), and address families enabling (3 times) and disabling (5 times) on eBGP sessions. These results testify that reconfigurations of already established BGP sessions are also performed by operators commonly, even if less frequently than addition and removal of BGP sessions.

In the following, we first present the theoretical model we adopt in this chapter. Then, we formally state the seamless BGP reconfiguration problem.

A Model for BGP Reconfigurations

In this section, we define the model and the notation we use in this chapter. Most of the concepts we will refer to are already introduced in Chapter 8. However, we now tailor the SPP formalism to model the outcome of the BGP decision process at each iBGP router. We call such extension i-SPP model.

Intuitively, an i-SPP instance is a special SPP instance, in which a path is permitted if and only if it is valid signaling path and path preferences reflect the underlying IGP configuration. More formally, an i-SPP instance is a tuple $S = (B, U, P, \Lambda)$, where $B$ accounts for the iBGP topology, $U$ accounts for the IGP topology, and $P$ and $\Lambda$ provide information on route preferences at each iBGP router.

Graph $B = (V, E)$ is a directed labeled graph where nodes in $V$ represent routers and edges in $E$ represent iBGP sessions. Each edge $(u, v)$ is associated with a label which is either UP or OVER, according to the type of session that the edge represents. Because of the way iBGP relationships are defined, for each edge $(u, v)$ labeled as OVER, an edge $(v, u)$ labeled as OVER must be in $E$. Also, edges labeled as UP are directed from the client to the route reflector.

The underlying IGP topology is modeled as an undirect weighted graph $U$, with a weight associated to each edge $(u, v) \in U$. We denote with $\text{dist}(u, v)$ the total weight of the shortest path in $U$ from $u$ to $v$.

Signaling paths (i.e., paths in $B$) that can be used by iBGP routers to reach destination prefixes are modeled by $P$, while $\Lambda$ accounts for path preferences
9.2. SEAMLESS BGP RECONFIGURATIONS

(according to Table 1.3). In particular, \( \mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u \) is the set of permitted paths, and \( \forall u \in V \mathcal{P}^u \) represent the set of paths that \( u \) can use to reach external destinations. In the following, we use path for indicating signaling paths and not forwarding paths, unless differently specified. Also, whenever it is clear from the context, we use a signaling path to refer to the route advertised over that path (e.g., we say that a router receives a path, or prefers a path over another). A path \( P \) on \( B \) is a permitted path (i.e., \( P \in \mathcal{P}^u \)) if and only if \( P \) is a valid signaling path terminating with an egress point. The empty path \( \epsilon \) represents destination unreachability, and is always permitted at any vertex in \( V \).

For each \( u \in B \), the preference level of paths in \( \mathcal{P}^u \) is expressed by a ranking function \( \lambda^u : \mathcal{P}^u \rightarrow \mathbb{N} \). If \( P_1, P_2 \in \mathcal{P}^u \) and \( \lambda^u(P_1) < \lambda^u(P_2) \), then \( P_1 \) is preferred over \( P_2 \). We define \( \Lambda = \{ \lambda^u | u \in V \} \). The following conditions hold on \( \mathcal{P}^u \) of each vertex \( u \in B \):

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

(ii) \( \forall P_1, P_2 \in \mathcal{P}^u, P_1 \neq P_2 \Rightarrow \lambda^u(P_1) \neq \lambda^u(P_2) \) (strict ranking of paths is assumed)

(iii) Let \( e_1, e_2 \) be two egress points, with \( \text{dist}(u, e_1) < \text{dist}(u, e_2) \). Then, \( \forall P_1, P_2 \in \mathcal{P}^u, P_1 = (\ldots e_1) \land P_2 = (\ldots e_2) \Rightarrow \lambda^u(P_1) < \lambda^u(P_2) \) (preferences reflect the IGP topology)

(iv) \( \forall P_1, P_2 \in \mathcal{P}^u, P_1 = (\ldots e), P_2 = (\ldots e), \text{ then } |P_1| < |P_2| \Rightarrow \lambda^u(P_1) < \lambda^u(P_2) \) (preferences privilege shorter iBGP paths)

Concepts like activation sequence, stable path assignment, and dispute wheel, are analogous to SPP.

Observe that \( \mathcal{P} \) and \( \Lambda \) are defined as all the possible egress points in the network injected a route towards any destination. This is not true in general, as routes injected in iBGP depends on the availability of eBGP routes at different egress points and eBGP routing policies. We model the fact that only a subset of egress points inject routes in iBGP for given prefixes by conveniently filtering the set of permitted paths in the i-SPP instance. More formally, we call egress set of a prefix \( p \) the set of egress points receiving routes to \( p \) equally preferred according to the eBGP policies, i.e., according to the first four steps of the BGP decision process (see Table 1.3). Also, given an i-SPP instance \( I = (B, U, \mathcal{P}, \Lambda) \) and an egress set \( S \), we define the corresponding egress instance

\( ^1 \text{this models determinism of the BGP decision process} \)
as \( I[S] = (B,U,P,I) \), with \( P_I \subseteq P \), \( P = \{ \ldots e \} \in P_I \Leftrightarrow e \in S \) and \( \forall u \in V, \forall P \in P^u_I, \lambda^u_I(P) = \lambda^u(P) \).

An iBGP configuration is a triple \( (I,S,Y) \), where \( I \) is an i-SPP instance, \( S \) is the set of egress sets, and \( Y \) is a function that associates each destination prefix \( p \) to its egress set \( S \in S \). \( S \) and \( Y \) take into account the eBGP policies. We say that a configuration \( C \) is oscillation-free if \( \forall S \in S I[S] \) is guaranteed to converge to a stable state (i.e., is safe). Similarly, \( C \) is dissemination correct, deflection-free, and loop-free if \( \forall S \in S I[S] \) is dissemination correct, not subject to deflections, and free from forwarding loops, respectively.

Since this chapter focuses on BGP reconfigurations, we deal with iBGP configurations that change over time. Whenever it is not clear from the context, we will add an index \( t \) to the notation to refer specifically to step \( t \) of the migration. For example, \( C_t = (B_t,U_t,S_t,Y_t) \) is the iBGP configuration at step \( t \) of the reconfiguration. We define two special indexes \( i \) and \( f \) that refer to the initial and the final iBGP topologies, respectively.

**Problem Statement**

We define a migration, or reconfiguration, as a sequence of configuration changes that turn an initial BGP configuration \( C_i \) into a final one \( C_f \). We assume \( C_i \) and \( C_f \) to be given as input and to be anomaly-free. The underlying IGP is supposed not to change during the reconfiguration.

Stringent Service Level Agreements deny the possibility to simply shut down the network, and restart it with the new configuration. Also, simultaneously overwriting configuration files on all the routers is unpractical, as it is likely to generate huge control-plane churn, which, in turn, can overwhelm routers. Moreover, the latter approach does not allow operators to keep the reconfiguration process under control, turning misconfigurations or human errors (e.g., typos) into a management nightmare.

Hence, an incremental approach is needed. In order to speed up reconfigurations and be effective from a management point of view, we also aim at lowering as much as possible the number of times the configuration of the same router is modified, e.g., by establishing and shutting down groups of sessions in a single step. Hence, we do not consider migrations in which router configurations can be modified for one prefix at the time. Indeed, acting on a per-prefix basis is not practical, given that the size of todays BGP RIBs (more than 350,000 prefixes) would force very slow and long migration processes. Consider the following rough estimation. If we wait for protocol convergence after each iBGP topological change is applied, a per-prefix migration will take
9.2. SEAMLESS BGP RECONFIGURATIONS

\(O((t_{comm} + t_{conv}) \ast s \ast p)\), where \(t_{comm}\) and \(t_{conv}\) are time taken for configuration commit and protocol convergence respectively, \(s\) is the number of affected sessions, and \(p\) is the number of prefixes in the RIB. Assuming that the order of magnitude of the commit time is 1 millisecond (it could be an unfair underestimation in case of big configuration files) and that the convergence time is proportional to the number \(n\) of routers in the network, we have a lower bound of \(O(n \ast s \ast p)\) milliseconds, i.e., almost 27 hours per session to modify on a network of 100 routers.

Intuitively, we define a migration as seamless if it guarantees no routing, dissemination, and forwarding anomalies in intermediate configurations. Also, a seamless migration is free from unintended traffic shifts. More formally, let \(C_j = (I_j, S_j, \Upsilon_j)\) be the configuration at step \(j\) of the migration, with \(i \leq j \leq f\), and assume that \(C_j\) is oscillation-free, LoV-free, and deflection-free. Also, let \(\pi^u_j(p)\) be the path steadily selected by router \(u\) in \(I_j[\Upsilon_j(p)]\). We say that \(C_j\) is not subject to unintended traffic shifts if, for each prefix \(p\) and for each router \(u\),

- the egress point used by \(u\) to \(p\) is consistent with previous configurations, that is, \(\forall k: i \leq k \leq j\), then \(\pi^u_j(p)\) and \(\pi^u_k(p)\) end with the same egress point; or

- the egress point used by \(u\) to \(p\) is consistent with the following configurations, that is, \(\forall k: j \leq k \leq f\), then \(\pi^u_j(p)\) and \(\pi^u_k(p)\) end with the same egress point.

We formally define a migration as seamless if for any migration step \(j\), with \(i \leq j \leq f\):

- \(C_j\) is oscillation-free;
- \(C_j\) is LoV-free;
- \(C_j\) is deflection-free;
- \(C_j\) is not subject to unintended traffic shifts.

Conversely, routing and forwarding anomalies occur in intermediate configurations during migrations which are not seamless. These anomalies persist until another intermediate configuration is reached, which might require several additional migration steps. We refer to such persistent anomalies as migration anomalies. Migration anomalies can cause long-lasting disruptive effects, among which forwarding deflections and loops, unintended traffic shifts, traffic
blackholes, congestions, unnecessary iBGP churn, and unnecessary eBGP updates which increase the risk of route dampening [VCG98]. On the contrary, we do not consider short-lived protocol-dependent issues, like those occurring transiently during protocol convergence, as they are unrelated to BGP reconfigurations.

