Methods and techniques for disruption-free network reconfiguration

Laurent Vanbever

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

October 2012

Pôle d'ingénierie informatique
ICTEAM
Université catholique de Louvain
Louvain-la-Neuve
Belgique

Thesis Committee:

Yves **Deville**	UCL, Belgium
Christophe **De Vleeschouwer** (President)	UCL, Belgium
Anja **Feldmann**	TU Berlin, Germany
Olaf **Maennel**	Loughborough University, UK
Bruno **Quoitin**	Université de Mons, Belgium
Olivier **Bonaventure** (Advisor)	UCL, Belgium

Methods and techniques for disruption-free network reconfiguration

by Laurent Vanbever

Université catholique de Louvain
ICTEAM
Pôle d'ingénierie informatique
Place Sainte Barbe, 2
1348 Louvain-la-Neuve
Belgique

This work was partially supported by a FNRS-FRIA scholarship (Fonds pour la formation à la Recherche dans l'Industrie et dans l'Agriculture, rue d'Egmont 5, B-1000, Bruxelles, Belgique). This work was also partially supported by Alcatel-Lucent.

Preamble

Started as a research experiment, the Internet has evolved into a worldwide network interconnecting more than 40,000 separately operated networks. Together, these networks provide Internet access to more than two billion of hosts [164, 157]. In addition to interconnect an always increasing number of hosts, the Internet supports more and more mission critical applications such as e-business, IP television (IPTV), teleconferencing, Voice over IP (VoIP), or even remote surgery. Networks need thus to be highly available putting a high pressure on network operators who must avoid disruption at all costs.

Each network constituting the Internet must be correctly configured. Configuring a network is similar to programming a distributed infrastructure albeit, in a low-level language. Starting from a set of network-level goals, the behavior of each device is tweaked—most of the time, manually—by using low-level languages provided by each network vendor (e.g., Cisco CatOS/IOS/IOS-XR, Juniper JunOS/JunOSe, Alcatel TiMOS, Brocade IronWareOS, Force10 FTOS). The primary goal of these configuration languages is to pass devices the operational settings they need for their proper operation. The operational settings include the list of network protocols as well as the set of parameters that govern their behavior. Obviously, these settings must be semantically consistent across all devices in order for the network to operate correctly.

As configuring a network is still essentially manual and device specific, it is renowned to be time-consuming and error-prone [39, 87, 44, 66]. Actually, the configuration problem is so ubiquitous that human errors are held responsible for the majority (between 50 and 80 percent) of network downtimes [166, 184].

As a network evolves in time (e.g., in terms of size or services), its configuration must be adapted. In this thesis, we refer to the process of modifying the configuration as *reconfiguration* or *migration*. Reconfiguring a network consists in applying a set of configuration modifications on a set of devices. Reconfiguration can bring several benefits to the entire network infrastructure in terms of manageability (e.g., introducing a hierarchy), performance (e.g., improving the network convergence), stability (e.g., hiding the visibility of some events), and security (e.g., switching to a more secure routing protocol). Reconfiguration can also enlarge the portfolio of services offered by a network. Interestingly, these benefits can be reaped without buying new hardware.

While configuring a network is known to be hard, reconfiguring a network is even more complex. Indeed, shutting down and restarting the network with the new configuration is not a viable approach as some networks have to forward traffic 24/7. Therefore, the reconfiguration has to be done in-place, while the network is running. Since network operators desperately lack configuration tools, they often have no choice but to perform the live reconfiguration manually, device-by-device. Manual reconfiguration can make sense for highly localized changes, e.g., changing one parameter on one device. However, when the reconfiguration affects several devices, manual reconfiguration can break the network-wide consistency and trigger reconfiguration anomalies as non-reconfigured devices interact unpredictably with reconfigured devices. Worse yet, such reconfiguration anomalies can last for a significant percentage of the migration process and create severe and service-affecting outages.