### 9.3 Current Solutions do not Work

In this section, we show that naive incremental approaches and current best practices [Smi10, Hv10] risk, both theoretically and practically, to introduce several migration anomalies. We propose and evaluate other approaches in Sections 9.4 and 9.6.

#### Current Best Practices Provide No Guarantees

Operators can currently rely on a few rules of thumb that apply to simple topological changes. For example, regarding iBGP topological changes, current best practice mainly apply to the replacement of a fully-meshed iBGP topology with a two-layer route reflection hierarchy [Smi10, Hv10]. In that case, operators are suggested to configure one cluster at the time, by establishing all UP sessions ending in a route-reflector before removing the unnecessary OVERs. This approach is difficult to generalize to other iBGP topological changes, like modifications of a pre-existing route reflection hierarchy. Current best practices do not explicitly prescribe an ordering in which to reconfigure routers. We argue that simple operational ordering are followed in practice, e.g., bottom-up or top-down approaches. According to private discussions with operators, it seems that a bottom-up approach is typically followed, hence routers are reconfigured on a per-layer basis, starting from the bottom layer up to the top one. In the following, we assume that when a router is reconfigured the final configuration is directly installed on it. Our considerations, however, can be easily extended to approaches in which routers are only partially reconfigured, e.g., by removing sessions only when both ends are already migrated. We now show that, from a theoretical point of view, migration anomalies cannot be avoided in the general case by only imposing a bottom-up reconfiguration ordering. We describe cases in which there exists an operational ordering which prevents migration anomalies raised by a bottom-up reconfiguration strategy. Other simple operational orderings, like a top-down approach, has similar problems.

An example of migration oscillation created by bottom-up reconfigurations is reported in Fig. 9.1. The graphical convention we adopt in the figure is
9.3. CURRENT SOLUTIONS DO NOT WORK

Figure 9.1: An example in which the bottom-up strategy, suggested by the current best practices, creates routing oscillations during the reconfiguration.

the same we will use throughout the chapter for iBGP topologies. Circles represent routers having no clients, while diamonds represent route-reflectors. UP sessions are drawn as lines terminating with an arrow on the side of the route-reflector, while OVER sessions are represented by lines with an arrow on both sides. Short dashed arrows entering a router $r$ and labeled with a prefix $p$ represent the fact that $r$ is an egress point for prefix $p$. Ranking of permitted paths at each router is conveyed by a list of paths, ordered from the most preferred to the least preferred, and drawn aside the router. Path $\epsilon$ is omitted for brevity. Whenever it is clear from the context, we will replace the list of path preferences with a list of egress point preferences, in which each egress point represents all the paths terminating on that egress point. Also, in the list besides any router $u$, some egress points can be omitted if paths from them are guaranteed not to be selected from $u$. In particular, less preferred egress points are omitted from $u$'s list if a more preferred egress point $e$ exists from which $u$ is guaranteed to receive a path, e.g., if $e$ is a direct client or a direct route-reflector of $u$. We will omit the IGP topology whenever it is easy to build starting from egress point preferences.

In the example of Fig. 9.1, a bottom-up approach requires to reconfigure routers at the bottom layer, i.e., $e_1$, $e_2$, and $e_3$, before all the others. However, after reconfiguration of $e_2$, session $(e_2, r_3)$ is removed, and the resulting intermediate configuration becomes subject to routing oscillations [GW02b]. The
problem is fixed only when one of the routers belonging to the middle layer in $B_f$ are migrated. Observe that a seamless migration can be achieved in this case by reconfiguring $r_3$ before any other router in the network.

Similar examples can be found for other kinds of migration anomalies, like forwarding loops and traffic shifts. Indeed, consider the example in Fig. 9.2, which can look like the introduction of route reflection into a Point of Presence of an ISP. Egress point preferences reflect the IGP topology depicted in the figure. We assume that $e_2$ has a lower egress-id with respect to $e_1$, which justifies $r_1$’s routes ranking. Observe that conflicting egress point preferences at $l_1$ and $r_1$ cannot result in routing oscillations in this example. Indeed, either $l_1$ is in full-mesh or $r_1$ is the only iBGP neighbor of $l_1$. In the former case $l_1$ learns its best route via an OVER direct path, in the latter case $l_1$ must agree with $r_1$’s choice as it does not receive routes from any other iBGP neighbor. A similar argument can be applied to the egress point preference conflict between $l_2$ and $rr_1$. Also, the initial and the final configurations are deflection-free, as

Figure 9.2: An example in which the bottom-up strategy, suggested by the current best practices, creates a loop during the reconfiguration.
9.3. CURRENT SOLUTIONS DO NOT WORK

the absence of forwarding anomalies is guaranteed in a full-mesh (i.e., in $B_i$), and the fact that all routers but $e_1$ choose the route from $e_2$ in $B_f$. Unfortunately, if a bottom-up approach is followed, a forwarding loops between $l_1$ and $l_2$ will occur immediately after the reconfiguration of $l_1$, as $l_1$ will have no visibility of routes from $e_1$ while $l_2$ still receive them via its OVER session. This loop is interrupted only after reconfiguration of $l_2$ or $e_2$.

Fig. 9.3 depicts an example in which an unintended traffic shift at $c_1$ is created during a bottom-up reconfiguration. Indeed, $c_1$ has only one route-reflector in both $B_i$, hence it is forced to steadily select the route from $e_1$ in the initial configuration. Moreover, the bottom-up approach prescribes to reconfigure $c_1$ before any router in the final middle layer. Consider the configuration $C_j$ generated immediately after the reconfiguration of $c_1$. In $C_j$, $c_1$ selects the route announced by $e_2$ which is currently steadily selected by $r_3$ as it learns that route from $r_2$. After the reconfiguration of $r_3$ or $r_2$, however, $r_3$
will switch to $e_3$, forcing $c_1$ to change again the egress point it uses for traffic to $p_1$. Observe that reconfiguring $r_3$ before $c_1$ would cause no unintended traffic shift in this example.

**Quantitative Analysis**

To quantify issues that occur when simple migration approaches are used, we performed several experiments on a Tier-1 network consisting of roughly 100 iBGP routers organized in three layers of route reflection. We performed two kinds of experiments: iBGP topology changes and eBGP policy modifications.

The first kind of experiments consisted in reconfiguring an iBGP full-mesh into the given route reflection hierarchy. We simulated per-router migrations performed according to the current best practices: at each migration step, we picked one router and we activated all its UP sessions before shutting down the OVER sessions not belonging to the final configuration. In particular, we evaluated completely random router migration orderings (e.g., the one generated by a script which simply iterates on all the routers), as well as random orderings in which top layer routers are reconfigured at the end, and bottom-up orderings. We denote these strategies as $RND$ (Random), $RBT$ (Random But Top), and $BTU$ (Bottom-Up), respectively. For each strategy, we run 50 different experiments, each experiment corresponding to a different ordering in which routers are migrated. For each experiment, we used C-BGP [QU05] to compute all the BGP routing tables in the intermediate configurations.

Fig. 9.4 plots the fraction of experiments during which different types of anomalies occur when simple per-router migrations are applied. A data point $(x, y)$ in the graph means that $(100 \times y)\%$ of the experiments for that particular strategy exhibited a given anomaly for at least $x\%$ of the migration steps. $RND$ orderings almost always triggered Loss of prefix Visibility (LoV) at some iBGP router, for some prefixes. This makes random orderings clearly not viable in practice. $RBT$ migrations were not subject to LoVs, however several migration issues were raised. Indeed, in more than 90% of the experiments, loops occurred during more than the 10% of the migration steps. Even worse, traffic shifts occurred during more than 55% of the $RBT$ migration process in almost all the experiments. Surprisingly, $BTU$ migrations were not subject to LoVs, however several migration issues were raised. Indeed, in more than 90% of the experiments, loops occurred during more than the 35% of the migration steps. On the other hand, $BTU$ performs better than $RBT$ regarding traffic shifts. Nevertheless, traffic shifts occurred during more than 40% of almost all the $BTU$ migrations.
9.3. CURRENT SOLUTIONS DO NOT WORK

In the second kind of experiments, we analyzed changes of eBGP policies, and we measured the amount of unintended traffic shifts created by those changes. In each of those experiments, we modified the value of the local-preference assigned to the routes received by a given neighboring ISP. Our data set consisted of the C-BGP [QU05] model of the Tier-1 along with a dump of all the Adj-RIB-In from the main route reflectors. In order to focus on significant traffic shifts, we restricted our analysis to the 940 prefixes that together were responsible for 80% of the traffic [UT06]. Given this input, we identified all the neighboring ISPs announcing at least one of the 940 prefixes. We further filter the list of ISPs by excluding those having only one eBGP peering with the Tier-1. After this process, we end up with 50 ISPs and 250 eBGP sessions. For simplicity, we assumed the Tier-1 to initially apply the same local-preference on all the eBGP peerings it keeps with the same neighboring ISPs. We used C-BGP to compute the BGP routing tables of every router in the network before, during, and after the local-preference configuration change. By comparing routing tables at different steps, we finally computed the unintended traffic shifts caused by different reconfiguration processes. In each experiment, we considered one among the 50 ISPs previously selected.
For each ISP, we repeated traffic shift measurements for 5 different orderings in which border routers are reconfigured. Also, we considered different final values of local-preference (i.e., in $C_f$), which we denote as $LP_f$. Namely, in different experiments, we set $LP_f$ to the minimum, maximum, and an intermediate (median) value among those found in the initial configuration. These scenarios correspond to turn a neighboring ISP into a provider, peer, customer, respectively.