Given the high availability requirement and the likelihood of creating reconfiguration anomalies, network operators are often reluctant to reconfigure their networks unless it is absolutely necessary [13, 175]. Doing so, they follow the conventional wisdom *"If it ain't broke, don't fix it"*. This situation hampers network flexibility and evolvability by delaying or even preventing useful reconfigurations.

In this thesis, we aim at enabling anomaly-free network reconfigurations. In particular, we focus on *routing* reconfigurations as they typically require to reconfigure each device in the network. To avoid reconfiguration anomalies, the key idea is to order the reconfiguration operations such that every intermediate state guarantees some global correctness properties. We formulate the reconfiguration problem as follows.

The reconfiguration problem. *Given an initial and a final anomaly-free routing configuration, find a sequence of configuration changes (if any) such that network anomalies never occur, during any migration step.*

The reconfiguration problem raises at least two additional questions: *Does an anomaly-free ordering always exist?* and *Is this ordering easy to find?* In the following, we will answer these questions and provide several practical reconfiguration techniques which apply to both intradomain (i.e., Interior Gateway Protocol, or IGP) and interdomain (i.e., Border Gateway Protocol, or BGP) routing protocols. To study each reconfiguration problem, we apply roughly the same methodology composed of three main steps. First, we carefully analyze the reconfiguration scenarios to identify what are the practical problems faced by network operators. Second, we elaborate a model of the reconfiguration problem so as to study it systematically and to discuss its computational complexity. Third, we develop provable reconfiguration techniques and evaluate them on real-networks. This thesis should thus be of interest to both the network theoretician and the network practitioner.

The thesis is divided into five parts. In the first part, we provide the necessary background material (Chapter 1). In the second part, we study IGP reconfigurations. We begin by characterizing the types of forwarding

anomalies that can appear in any IGP reconfiguration scenario (Chapter 2). In particular, we show that two types of forwarding anomalies can appear: forwarding loops and traffic shifts. Knowing *what* and *when* forwarding anomalies can happen, we then tackle the problem of avoiding them in two practically relevant reconfiguration scenarios: (i) reconfiguring link-state IGPs (Chapter 3) and (ii) transitioning a network from using a distance-vector IGP to a link-state IGP (Chapter 4). In the third part, we tackle the problem of reconfiguring a BGP network. To that extent, we begin by defining what a correct BGP network is (Chapter 5). In particular, we introduce a new correctness property—the dissemination correctness property—which models the ability of all the BGP routers to learn at least one route to each destination. We then study the BGP reconfiguration problem *per se* (Chapter 6). In the second and in the third parts, we consider the IGP and the BGP reconfiguration problems as independent. However, BGP depends on the IGP to work. In the fourth part, we therefore study the impact of IGP reconfiguration on BGP (Chapter 7). In the fifth part, we aim at solving one of the root causes behind reconfiguration problems: the difficulty of managing network configurations (Chapter 8). Finally, we draw conclusions and suggest future research directions (Chapter 9).

Bibliographic notes

Most of the work presented in this thesis appears in previously published conference proceedings and journals. The list of accepted and submitted publications is presented hereafter.

Journal publications

1. Lossless Migrations of Link-State IGPs,

   Laurent Vanbever, Stefano Vissicchio, Cristel Pelsser, Pierre François and Olivier Bonaventure, In *IEEE/ACM Transactions on Networking*, 2012, (To appear).

2. Improving Network Agility with Seamless BGP Reconfigurations,

   Stefano Vissicchio, Laurent Vanbever, Cristel Pelsser, Luca Cittadini, Pierre François and Olivier Bonaventure, In *IEEE/ACM Transactions on Networking*, 2012, (To appear).