Fig. 9.5 shows the complementary cumulative distribution of the average number of unintended traffic shifts per router. Each point in the plot corresponds to an experiment involving a different neighboring ISP, a different value of $LP_f$, and a different ordering. On average, 50% (20%, resp.) of the routers experience at least 1 (1.5, resp.) unintended traffic shifts for each prefix announced by the ISP considered in the experiment when $LP_f$ is set to the median or maximum value. In some experiments, we recorded more than 2 and 2.5 unintended traffic shifts on average per router per prefix when $LP_f$ is set to the maximum and the median value, respectively. This means that each router in the network can change multiple times its egress point to each interdomain destination, potentially violating load-balancing and traffic engineering policies (e.g., forwarding packets to other continents over high-cost transoceanic cables).
and unleashing traffic congestion for many migration steps. Additionally, eBGP churn can increase the likelihood of route damping. We expect these results to scare network operators, especially if they have to change eBGP policies applied to ISPs announcing the few prefixes that drive the vast majority of the Internet traffic [LIJM+10]. Lowering the local-preference to the minimum value creates less traffic shifts on average than setting $LP_f$ to the maximum or the median value. In fact, contrary to the other two scenarios, routes affected by setting $LP_f$ to the minimum value never attract additional traffic, and can only be de-selected by routers that prefer them before. Still, in few experiments, the average number of unintended traffic shifts is more than 2.5 per router per prefix.

### 9.4 An Algorithmic Approach is not Viable

In this section, we present examples in which the BGP seamless reconfiguration problem cannot be solved by only adding sessions in $C_f$ and removing sessions in $C_i$. We first tackle iBGP topology changes, then we address the problem of seamlessly changing eBGP policies.

**iBGP Topology Changes**

From an algorithmic point of view, the problem of changing the iBGP topology can be formalized as follows. We refer to an ordering in which to reconfigure iBGP sessions achieving a seamless migration as *seamless ordering*.

**Session Ordering Computation Problem (SOCP):** given $B_i$ and $B_f$, compute a seamless ordering in which to add sessions in $B_f \setminus B_i$ and to remove sessions in $B_i \setminus B_f$.

Observe that, in SOCP, eBGP routes are assumed not to change throughout the reconfiguration process. Indeed, given an initial configuration $C_i = (B_i, U, \mathcal{S}, \mathcal{Y})$, $\mathcal{S}$ and $\mathcal{Y}$ are supposed to remain the same at any migration step. In the following, we show that even if eBGP is totally static, there are cases in which a seamless ordering does not exist. In the following, we refer to sessions to be added or removed during the migration, that is, sessions in $B_i \Delta B_f$, as *affected sessions*. Also, when non ambiguous, we use symbol $<$ to denote relative ordering between two operations such as the addition or removal of an affected session. For example, $o_1 < o_2$ means that operation $o_1$ is performed before $o_2$, i.e., $o_1$ and $o_2$ are respectively performed at migration steps $i$ and $j$, with $i < j$. We use $\leq, =, \geq, >$ with similarly semantics.
We consider per-session and per-router migrations. To be as flexible as possible, we allow multiple sessions involving the same router to be simultaneously added or removed, at any migration step. Such an approach closely reflects the degree of freedom that operators have. Indeed, multiple sessions involving the same router \( r \) can be simultaneously reconfigured by changing the configuration of \( r \). On the contrary, assuming synchronism between changes on arbitrary sessions is less realistic, since perfect synchronism between configuration commits and BGP updates processing at multiple routers is hard to achieve. Moreover, allowing simultaneous operations involving different routers overcomplicates piloting the reconfiguration, e.g., in case of a single commit that fails.

In the following, we show that there are cases in which a seamless ordering does not exist. Even worse, we show examples in which i) any reconfiguration ordering is not oscillation-free; ii) any reconfiguration ordering is not dissemination correct; iii) any reconfiguration ordering is not deflection-free; iv) any reconfiguration ordering is subject to unintended traffic shifts. It is simple to extend those examples to show cases in which no reconfiguration ordering is free from different kinds of anomalies, e.g., some orderings creates migration oscillations while others forwarding loops.

The graphical convention we adopt for iBGP topologies is the following (e.g., see Fig. 9.6). Circles represent routers having no clients, while diamonds represent route-reflectors. UP sessions are drawn as lines terminating with an arrow on the side of the route-reflector, while OVER sessions are represented by lines with an arrow on both sides. Short dashed arrows entering a router \( r \) and labeled with a prefix \( p \) represent the fact that \( r \) is an egress point for prefix \( p \). Ranking of permitted paths at each router is conveyed by a list of paths, ordered from the most preferred to the least, and depicted aside the router. Whenever it is clear from the context, we will replace the list of path preferences with a list of egress point preferences, in which egress point represents all the paths terminating on that egress point. Also, in the list beside any router \( u \), some egress points can be omitted if \( u \) is guaranteed to never select routes announced by them. In particular, less preferred egress points are omitted from \( u \)'s list if a more preferred egress point \( \bar{e} \) exists from which \( u \) is guaranteed to receive a path, e.g., if \( u \) is a direct client or a direct route-reflector of \( \bar{e} \). Finally, we will omit underlying IGP topologies when they are easy to build starting from egress point preferences.
9.4. AN ALGORITHMIC APPROACH IS NOT Viable

Unavoidable Routing Oscillations

Fig. 9.6 depicts an example in which any reconfiguration ordering creates a permanent oscillation at some reconfiguration step, even if the initial and the final configurations are oscillation-free. Indeed, IGP metrics dictate preferences on routers such that a Bad-Gadget $\Pi'$ (see Chapter 2) between routers $r_1$, $r_2$, and $r_3$ for prefix $p_1$ cannot be prevented from oscillating in absence of $(e_1, r_3)$ and $(e_x, r_2)$. Spoke paths in $\Pi'$ are $\vec{Q}' = ((r_1 e_1) (r_2 e_2) (r_3 e_3))$, and rim paths are $\vec{R}' = ((r_1 r_2) (r_2 r_3) (r_3 r_1))$. Moreover, another Bad-Gadget $\Pi''$ exists between $r_2$, $r_3$, and $r_4$ for prefix $p_2$ in presence of both $(e_1, r_3)$ and $(e_x, r_2)$. Pivot vertices in $\Pi''$ are $\vec{U}'' = (r_2 r_3 r_4)$, spoke paths are $\vec{Q}'' = ((r_2 e_x) (r_3 e_1) (r_4 e_4))$, and rim paths are $\vec{R}'' = ((r_2 r_3) (r_3 r_4) (r_4 r_2))$.

$B_i$ and $B_f$ are oscillation-free. In $B_i$, $\Pi'$ is prevented from oscillating because of the presence of session $(e_x, r_2)$, and $\Pi''$ does not exist because of the absence of session $(e_1, r_3)$ (i.e., one spoke path is missing in the dispute wheel). Similarly, in $B_f$, $\Pi'$ is prevented from oscillating because of the presence of session $(e_1, r_3)$, and $\Pi''$ does not exist because of the absence of session $(e_x, r_2)$ (i.e., one spoke path is missing in the dispute wheel). However, $B_i$ and $B_f$ are not signaling correct, i.e., correct for any combination of egress points [GW02b].
Consider now any possible migration. Since sessions to add and remove have no router in common, we have only two cases.

- \( \text{add}(e_1, r_3) < \text{remove}(e_x, r_2) \). In this case, immediately after \( \text{add}(e_1, r_3) \), nothing prevents \( \Pi'' \) from permanently oscillating. Such an oscillation is interrupted only after that \( (e_x, r_2) \) is removed.

- \( \text{remove}(e_x, r_2) < \text{add}(e_1, r_3) \). Immediately after \( \text{remove}(e_x, r_2) \), nothing prevents \( \Pi' \) from permanently oscillating. Such an oscillation is interrupted only after that \( (e_1, r_3) \) is added.

In both cases, an intermediate configuration exists which is subject to permanent oscillations.

We experimentally confirmed that no oscillation-free ordering exists, by emulating the four iBGP topologies (initial, final, and the two possible intermediate topologies) in a virtual environment.

**Unavoidable Dissemination Anomalies**

Fig. 9.7 shows an example in which all the reconfiguration orderings cause a dissemination anomaly. Observe that all the routers have a route to \( p \) in \( B_i \) and \( B_f \), as highlighted in the bottom part of the figure. However, in any reconfiguration ordering, one of the following cases applies.

- \( \text{add}(e_1, r_2) < \text{remove}(e_2, r_1) \). Consider the configuration generated immediately after \( \text{add}(e_1, r_2) \). Because of egress point preferences, \( r_2 \) selects path \( (r_2 e_1) \) while \( r_1 \) keeps selecting \( (r_1 e_2) \). Hence, neither \( r_1 \) nor \( r_2 \) propagate their best route further, causing top layer routers to have no route to \( p \).

- \( \text{remove}(e_2, r_1) < \text{add}(e_1, r_2) \). Consider the configuration generated immediately after \( \text{add}(e_2, r_1) \). Both \( r_1 \) and \( r_2 \) are forced to select the route from their respective clients, and propagate that route to their route-reflectors and peers. Because of egress point preferences, \( rr_1 \) \( (rr_2, \text{resp.}) \) will select route \( (r_2 rr_1) \) \( ((r_1 rr_2), \text{resp.}) \). Hence, neither \( rr_1 \) nor \( rr_2 \) propagate any route to \( rr_3 \), causing \( rr_3 \) to have no route to \( p \).

In both cases, we end up with a dissemination anomaly.
9.4. AN ALGORITHMIC APPROACH IS NOT VIABLE

Unavoidable Deflections and Loops

Similarly to control-plane issues, there are cases in which forwarding deflections and loops cannot be avoided by any reconfiguration ordering, even if the $B_i$ and $B_f$ are deflection-free.

Consider, for example, Fig. 9.8. Both $B_i$ and $B_f$ are not subject to any forwarding issue. Indeed, in $B_i$, all routers but $s$ select $e_0$ as an egress point, since $r_1$ does not receive the route announced by $e_1$, and $r_2$ prefers routes from $e_0$ over to those from $e_1$. Similarly, in $B_f$, all routers but $s$ select a path from $e_1$, since $r_2$ does not receive the route announced by $e_0$, and $r_1$ prefers routes from $e_1$ over those from $e_0$.

However, in any reconfiguration ordering one of the following cases apply.

- $\text{remove}(e_0, r_2) < \text{add}(e_1, r_1)$. In the configuration generated immediately after $\text{remove}(e_0, r_2)$, $r_1$ and $r_2$ are forced to select $(r_1 e_0)$ and $(r_2 e_1)$.
CHAPTER 9. SEAMLESS BGP RECONFIGURATIONS

Figure 9.8: Pylon gadget. No iBGP reconfiguration ordering can prevent a forwarding loop in intermediate configurations.

respectively, . Hence, a loop occurs between \( r_1 \) and \( r_2 \) (see the IGP topology).

- \( \text{add}(e_1, r_1) < \text{remove}(e_0, r_2) \). In the configuration generated immediately after \( \text{add}(e_1, r_1) \), \( r_1 \) and \( r_2 \) will select \((r_1, e_1)\) and \((r_2, e_0)\) respectively, because of the IGP topology. As a consequence, \( rr_1 \) and \( rr_2 \) will select \((rr_1, r_1, e_1)\) and \((rr_2, r_2, e_0)\) respectively, giving raise to a loop between \( rr_1 \) and \( rr_2 \) (see the IGP topology).

In both cases, a forwarding loop occurs in an intermediate configuration.

**Unavoidable Traffic Shifts**

Consider the example in Fig. 9.9. In this example, unintended traffic shifts happen for either \( p_1 \) or \( p_2 \), whatever the reconfiguration ordering is. Traffic
9.4. AN ALGORITHMIC APPROACH IS NOT VIABLE

Figure 9.9: PYRAMID gadget. No iBGP reconfiguration ordering can prevent unintended traffic shifts in intermediate configurations.

shifts happen at $t$. For prefix $p_1$, $t$ steadily selects $(t e_3)$ in $B_i$ since it receives no path from $r_1$ and $r_2$ whose best routes $(r_1 e_2)$ and $(r_2 e_1)$ are learned via an OVER session. In $B_f$, $r_1$ and $r_2$ steadily selects $(r_1 e_1)$ and $(r_2 e_4)$ respectively, hence $t$ will select $(t e_1)$, because of egress point preferences. For prefix $p_2$, $t$ steadily selects $(t e_5)$ in $B_i$ since it receives no path from $r_1$ and $r_2$ whose best routes $(r_1 e_2)$ and $(r_2 e_1)$ are learned via an OVER session. In $B_f$, $r_1$ and $r_2$ steadily selects $(r_1 e_3)$ and $(r_2 e_2)$ respectively, hence $t$ will select $(t e_2)$, because of egress point preferences.

However, in any reconfiguration ordering, one of the following cases applies.

- $\text{remove}(e_1, r_2) < \text{remove}(e_2, r_1)$. In this case, consider prefix $p_1$. Immediately after $\text{remove}(e_1, r_2)$, $r_1$ keeps selecting the route from $e_2$, but $r_2$ switches to $(r_2 e_4)$, and starts propagating that path to $t$. Because of egress point preferences, $t$ steadily selects $(t r_2 e_4)$, i.e., the route an-
announced by an egress point to which \( t \) will not send traffic in \( B_i \) nor in \( B_f \).

- \( \text{remove}(e_2, r_1) < \text{remove}(e_1, r_2) \). In this case, consider prefix \( p_2 \). Immediately after \( \text{remove}(e_2, r_1) \), \( r_2 \) keeps selecting the route from \( e_1 \), while \( r_1 \) switches to \( (r_1, e_3) \), and starts propagating that path to \( t \). Because of egress point preferences, \( t \) will steadily select \( (t, r_1, e_3) \), that is the route announced by an egress point to which \( t \) will not send traffic in \( B_i \) nor in \( B_f \).

In both cases, an unintended traffic shift occurs during the reconfiguration.

**eBGP Policy Changes**

Modifying eBGP policies can affect the set of routes injected in iBGP. In our theoretical model, changing an eBGP policy corresponds to temporarily change the set of egress sets and the mapping between given prefixes and their corresponding egress sets during the reconfiguration process. On the other hand, the iBGP topology is not modified. More formally, given \( C_i = (B_i, U_i, S_i, \Upsilon_i) \) and \( C_f = (B_f, U_f, S_f, \Upsilon_f) \), we have that \( B_i = B_f \) but possibly \( \Upsilon_i \neq \Upsilon_f \). Also, \( \Upsilon_j \) at a given step \( j \) can be different from both the initial and the final ones, i.e., \( \Upsilon_j \neq \Upsilon_i, \Upsilon_f \). The same applies to the set of egress sets, i.e., possibly \( S_i \neq S_f \) and for some migration step \( j \) \( S_j \neq S_i, S_f \).

Assuming again that eBGP routes do not change throughout the migration, egress sets and the \( \Upsilon \) function in intermediate configurations depends only on the BGP reconfiguration ordering. Unless \( B_i \) and \( B_f \) are guaranteed to be (signaling, forwarding and dissemination) correct for any possible set of egress points (refer to [GW02b] and Chapter 8), reconfiguration orderings can create signaling, dissemination, and forwarding anomalies during the migration. Observe, however, that check for any correctness property to be enforced for any combination of egress points has been shown to be an \( NP \)-hard problem. Also, there are cases in which unnecessary traffic shifts cannot be avoided by any configuration ordering.

Consider the example in Fig. 9.10, which we called INCONSISTENT gadget. It represents a scenario in which preference of eBGP routes \( R_1 \) and \( R_2 \) has to be modified, e.g., because a commercial relationship with a neighboring ISP has changed. In this case, an unnecessary traffic shift occurs in every migration ordering. Indeed, traffic towards \( p_1 \) is load-balanced among \( e_1 \) and \( e_2 \) in both \( B_i \) and \( B_f \), since \( r_1 \) and \( e_1 \) exit through \( e_1 \) while \( r_2 \) and \( e_2 \) use route \( R_2 \). However, if \( e_1 \) is migrated first, then all iBGP routers start preferring \( R_2 \)
9.5 Problem Complexity

We now study the computational complexity of the decision problem associated to SOCP. In particular, we restrict to oscillation-free orderings, and we prove that deciding if an oscillation-free ordering exists for a given reconfiguration is NP-hard. Observe that orderings cannot be seamless if they are not oscillation-free. Also, our results can be generalized to LoV-free and deflection-free orderings, as we discuss later in this section.

Safe Session Ordering Decision Problem (SODP): given $B_i$ and $B_f$, decide if there exists seamless ordering in which to add sessions in $B_f \setminus B_i$ and to remove sessions in $B_i \setminus B_f$.

Unfortunately, we prove that SODP is computationally hard, even if eBGP stability is assumed, and a single prefix is considered.

Figure 9.10: INCONSISTENT gadget, an eBGP policy reconfiguration case in which unintended traffic shifts occur in every reconfiguration ordering.

because the route is temporarily assigned a higher local-preference value with respect to $R_1$. Hence, $r_1$ and $e_1$ are subject to an unnecessary traffic shift after the migration of $e_1$ and before the migration of $e_2$. A symmetrical traffic shift occurs if $e_2$ is migrated before $e_1$. 
SODP is NP-Hard

In the following, we show that SODP is \( \mathcal{NP} \)-hard even for a single prefix entering at given set of egress points. In particular, we show that the well-known \( \mathcal{NP} \)-hard 3-sat problem can be reduced to SODP in polynomial time.

Let \( F \) be a boolean formula in conjunctive normal form. Each clause \( C_i \) in \( F \) contains three literals \( L_{ij} \), with \( j = 1, 2, 3 \). In turn, each literal is bound to a variable and has either the same or the opposite value of the variable. We say that a literal \( L_{ij} \) is a positive literal if \( L_{ij} = X_k \), for some \( X_k \) in \( F \), and we say that it is negative literal otherwise.

We now construct the SODP instance \( S = (B_i, B_f) \). We refer to a single prefix \( p \). The base structure of both \( B_i \) and \( B_f \) is represented in Fig. 9.11. It consists in four top layer routers, namely \( t_1, t_2, t_3, t_c \), among which \( t_c \) is also an egress point for \( p \). Vertices \( e_{z1}, e_{z2}, e_{t2}, e_{p1}, e_{p2} \) are also egress points for \( p \), while \( x_z \) and \( x_p \) are two route reflectors outside the top layer full mesh. In the figure, the edges in \( B_f \setminus B_i \) are tagged as add, while \( B_i \setminus B_f = \emptyset \). Observe that \( t_1, t_2, t_3 \) form a BAD-GADGET if \( e_{z1} \) is present and \( e_{p1} \) is not. This forces any oscillation-free reconfiguration to add \((e_{p1}, x_p)\) before \((e_{z1}, x_z)\).

**Property 9.1** A reconfiguration ordering is oscillation-free only if \( \text{add}(e_{p1}, x_p) < \text{add}(e_{z1}, x_z) \).

The rest of \( B_i \) and \( B_f \) depends on \( F \), and consists in variable and clause gadgets that are connected at \( x_z, x_p, \) and \( t_c \). Observe that \( t_c \) will always select its eBGP route, while \( x_z \) and \( x_p \) always select a path from one direct client of...
its. This enforces no path exchange between vertices in Fig. 9.11 and vertices in other parts of the topology. Also the following property holds.