Conference and workshop publications

1. Towards validated network configurations with NCGuard,

   Laurent Vanbever, Grégory Pardoen and Olivier Bonaventure, In *Proc. Internet Network Management Workshop 2008 (INM)*, Orlando, USA, October 2008

2. A Hierarchical Model for BGP Routing Policies,

   Laurent Vanbever, Bruno Quoitin and Olivier Bonaventure, In *Proc. of the ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of TOmorrow (PRESTO)*, Barcelona, Spain, August 2009

3. Seamless Network-Wide IGP Migrations,

   Laurent Vanbever, Stefano Vissicchio, Cristel Pelsser, Pierre François and Olivier Bonaventure, In *Proc. of ACM SIGCOMM*, Toronto, Canada, August 2011

4. iBGP Deceptions: More Sessions, Fewer Routes,

   Stefano Vissicchio, Luca Cittadini, Laurent Vanbever and Olivier Bonaventure, In *Proc. of IEEE INFOCOM*, Orlando, USA, March 2012

5. When the Cure is Worse than the Disease: the Impact of Graceful IGP Operations on BGP,

   Laurent Vanbever, Stefano Vissicchio, Luca Cittadini and Olivier Bonaventure, In *Proc. of IEEE INFOCOM*, Turin, Italy, April 2013

Acknowledgments

While it would be impossible for me to acknowledge every single person who helped me during my PhD thesis, a number of them deserve special thanks.

First and foremost, I would like to thank Professor Olivier Bonaventure for being such a great advisor and mentor. Olivier's scientific knowledge is impressive and I consider myself lucky to have learned networking under his guidance. After four years, I can confirm that Olivier takes great care of his researchers and I would like to thank him for his support when I had this crazy idea of following evening courses. Moreover, I would like to thank him for having given me the opportunity to do a PhD. Although I was not convinced before or even during, in the end, I think it was a great experience.

I would like to thank Yves Deville, Christophe De Vleeschouwer, Anja Feldmann, Olaf Maennel, and Bruno Quoitin for their participation in the jury of this thesis. Together, they have provided me with a lot of useful comments and advices on how to valorize my results.

As "strange" as it may sound, I would like to thank the "Italian gang", especially Stefano Vissicchio and Luca Cittadini. They are not only direct contributors (and reviewers) to this thesis but also good friends. I enjoyed very much working and hanging around with them as they both share this unique ability of being able to combine theory and practice. Besides me, if someone else ever complains about the number of theorems and proofs in this thesis, they are probably the main culprits. I would like also to thank Giuseppe Di Battista for his hospitality, Marco Chiesa and the entire Roma Tre crew for their warm welcome during my visit in their lab.

I owe a special thank to Cristel Pelsser. Cristel is a great researcher who shares the same view of networking as me. I guess that having the same advisor helps. Cristel brought a lot to my work and is a great reviewer. I am particularly indebted to her and Lionel, her husband. Not only have they been great guide when I was in Japan (thanks for the many evenings of delicious food!), but they also both hosted me after the massive earthquake that hit Japan in March 2011. An experience I am not going to forget. During my two visits in Japan, I also had the chance to work with Randy Bush. Randy is without any doubt the wiser person I know in the entire networking community. With his sharp advices and his broad knowledge, he is the kind of person every PhD student wants to work with. Thank you also for the daily (and freshly brewed!) coffee I used to drink at your place

before going to work. I would also want to thank the entire IIJ lab for their kindness during my stays, especially Kenjiro Cho who funded a part of my trip.

I was lucky enough to be surrounded by a lot of people who made my stay at UCL a pleasant experience. First, I would like to thank my former and present INL colleagues. Among them, I owe a special thank to Pierre François who mentored me since I was a master student and provided me with a bunch of useful advices during more than four years. Even if he is sometimes hard to follow, every single discussion we had was highly motivating. Also, I would like to thank more particularly Damien Saucez, Bruno Quoitin and Virginie Van den Schrieck for the numerous discussions (not necessarily work related) we had. The rest of the INL crew also contributed to this thesis, even indirectly. Thanks to Sébastien Barré, Sébastien Dawans, Gregory Detal, Fabien Duchêne, Benoit Donnet, Mickaël Hoerdt, Pascal Mérindol, Damien Leroy, Christoph Paasch, and Simon Van der Linden. Beyond INL, I also want to thank the members of the INGI department, especially the academic, the administrative and the technic staff for their help.