**Property 9.2** Vertices in the base topology (see Fig. 9.11) are guaranteed to converge to a stable state in both $B_i$ and $B_f$.

We now describe how to build variable and clause gadgets. Given $F$, we add one variable gadget per variable in $F$, and one clause gadget for each clause in $F$. The variable gadget for any variable $X_i$ is depicted in Fig. 9.12. It consists of two egress point, namely $e_i$ and $\bar{e}_i$, and two route-reflectors, namely $x_i$ and $\bar{x}_i$. Session $(x_i, e_i)$ is in $B_f \setminus B_i$. We will map $\text{add}(x_i, e_i) < \text{remove}(x_p, e_p)$ to assign $X_i = \text{TRUE}$, while $\text{add}(x_i, e_i) > \text{remove}(x_p, e_p)$ corresponds to $X_i = \text{FALSE}$. This mapping prevents any variable from being simultaneously i) TRUE and FALSE, and ii) not TRUE and not FALSE.

Egress point preferences are set as in figure. $x_i$ prefers routes from its transitive clients. Observe that $x_i$ is guaranteed to receive either $e_{z_1}$ or $e_{z_2}$, hence we can disregard egress points which are less preferred than $e_{z_2}$. Moreover, $x_i$ cannot be influenced by the routing choice of any of its route-reflectors unless one of them has a direct iBGP path to either $e_{z_1}$, $e_{z_2}$, or $\bar{e}_i$. Also, $x_i$ and $\bar{x}_i$ form a DISAGREE if $x_i$ does not receive any path to $e_{z_1}$ and $\bar{x}_i$ does not receive any path to $e_{p2}$. Such a DISAGREE does not exist in $B_i$, since spoke path $(x_i, e_i)$
is missing, and is prevented from oscillating at $B_f$ since $x_i$ is guaranteed to select $e_{z1}$.

**Property 9.3** If none of $x_i$’s route-reflectors has a valid iBGP path to $e_{z1}$, $e_{z2}$, or $\bar{e}_i$ shorter than $x_i$, vertices in each variable gadget (see Fig. 9.12) are guaranteed to converge to a stable state in both $B_i$ and $B_f$.

**Property 9.4** If none of $x_i$’s route-reflectors has a valid iBGP path to $e_{z1}$, $e_{z2}$, or $\bar{e}_i$ shorter than $x_i$ and $\text{add}(e_{i}, x_i) \neq \text{add}(e_{p}, x_p)$, vertices in each variable gadget are guaranteed to converge to a stable state for any reconfiguration ordering.

We now prove a couple of interesting properties of this variable gadget, which we will use after for showing the correctness of the reduction. Intuitively, they are based on the fact that the \textsc{Disagree} between $x_i$ and $\bar{x}_i$ forces different choices on $x_i$ and $\bar{x}_i$ according to the relative order in which sessions $(e_{i}, x_{i})$ and $(e_{p_{1}}, x_{p})$ are added. In particular, the following Lemmas state that when variable $X_i$ is TRUE, router $x_i$ will advertise $e_i$ during the migration, namely in migration steps between addition of $(e_{p_{1}}, x_{p})$ and addition of $\text{add}(e_{z1}, x_{z})$. Otherwise, if variable $X_i$ is FALSE, router $x_i$ will advertise $\bar{e}_i$ in the same migration steps.

**Lemma 9.1** Assume that $\text{add}(e_{i}, x_{i}) < \text{add}(e_{p_{1}}, x_{p}) < \text{add}(e_{z1}, x_{z})$. Then, $x_i$ never selects $\bar{e}_i$, and selects $e_i$ at every step after $\text{add}(e_{i}, x_{i})$ and before $\text{add}(e_{z1}, x_{z})$.

**Proof:** Before $\text{add}(e_{i}, x_{i})$, $\bar{x}_i$ steadily selects $e_{p2}$, and $x_i$ steadily selects $e_{z2}$, because of egress point preferences and visibility. Consider now the steady state immediately after $\text{add}(e_{i}, x_{i})$. Vertex $x_i$ switches to $e_i$, because of egress point preferences, and announces it to $\bar{x}_i$. As a consequence, $\bar{x}_i$ also selects $e_i$, because of egress point preferences. Also, since $e_i$ is the most preferred egress point at $\bar{x}_i$, $\bar{x}_i$ will never change its choice until $x_i$ changes egress point. In particular, $\text{add}(e_{p_{1}}, x_{p})$ does not cause any change in path selection at $\bar{x}_i$, hence at $x_i$. Finally, after $\text{add}(e_{z1}, x_{z})$, $x_i$ switches to its most preferred egress point, i.e., $e_{z1}$, yielding the statement. $\square$

**Lemma 9.2** Assume that $\text{add}(e_{p_{1}}, x_{p}) < \text{add}(e_{z1}, x_{z})$ and $\text{add}(e_{p_{1}}, x_{p}) < \text{add}(e_{i}, x_{i})$. Then, $x_i$ never selects $e_i$, and selects $\bar{e}_i$ at every step after $\text{add}(e_{p_{1}}, x_{p})$ and before $\text{add}(e_{z1}, x_{z})$.
9.5. PROBLEM COMPLEXITY

\[ C_i = (X_1 \lor \bar{X}_2 \lor X_3) \]

Figure 9.13: Clause gadget corresponding to clause \( C_i \).

**Proof:** Before \( \text{add}(e_{p1}, x_p) \), \( \bar{x}_i \) and \( x_i \) steadily selects \( e_{p2} \) and \( e_{z2} \) respectively, because of egress point preferences and visibility. When \( (e_{p1}, x_p) \) is added, \( x_p \) switches to \( e_{p1} \) and does not propagate \( e_{p2} \) to its neighbors any more. Hence, \( \bar{x}_i \) switches to \( \bar{e}_i \), making it available at \( x_i \). Thus, \( x_i \) also switches to \( \bar{e}_i \). Also, \( x_i \) will never change its choice until it receives and steadily selects \( e_{z1} \), that is, after \( \text{add}(e_{z1}, x_z) \). \( \square \)

Also, for each clause \( C_i \) in \( F \), a clause gadget is added to both \( B_i \) and \( B_f \). An example of clause gadget is represented in Fig. 9.13. A clause gadget contains three literal vertices \( l_{ij} \), with \( j = 1, 2, 3 \), each corresponding to a literal in \( C_i \). In both \( B_i \) and \( B_f \), each literal vertex \( l_{ij} \) is a route-reflector of \( x_k \) if and only if \( L_{ij} = X_k \) or \( L_{ij} = \bar{X}_k \). In addition, a clause gadget also contains auxiliary vertices \( a_{i1} \) and \( a_{i2} \), which are route-reflectors of \( x_z \) and \( x_p \) respectively. Egress point preferences at auxiliary and literal vertices are such that a BAD-GADGET \( \Pi_i \) potentially exists in the clause gadget corresponding to clause \( C_i \). Pivot vertices in \( \Pi_i \) are \( \bar{U} = (a_{i1} \ l_{i1} \ l_{i2} \ l_{i3} \ a_{i2}) \), and rim paths are \( \bar{R} = ((a_{i1} \ l_{i1}) \ (l_{i1} \ l_{i2}) \ (l_{i2} \ l_{i3}) \ (l_{i3} \ a_{i2}) \ (a_{i2} \ a_{i1})) \). Spoke paths depend on literals in \( C_i \). In particular, for each positive literal \( L_{ij} = X_k \), the spoke path at the corresponding vertex \( l_{ij} \) is \( (l_{ij} \ x_k \ \bar{e}_k) \). On the contrary, for each negative
literal $L_{ij} = \bar{X}_k$, the spoke path at $l_{ij}$ is $(l_{ij} x_k e_k)$. Spoke path at $a_{i1}$ is always $(a_{i1} x_z e_{z2})$, while spoke path at $a_{i2}$ is $(a_{i2} x_p e_{p1})$. Intuitively, spoke paths at every literal vertex are available only if the corresponding literal is FALSE in $C_i$. On the contrary, spoke paths at auxiliary vertices are always available in migration steps between $add(e_{p1}, x_p)$ and before $add(e_{z1}, x_z)$. Hence, if all the three literals in a clause $C_i$ are false, then all routers $l_{ij}$ will be able to select their spoke paths, hence the BAD-GADGET $\Pi_i$ will oscillate in migration steps between $add(e_{p1}, x_p)$ and before $add(e_{z1}, x_z)$.

Observe that the connection points between each clause gadgets and the rest of the topology are $x_z, x_p, t_c$, and all $x_i$ corresponding to literal in the clause. Because of egress point preferences and iBGP route propagation rules, all those connection points always selects a route announced by one of its transitive clients, except $t_c$ which will always choose the eBGP route it receives. This provides isolation between each clause gadget and the rest of the network, that is, both $a_{ik}$ (with $k = 1, 2$) and $l_{ij}$ (with $j = 1, 2, 3$) receive routes from any other router that is not in the same clause gadget.

A possible IGP topology $U$ enforcing given path preferences is built by the following algorithm. Let $\Upsilon(p)$ be the set of egress points for $p$. For each pair $(r, e)$ with $r \notin \Upsilon(p)$ and $e \in \Upsilon(p)$, an edge is added to $U$. Edge weights are asymmetric. The weight $w(r, e)$ for $r$ to traverse $(r, e)$ is set according to path preferences in Fig. 9.11, 9.12, and 9.13. That is, $w(r, e) = 1$ if $e$ is the most preferred egress point for $r$, $w(r, e) = 2$ if $e$ is the second most preferred egress point for $r$, and so on. On the contrary, weight $w(e, r)$ for $e$ to traverse $(r, e)$ is set an arbitrarily high value, e.g., 100. This ensures that all shortest paths consist of a single edge.

We now prove that the reduction is correct. First of all, note that $B_i$ and $B_f$ are not prone to routing oscillations. Indeed, vertices in the base topology and in the variable gadgets are guaranteed to converge by Properties 9.2 and 9.3. Also, the BAD-GADGET among literal and auxiliary vertices in each clause gadget are prevented from oscillating in $B_i$ and in $B_f$. Indeed, in $B_i$, $a_{i2}$ cannot select its spoke path $(a_{i2} x_p e_{p1})$ since that path does not exist at all. In $B_f$, since $x_z$ selects $e_{z1}$, $a_{i1}$ does not receive any path to $e_{z2}$, hence it does not receive its spoke path $(a_{i1} x_z e_{z2})$. Observe that $B_i$ and $B_f$ are oscillation-free but not signaling correct, i.e., they are not safe for any possible combination of egress points.

We denote with $M$ a boolean assignment on variables in $F$. Also, we define $O(M)$ as the partial ordering corresponding to $M$ in which $add(e_{p1}, x_p) < add(e_{z1}, x_z)$. In the following, we show that $O(M)$ does not create any oscillation in a clause gadget $G$ if and only if $M$ satisfies the clause corresponding
to $G$. Observe that we disregard orderings where $\text{add}(e_{p1}, x_p) > \text{add}(e_{z1}, x_z)$ because they cannot be oscillation-free (see Property 9.1).

**Lemma 9.3** If $M$ satisfies a clause $C_i$, then $O(M)$ ensures that the corresponding clause gadget does not oscillate at any reconfiguration step.

**Proof:** Let $L^*$ be one of the literals of $C_i$ such that $L^* = \text{TRUE}$ in $M$. $L^*$ must exist since $M$ satisfies $C_i$ by hypothesis. Assume, without loss of generality, that $X_1$ is the variable associated to $L^*$. We have two cases.

- $L^* = X_1$. Then, $X_1 = \text{TRUE}$ in $M$ implies that $\text{add}(x_1, e_{11}) < \text{add}(x_p, e_{p1})$ in $O(M)$. By Lemma 9.1, $x_1$ never selects $\bar{e}_1$ in $O(M)$.
- $L^* = \bar{X}_1$. Then, $X_1 = \text{FALSE}$ in $M$ implies that $\text{add}(x_1, e_{11}) > \text{add}(x_p, e_{p1})$ in $O(M)$. By Lemma 9.2, $x_1$ never selects $e_1$ in $O(M)$.

In both cases, $l^*$ never receives its spoke in $\Pi_i$, which is always prevented from oscillating. □

**Lemma 9.4** If a given reconfiguration ordering $O(M)$ guarantees that a clause gadget corresponding to $C_i$ does not oscillate, then $M$ satisfies $C_i$.

**Proof:** Let $B_o$ be the intermediate configuration generated immediately after the addition of session $(x_p, e_{p1})$, if $O(M)$ is followed. Assume by contradiction that $M$ does not satisfy $C_i$. By construction, this implies the following ordering constraints on $O(M)$.

- For each positive literal $L_{ij} = X_k$ in $C_i$, we have $\text{add}(x_k, e_{k1}) > \text{add}(x_p, e_{p1})$. By Lemma 9.2, $x_k$ select $\bar{e}_k$ in $B_o$, and steadily propagates to $l_{ij}$.
- For each negative literal $L_{ij} = \bar{X}_k$ in $C_i$, we have $\text{add}(x_k, e_{k1}) < \text{add}(x_p, e_{p1})$. By Lemma 9.1, $x_k$ select $e_k$ in $B_o$, and steadily propagates to $l_{ij}$.

This means that all the spoke paths are available at all the literal vertices $l_{ij}$. Also, spoke paths are available at auxiliary vertices $a_{i1}$ and $a_{i2}$, since $(x_p, e_{p1})$ is already added, and $(x_z, e_{z1})$ is still not added in $B_o$. Hence, nothing prevents BAD-GADGET $\Pi_i$ from oscillating, contradicting the hypothesis. □

**Theorem 9.1** SODP is $NP$-hard.

**Proof:** Consider a logical formula $F$ in conjunctive normal form. Let $S$ be the SODP instance corresponding to $F$ (built as described above). We now prove that the reduction is correct.
if \( F \) is satisfiable, then there exists an oscillation-free reconfiguration ordering. Let \( M \) be a boolean assignment that satisfies \( F \). By Lemma 9.3, \( O(M) \) ensures that any clause gadget does not oscillate at any migration step. Vertices outside clause gadgets are also guaranteed to converge to a stable state by Properties 9.1 and 9.4.

- if \( F \) is not satisfiable, then there does not exist any oscillation-free configuration ordering. Assume by contradiction that an ordering \( O(M) \) which preserves safety do exist. Since, by construction, vertices in different clause gadget do not exchange routes between each other, then \( O(M) \) must enforce that no routing oscillation can arise in any clause gadget at any reconfiguration step. Then, by Lemma 9.4, \( M \) satisfies each clause in \( F \), contradicting the hypothesis.

The SODP instance built in the reduction is an example of an iBGP topology change in which some egress points are added to the bottom level of the route reflection hierarchy. The proof shows that migration anomalies can be created even in this simple scenario. Note that the proof can be easily extended to the case in which iBGP sessions are added to already connected routers. For example, a link (or a path) can be added in \( B_i \) between each egress point at the bottom layer and \( t_c \). In this case, the additional links will not cause any router to change its best path, since \( t_c \) will never propagate any route but the one it receives from its eBGP neighbor.

Observe that, from a theoretical point of view, adding egress points to the initial iBGP topology in the SODP instance reduced from 3-sat corresponds to increase the local-preference that those egress points apply to the eBGP routes they receive (without changing the iBGP topology). Hence, the framework of the proof above can be reused to show that the problem of deciding if a seamless reconfiguration ordering exists is \( \mathcal{NP} \)-hard also in case of eBGP policy changes. More precisely, we can build the corresponding instance \( \mathcal{I} \) of the eBGP policy change decision problem as follows. The iBGP topology in \( \mathcal{I} \) is the final one in \( \mathcal{I} \), and does not change during the reconfiguration. Let \( L \) be the maximum value of local-preference assigned to any route to \( p \) in the initial configuration in \( \mathcal{I} \). Each egress point added in \( \mathcal{I} \) is configured in \( \mathcal{I} \) so that it sets a local-preference value strictly smaller than \( L \) in the initial configuration and equal to \( L \) in the final one. This local-preference change is equivalent to add new iBGP sessions since the effect of both operations is to enrich the initial set of routes equally preferred AS-wide.
Also, consider the main intuition behind the reduction: we can summarize it as follows. During given migration steps, routers $x_i$ propagate to routers in the clause gadgets one among two routes depending on the boolean assignment of variables in the 3-sat problem. Combinations of routes which reflect boolean assignments making some clause false cause routing oscillations in the clause gadgets. It is easy to tweak the clause gadget in order to create forwarding and dissemination anomalies when they are fed with given combinations of routes. Hence, due to the modularity of the SODP instance built in the reduction, the main intuition of the proof can be leveraged to show the computational intractability of LoV-free and loop-free orderings.

Discussion

We have just shown that SODP is $NP$-hard. Observe that some $NP$-hard problems can be solved by algorithms (e.g., 3-SAT solvers, or Linear Program solvers) that work well in practice. Similarly, we presented a correct and complete algorithm which efficiently solves the $NP$-hard IGP reconfiguration problem real-world network topologies in Chapter 7. Unfortunately, at the time of writing, designing an algorithm or a heuristic to compute seamless BGP reconfiguration orderings in practice is not doable.

First of all, consider that checking an SPP (hence, an $i$-SPP) instance for any kind of correctness (i.e., signaling, dissemination, or forwarding) is $NP$-hard, and no complete algorithm is known for those problems.

Also, no sufficient and necessary condition is known that ensures a reconfiguration ordering to be either oscillation-free, LoV-free, or deflection-free. Moreover, no algorithm currently exists to pinpoint issues that are created by the addition (or removal) of a single iBGP session. Thus, identifying ordering constraints among sessions to be added and removed is not viable.

Exploring the research space of the reconfiguration problem seems the most straightforward way. Such an approach however is highly inefficient, as the number of orderings to be checked is equal to the permutations of the sessions to be reconfigured, in the worst case. Even worse, for each ordering at least one $NP$-hard problem must be solved at each migration step.

We expect that pruning the research space cannot be done efficiently as well. Intuitively, if an efficient algorithm existed that discriminates promising and non promising branches in the research space, it could be applied to $B_i$ in order to solve SODP, which however is $NP$-hard.

We argue that the fact that SODP belongs to NP is also questionable. Consider that not even a seamless ordering can be considered a succinct certificate
for SODP, unless a polynomial-time checkable characterization of seamless orderings exists. Indeed, checking whether an ordering is seamless cannot be done in polynomial time, as testing correctness of intermediate configurations is computationally hard. For all those reasons, we conjecture that the problem is even more complex than an \( \mathcal{NP} \)-hard problem.

**Conjecture 9.1** SODP is \( \mathcal{PSPACE} \)-hard.

### 9.6 A General Solution for BGP Migrations

Section 9.4 shows that seamless iBGP reconfigurations cannot be always achieved by just adding and removing sessions. Intuitively, the problem is that local changes can unpredictably impact routing decisions at remote iBGP routers.

We argue that additional configuration tools are needed to build a general approach guaranteeing seamless migrations in any reconfiguration scenario. We propose to run two distinct control-planes on all routers in the ISP, as it is normally suggested to perform IGP reconfigurations (e.g., [Hv10]). The co-existing control-planes should work in isolation (no route leakage from one plane to the other), and according to different configurations, e.g., one control-plane working in the initial configuration \( C_i \) and the other one working in the final configuration \( C_f \). Ideally, we also would like to tell each router which of the two control-planes should be used for forwarding in the data-plane. We refer to this approach as BGP *Ships-In-The-Night* (SITN).