The collaborations and contacts with the following persons were also interesting and particularly pleasant. Dimitri Papadimitriou from Alcatel-Lucent, João Luís Sobrinho from Technical University of Lisbon with whom I had very nice and enlightening routing discussions. Geoffrey Xie from Naval Postgraduate School provided me with a lot of comments on the first part of the thesis and later became a collaborator. Jennifer Rexford from Princeton University for having set up for me a nice “marathon day” where I had the opportunity to exchange ideas with numerous Princeton staff and students. Thanks also to Aman Shaikh from AT&T Labs for his warm welcome and the interesting discussions that we had while I was visiting AT&T. I would also like to thank Dirk Haex, Jan Torreele and Pierre Wallemacq from BELNET for the time I spent there during my internship and also for the data they provided. I am also indebted to Jean-Luc Doumont who helped me improving some of the graphs presented in Chapter 3.

My thanks also go to my parents and to my sister for their support and encouragement. My friends, with a special thanks to Gregory Pardoen who shares my passion for building and orchestrating computer systems. I really enjoyed all the enthusiastic discussions we had whether they were network or not network-related. Moreover, without Gregory, NCGuard wouldn’t exist today.

Last but not least, I would like to thank Carine for her unlimited patience and support. Thank you for reminding me that there is a life beyond work.

Merci à tous!

Laurent Vanbever
October 2012

Table of contents

Part I

Background

Chapter 1

Internet routing

The Internet Protocol [107, 33] (IP) is a connectionless and unreliable protocol whose goal is to relay datagrams or IP packets across one or more networks. Different networks are connected together by intermediate systems known as IP *routers*. The role of an IP router is to forward IP packets from one network to another one based on their destination address. An IP router is connected to different networks via dedicated hardware known as network interfaces. An IP packet generated by a end-system (the source) and destined to another end-system (the destination) residing in another network is thus first transmitted to a router which forwards it to another router and so on until the final end-system is reached.

IP routing is the process of computing paths in a network along which routers forward IP packets to reach other destinations. IP routing is realized by a set of distributed routing protocols running on each router. IP routing drives IP forwarding which is the action of directing IP packets received on a network interface to another one.

Traditionally, IP routing is implemented by two families of IP routing protocols: intradomain and interdomain routing protocols. As their names indicate, intradomain routing protocols are used to compute forwarding paths within a routing domain, while interdomain protocols are used to compute forwarding paths across different routing domains. In the Internet, each routing domain is known as an Autonomous System (AS) and is operated under the same administrative control.

Intradomain and interdomain routing protocols differ in terms of the goal and the size of the problem they are facing. The goal of an intradomain routing protocol is to find optimal (with respect to some cost function) routes to all the internal destinations. The goal of an interdomain routing protocol is to find routes which are compliant with the commercial agreements that exist between the AS and its neighbors. In an intradomain routing protocol, all routers usually know the entire topology. In contrast, given the size of the Internet topology, interdomain routing protocols provide only a limited view of the topology. Interdomain routing protocols are thus information hiding protocols. Intradomain routing is implemented by

Interior Gateway Protocols (IGP). Examples of IGPs include OSPF [92], IS-IS [99], RIPv2 [67] and EIGRP [2]. Interdomain routing is implemented by an Exterior Gateway Protocol (EGP). Currently, only one EGP is used, the Border Gateway Protocol [118] (BGP).

This first chapter reviews the main concepts behind an IP router, intradomain and interdomain routing.

1.1 IP router

From an architectural point of view, a modern IP router can be divided in two dedicated planes: the *control-plane* and the *forwarding-plane*. A sketch of an IP router is illustrated in Fig. 1.1.