#### Requirements and Challenges for Two Control Planes

The main advantage of BGP SITN is that it allows us to reconfigure a single router without affecting routing decision of other routers. Intuitively, we would like to have both the initial and the final configurations up and running network-wide, so that each router can compute both the initial and the final BGP routing tables (RIBs). Under this assumption, reconfiguring a router would be simply a matter of telling the router to start forwarding traffic according to the final RIB instead of the initial one.

Unfortunately, current routers cannot natively support multiple BGP routing processes on the same set of eBGP routes.

From an abstract point of view, the following functionalities are needed in order to implement BGP SITN:

- co-existence of multiple isolated routing processes on the same router; and
9.6. A GENERAL SOLUTION FOR BGP MIGRATIONS

- independent propagation of all the routes to all the routing processes within each router.

In order to simulate co-existence of multiple routing processes on the same router, we can leverage the Virtual Routing and Forwarding feature [cisb] available on commercial devices. This feature is especially used as a basis for MPLS L3VPNs and BGP multi-topology.

Basically, Virtual Routing and Forwarding creates isolated namespaces for prefixes by tagging each set of prefixes with a route distinguisher. Two routes having distinct route distinguishers cannot be compared, and can co-exist in the routing table. By default, namespaces do not share any route, though route import and export mechanisms enable leakage of best routes to given prefixes from one namespace to another. Each network interface of the router can be assigned to a single namespace in such a way that forwarding depends both on the destination prefix and on the ingress interface. In the following, we will refer to each namespace as VRF.

In order to simulate two control-planes running at the same time, we would use an initial VRF to run the initial configuration, and a final VRF to run the final configuration. Unfortunately, because of the one-to-one mapping between interfaces and VRFs, routes learned from external peers are collected in a single VRF. This prevents independent propagation of external routes to all the VRFs. In fact, except for the best routes, external routes cannot be propagated from one VRF to others, even with route import and export mechanisms. A workaround to achieve propagation of external routes to all the VRFs could be to configure multiple parallel eBGP peerings. However, this solution is not practical as it unnecessarily duplicates eBGP peerings and requires coordinated configuration changes on both sides of each of those peerings.

Forwarding in SITN

If two routers disagreed about which VRF a packet should be assigned to, the network could experience forwarding deflections, loops and congestion, hence packet loss. Thus, correct forwarding requires that every router on the data path of a packet forwards it according to the same VRF. For this reason, packets need to be tagged with VRF information.

We distinguish between explicit and implicit tagging. Explicit tagging involves modifying the packet to encode additional information which is processed at every router. Traffic encapsulation mechanisms, e.g. MPLS or GRE, are examples of explicit tagging. Conversely, implicit tagging requires no change
to data packets. Tags are inferred and assigned to packets on the basis of information at lower layers in the protocol stack, e.g., the logical interface packets are received on. An example of implicit tagging is what is commonly known as VRF-lite. In a VRF-lite based network, routers are configured with multiple logical interfaces on the same links and separate IGP instances are run in each VRF. In this case, the VRF tag is implicitly assigned to each data packet according to the destination MAC address of the frame.

Proposed Solution

The BGP SITN approach requires three key components: a dispatching mechanism to propagate all the external routes to multiple namespaces, a front-end interface which propagates iBGP updates from one “active” namespace to the eBGP neighbor, and a tagging mechanism, either implicit or explicit. While we can leverage multiple tagging mechanisms (MPLS and VRF-lite, for instance), we currently lack support for the other two key components.

To this end, we propose to interpose a proxy component between each border router and its eBGP peers, as depicted in Fig. 9.14. The architecture of the proxy is similar to the one of BGP-Mux [VF07] in that the proxy maintains an eBGP peering with external neighbors and one iBGP client session per VRF configured on the border router. However, we extend the architecture proposed in [VF07] to support the concept of “active” namespace and the selective propagation of iBGP updates to the eBGP neighbor. Indeed, the proxy distinguishes one active VRF from several passive VRFs. All VRFs receive external routes from eBGP peers, but only information in the active VRF is considered when sending eBGP updates to external neighbors. While the proxy can be implemented as a standalone device, its functionality could
be built directly inside border router to facilitate reconfigurations.

Since the proxy maintains eBGP peerings on behalf of a border router, it needs to be properly configured. However, the proxy configuration is simple as it only needs the following information.

- the address of each eBGP peer;
- for each VRF, the name of the VRF and the address of the interface on the border router which is assigned to that VRF; and
- the name of the active VRF.

Finally, the proxy must support a tagging mechanism that maps each VRF to a label and tags data packets coming from eBGP neighbors with the label associated to the active VRF.

Intuitively, reconfigurations are performed by dynamically switching VRFs from active to passive and vice versa. A BGP migration can be achieved on a per border router basis. Observe that when we change the active VRF on one proxy, the tagging mechanism ensures that every router in the network will use the same VRF to forward the data packet, avoiding forwarding anomalies.

The ability of switching a VRF from active to passive makes it easy to deploy changes at border routers, e.g., changing eBGP policies. Reconfigurations that involve iBGP topology changes need extra care. Whenever a reconfiguration encompasses addition or removal of an iBGP session, we run multiple iBGP sessions in parallel. By using route-maps, each router is mandated to filter out routes belonging to the initial (final, resp.) VRF over iBGP sessions that are not in the initial (final, resp.) configuration.

**Implementation**

In order to show the feasibility and effectiveness of our solution, we implemented a prototype that can perform seamless reconfigurations. The system is based on a *Configuration Generator* which, at each migration step, updates router configurations. The Configuration Generator is based on an extended version of the NCGuard tool [VPB08], to which we added support for VRFs and route-maps.

We implemented the proxy component as a standalone script (about 400 lines in Perl). As a tagging mechanism, we exploit the third-party BGP next-hop feature that implicitly maps packets from external neighbors to the active VRF. More precisely, whenever the active VRF is changed, the proxy advertises
to its eBGP peers a change of the BGP next-hop, forcing them to send data packets to the interface bound to the new active VRF. For this reason, the proxy does not need any packet forwarding ability. The proxy can be interposed between a border router and an eBGP neighbor without disruptions taking advantage of BGP graceful shutdown [FDP+11].

Our prototype proxy has some known limitations: first, it requires the ability to define logical interfaces on the border router; second, it requires the proxy, the external neighbor and the border router to share the same layer 2 infrastructure. However, these limitations could be easily avoided if the proxy component were integrated directly in the router operating system. Given the simple architecture of the proxy, we believe such an integration to be possible on commercial routers.

Finally, a central component of the system coordinates the Configuration Generator and the prototype proxy, and pilots BGP migrations by reconfiguring one border router at a time.

9.7 Case Study

Based on our prototype implementation, we simulated a full-mesh to route reflection reconfiguration of Geant, the pan-European research network. We run the simulation in a virtual environment on a Sun Fire X2250 (quad-core 3GHz CPUs with 32GB of RAM). Routers were emulated using a major router vendor operating system image.

In our case study, we assumed Geant to offer MPLS L3VPN services, with VRFs (one per customer) configured on the border routers, and MP-BGP running in the core of the network. We built the IGP and the iBGP configurations founding on the layer 2 topology of Geant [gea10]. The IGP configuration consists of a single area where link weights are inversely proportional to their speed. The route reflection configuration was designed on the basis of the geographical position of the routers, a design practice commonly used by network operators [ZB03]. The route reflection top layer is composed of four routers, namely, DE, FR, NL, and UK. The routers having a fiber link to one top layer router were assigned to the middle layer. The remaining routers were added to the bottom layer. Each router in the middle and in the bottom layer was assigned with two route reflectors belonging to the layer immediately above.

To identify the set of sites at which different customers connect to Geant, we used real-world BGP updates. We found 16 different sets of egress points that receive BGP routes for the same prefix. We mapped those sets on different
9.7. CASE STUDY

customers of Geant, and we injected through each of them a different summary prefix, representing all the prefixes for the customer.

Then, we evaluated two different reconfiguration strategies. In the first experiment, we reconfigured the network using our system. In particular, we configured the initial and the final VRFs on each border router, and added final UP/DOWN iBGP sessions to the iBGP configuration. Two route-maps per router ensured correct propagation of routes on the initial and final iBGP topologies. Then, to migrate a border router, we activated the final VRF on the border router. We proceeded one border router at a time. When the final VRF is used on all the border routers, we removed the initial iBGP sessions, the initial VRFs and both route-maps from the routers. In the second experiment, we followed the current best practices [Smi10, Hv10]. In particular, for each router to be migrated, we firstly activated the sessions with its route reflectors, then we waited for route propagation, and finally we removed the initial sessions. We applied a bottom-up reconfiguration order. In the order we applied, within each layer, we picked routers according to the alphabetical order of their names. We repeated each experiment 30 times to minimize the impact of factors beyond our control (e.g., related to the virtual environment). To measure possible traffic disruptions, we injected ICMP echo request from

Figure 9.15: Using our system, no packet was lost when converting the Geant network from an iBGP full-mesh to a route-reflection hierarchy. On the contrary, significant traffic losses occurred with current best practices.
each router towards each summary prefix throughout the migration process.

Fig 9.15 reports the median, and the 5th and 95th percentiles of ICMP packets lost during each migration step. No packet was lost using our framework, while current best practices induced forwarding loops between reconfiguration steps 13 and 20. As a consequence, packets were lost during approximately 30% of the migration time. We found that 7 routers lost traffic because of forwarding loops to two summary prefixes. Together, these two summary prefixes correspond to more than 60% of all the prefixes known by routers in Geant. Even worse, discovered loops affected Equal Cost Multi-Path (ECMP) traffic, which also overcomplicates possible debugging activities performed by network operators in a realistic scenario. We think that this use case show the advantage of relying on our framework, as it provably avoids packet losses that can affect traffic to a significant portion of the full routing table during several reconfiguration steps.