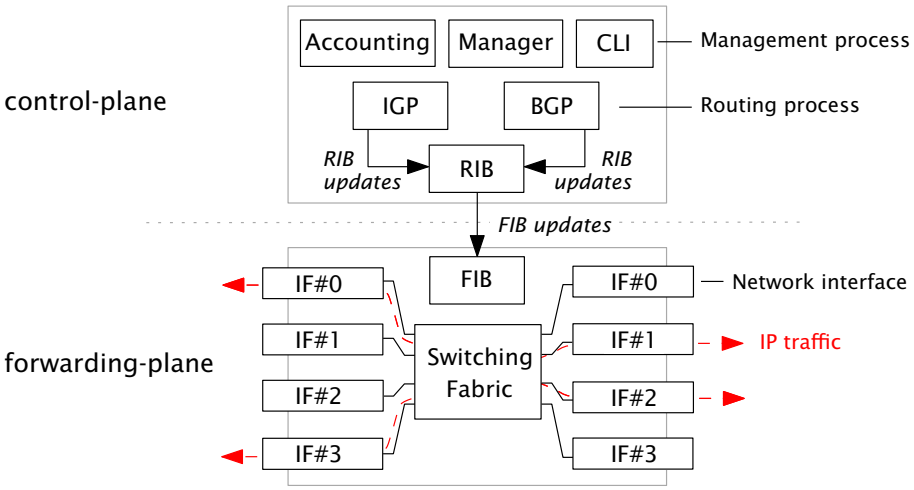


Figure 1.1 Sketch of an IP router. Modern IP routers are divided in two dedicated planes: the *control-plane*, in charge of IP routing, and the *forwarding-plane*, in charge of IP forwarding.

The control-plane is the brain of the router. It runs the management and routing processes and maintains the global routing table known as Routing Information Base (RIB). We distinguish between three types of management processes. The `Manager` process manages the entire router. It maintains the router configuration, and starts, stops or restarts routing processes according to it. The `CLI` process enables network operators to modify the router configuration, monitor the router state and troubleshoot the network. Finally, the `Accounting` process (e.g., SNMP [22]) enables operators to query the state of the router remotely. The control-plane also runs the routing protocols processes. Routing processes update the RIB which stores all the routes known by the router. Among all the known routes towards a destination, one is elected and is downloaded to the forwarding-plane. Finally, the control-plane is also responsible for han-

dling “special” IP packets, such as IP packets with options or IP packets directed to the router itself.

As its name indicates, the forwarding-plane is responsible for IP forwarding. It is composed of a set of input and output interfaces. Each interface is connected to a different network and is assigned an IP address. When a packet reaches an input interface, the forwarding-plane performs a lookup on the destination address in order to determine the output interface on which to relay it. The lookup is performed in a data structure called Forwarding Information Base (FIB). While the RIB contains all the routes known by the router, the FIB only contains the ones used for forwarding. Also, the FIB contains summarized informations with respect to the RIB. Once the output interface is determined, the forwarding-plane moves (“switches”) the packet from the input to the output interface across a switching fabric (e.g., shared bus, crossbar, shared memory). In addition to pure forwarding, the forwarding-plane is also responsible of buffering, filtering and scheduling IP packets.

1.2 Intradomain routing

The role of an intradomain routing protocol is to compute the shortest path to reach any IP prefix belonging to the domain. In an IGP, the shortest path minimizes the sum of the link metrics that compose it. If multiple shortest paths exist towards a destination, traffic is split evenly on the outgoing links. Link metrics are configured by the network operators, usually according to the link delay or to its bandwidth. For instance, in the Internet2 network (Fig. 1.2), link metrics are proportional to the distance between two adjacent routers. In this network, the shortest path between HOUS and NEWY is thus HOUS, ATLA, WASH, NEWY.

Two families of IGPs exist: *Distance-Vector* (DV) and *Link-State* (LS). These two families differ mainly by the dissemination mechanisms they use to propagate routing information and by the algorithm they use to compute the shortest paths.