All the configurations that we generated, along with the IGP and iBGP topologies, are available online [com].

9.8 Related work

Considerable effort has been devoted to BGP configuration correctness [MWA02, GW02b, FB05] and iBGP topology design [RS06, VVKB06, BUM08]. However, to the best of our knowledge, few works are specifically targeted to approaches for modifying the iBGP configuration of a running network without impacting traffic.

The closest work that deals with reconfigurations and can be applied to BGP is [AWY08]. In that work, Alimi et al. propose firmware modifications that enable routers to manage a shadow configuration beyond the active configuration that devices use to forward data traffic. A shadow bit is set in IP packets to discriminate which configuration to use for forwarding. Shadow and active configurations can be switched using an ad-hoc commit protocol. The entire approach could be seen as a way to implement two BGP control-planes, hence close to our proposal. In this chapter, however, we evaluated simpler solutions to reconfigure BGP, and we justified the need for an additional control-plane by a thorough theoretical study. Also, our solution is more lightweight and easier to implement with respect to [AWY08], as it requires no device modification, and no ad-hoc protocol for either tagging packets and committing configuration changes.

Graceful session reset is tackled in [FDP+11, FBDC07]. Also, Route Refresh
and BGP Soft-Reset capabilities are standardized in [Che00]. Contrary to these approaches, in this work we aim at managing iBGP configuration changes which encompass many sessions and affect several routers in the network.

Recently, some techniques [WKB08, KRvdM10] have been proposed to enable virtual routers or parts of the configuration of routers (e.g., BGP session) to be moved from one physical device to another. Their works differ from ours as we aim at seamlessly changing network-wide configurations.

In [RFRW11], Reitblat et al. study the problem of consistent network updates in software defined networks. They propose a set of consistency properties and show how these properties can be preserved when changes are performed in the network. Unlike our approach, this work only applies to logically-centralized networks (e.g., OpenFlow).

Some recent work addressed network-wide IGP reconfigurations [RZC11] (also see Chapter 7). They are based on the idea of running two IGPs on the same network, and finding an operational ordering in which to reconfigure routers without creating forwarding anomalies. Proposing to rely on two BGP control-planes, we took inspiration from them in our solution. However, algorithm proposed for avoid packet losses in IGP reconfigurations cannot be extended to BGP, mainly because of the different nature of the protocols. Indeed, contrary to the neat route visibility ensured by link-state IGPs, only best routes are propagated from one BGP router to others, hence a single change on one BGP router can have unexpected side effects on routing information received by remote routers. We further discussed difficulties in building an algorithm to compute an operational ordering that ensure lossless BGP reconfigurations in Section 9.5.

9.9 Conclusions

Network operators regularly change router configurations. BGP reconfigurations are not exceptions, as confirmed by our analysis of a Tier-1 ISP’s historical configuration data. Since today’s SLAs are stringent, reconfigurations must be performed with minimal impact on data-plane traffic and without affecting service availability.

In this chapter we make three contributions. First, we show that routing and forwarding anomalies can occur during BGP reconfigurations, possibly resulting in high packet loss ratios. Unfortunately, current best practices do not provide theoretical guarantees. Also, they do incur in long-lasting anomalies even during common BGP reconfigurations, as we show by simulating a full-
mesh to route reflection reconfiguration on a Tier-1 ISP. Second, we study the problem of ordering router reconfigurations so that all intermediate configurations are anomaly-free. We show several cases where such an ordering simply does not exist. Even worse, the problem of deciding whether such an ordering exists is computationally intractable. Third, we propose a framework which overcomes those practical and theoretical limitations, and leverages existing technology to perform provably lossless BGP reconfigurations. Our framework allows routers to run multiple BGP control-planes in parallel. We describe an implementation of this framework, and illustrate its effectiveness through a case-study.

Several directions are left opened and may worth deeper study. From a theoretical point of view, at least two kinds of sub-problems can be investigated: making additional assumptions on the initial and the final configurations (e.g., on their correctness for any combination of egress points), and restricting to specific reconfiguration scenarios (e.g., cases in which the route reflection hierarchy is not completely overturned). In both cases, it would be interesting to understand whether the operational ordering problem remains intractable, and if heuristics can be used to solve it. Also, other practical solutions to BGP reconfiguration can be explored, e.g., by defining a tailored migration protocol.
Conclusions and Bibliography
Conclusions and Open Problems

Network management is a challenging task. In order to satisfy stringent Service Level Agreements and to accommodate the natural need for network evolution, network administrators typically face hard-to-solve issues originated by complex interactions among hundreds of heterogeneous devices running a variety of network protocols. Moreover, the degree of automation in network management is still limited, and operators must often deal with low-level and subtle configuration languages that are not designed to prevent human errors.

In this thesis, we studied research problems related to governance and evolution of routing in the Internet, mainly taking the perspective of a single Autonomous System. We focused on three common management activities, namely pre-deployment configuration validation, network monitoring, and configuration deployment. Firstly, we deepened the configuration testing problem of checking a set of BGP configuration for correctness properties. Indeed, it has been shown that BGP configurations exist (in both iBGP and eBGP) that are prone to control-plane and data-plane issues. In this context, we studied how to statically assess guaranteed routing convergence to a stable routing state. We showed the computational complexity of several decision problems related to BGP guaranteed convergence, and we proposed new sufficient and necessary conditions which relaxed the existing ones. Unfortunately, we found that all the most important problems related to this kind of static configuration testing are computationally intractable, as well as checking for sufficient conditions. However, exploiting the insight we gained in our theoretical study, we proposed a heuristic algorithm and we implemented a prototypical tool to efficiently check BGP configurations for guaranteed stability. The tool never misreports a BGP network that is guaranteed to converge to a stable state.

In the second part of the thesis, we studied novel approaches to monitor routing in operational networks. In order to monitor control-plane traffic generated by routing protocols, we designed a centralized architecture which
leverages the possibility of today’s routers to selectively copy packets from one network interface to a remote location. We developed code to implement a prototypical central monitoring station able to decode, process, and dump BGP messages. By experimental evaluation in a testbed, we ensured the performance degradation at routers to be limited, and the central monitoring station to be scalable. Regarding data-plane traffic, we analyzed the long-standing problem of how to accurately and efficiently compute traffic matrices, to quantify the amount of traffic entering and exiting a network at a given location. We proposed a decentralized architecture in which programmable routers autonomously compute parts of the traffic matrix that they can directly measure. Highly-optimized mechanisms are used for measuring traffic. Preliminary experimental evaluation of the additional workload on routers confirms that the approach is promising.

The third part of the thesis deals with the problem of installing a new configuration in a running network, with theoretical guarantees on the ability to not lose any data packet. We focused on network-wide configuration changes which involve link-state IGPs and BGP. In both cases, we firstly showed long-lasting routing and forwarding anomalies that can be created during the reconfiguration process, and we experimentally evaluated the likelihood for them to occur in practice, by using both inferred and private AS topologies. After assessing the impossibility of simple approaches, we proposed reconfiguration methodologies that provably guarantee no traffic loss. Our proposals are mainly based on the possibility of running two control-planes at the same time, and progressively activate the final one on a per-router basis. In the case of IGP reconfigurations, we described algorithms and heuristic to compute a lossless ordering in which to apply configuration changes, and we implemented a provisioning system that automates most of our methodology. We experimentally evaluated both proposed algorithms and system, assessing their effectiveness, and we compared our techniques with current best practices. Regarding BGP, we showed that the operational ordering problem is much more complex. Hence, we discuss how to leverage current technologies both to establish the two control-planes and perform lossless reconfigurations. We are currently working on prototype implementation and evaluation of proposed approach.

Yet, as pointed out in each chapter, there is plenty of room for further research activities. In this thesis, we have only scratched the surface of the configuration testing problem. In particular, we focused on checking routing configurations for guaranteed convergence to a stable state. Despite improvement of the state of the art, some theoretical problems remain open, among which determining a characterization for the SAFETY problem. Also, we think
that the same approach we adopted in this thesis should be applied to other
correctness properties (e.g., absence of forwarding issues, and assessment of
requirements on routing after a network failure) to be checked before the de-
ployment of new configurations. Regarding network monitoring, our proposals
can be improved and conveniently tailored to specific settings or business needs.
For example, in Chapter 6 we focused on PoP-to-PoP traffic matrices, but our
approach can be adapted to take into account additional protocols (e.g., MPLS)
and protocol-specific features (e.g., BGP next-hop self), and to support ad-
vanced monitoring requirements (e.g., network tomography). More in general,
our monitoring solutions show that today’s routers are able to do more than
just route and forward packets, as features like router programmability and
selective packet cloning are available on them. We believe that such features
have the potential to open new opportunities for several network management
tasks. Finally, the reconfiguration work can be extended in many directions.
Beyond testing the approaches we proposed in the real-world, we plan to ex-
tend our study to other protocols (e.g., distance-vector IGPs and multicast
protocols), to problems arising from combined reconfigurations (e.g., involv-
ing more than one protocol at the time), and to other configuration aspects
(e.g., providing guarantees on security policies throughout the reconfiguration
process). Moreover, reconfigurations already highlighted theoretical problems
that are not deeply studied and deserve attention from the research community.
Among them, we recall effective modeling of the interaction between different
protocols, and proposal of algorithms and heuristics for ensuring correctness
properties throughout a reconfiguration process.
Publications

Journal Publications


Conference Publications


Technical Reports


Bibliography


BIBLIOGRAPHY


[cisa] Configuring Local SPAN, Remote SPAN (RSPAN), and Encapsulated RSPAN (ERSPAN). Official Cisco Documentation, Cisco Systems Inc.


BIBLIOGRAPHY


[junb] Configuring Port Mirroring. Official Juniper Documentation, Juniper Networks Inc.


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