In a LS IGP, a router describes its link connectivity in a Link-State Advertisement (LSA). This LSA is then reliably flooded in the entire routing domain. Whenever a router receives a LSA, it stores it in a database known as Link-State Database (LSDB). Based on the LSDB, each router builds a weighted directed graph representing the routing domain and uses the Dijkstra’s shortest path algorithm [34] to compute the shortest paths from itself towards every other router. Whenever the connectivity of a router changes (e.g., after a link failure), the router regenerates a LSA to allow other routers to compute new shortest paths if needed.

In a DV IGP, a router maintains a table containing the distance and the next-hop to reach each destination. It then periodically exchanges these entries with its neighbors or immediately after a topology change. To compute its shortest paths, each router implements a distributed version of the Bellman-Ford algorithm [12, 30]. When a router receives a routing en-

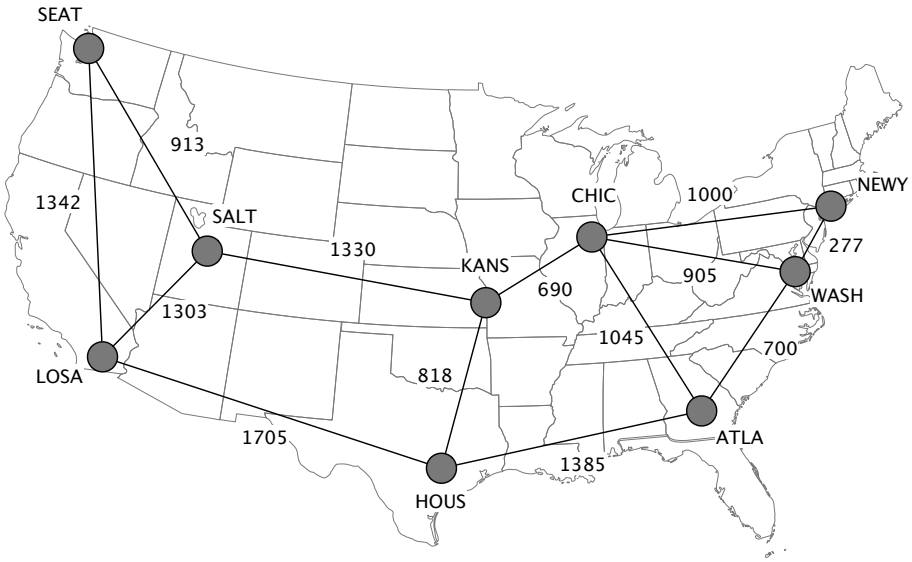


Figure 1.2 Topology of the Internet2 network [203].

try from a neighbor, it first increments the distance of each received route with the cost of reaching this neighbor. It then compares the compounded distance with the distance that it currently knows. If it is smaller, the router updates its table and installs the neighbor as next-hop to reach the destination. Consequently, in a DV IGP, a router does not know anything about the topology beyond the first hop and is thus completely dependent on the information provided by its direct neighbors.

In the early days of the Internet, DV IGPs were heavily used since they are relatively easy to implement and impose lower CPU and memory overhead than LS IGPs [2]. Unfortunately, in comparison with LS IGPs, DV IGPs suffer from severe limitations [191, 73, 69].

LS IGPs scale better To lower the overhead induced by flooding LSAs (bandwidth, memory and CPU), LS IGPs allow a routing domain to be hierarchically divided into zones. In a hierarchical IGP, the LSA flooding scope is limited to the zone in which the LSA has been originated. A router is thus aware of the complete topology of its own zone only. To maintain inter-zone connectivity, special LSAs summarizing zone connectivity are sent across zones boundaries. Current LS IGPs implementations can easily accommodate on the order of 10 000 nodes per routing domain [86]. In contrast, DV IGPs do not offer the ability to perform hierarchical routing.

LS IGPs converge faster LS IGPs are not prone to well-known convergence issues that plague DV IGPs like RIP's "count-to-infinity" or EIGRP's "stuck-in-active" problems [155]. Regarding convergence time, sub-second LS IGP convergence is now a fact, even in large networks [52]. Finally, LS

IGPs enable local protection mechanisms such as Loop Free Alternates [6] or MPLS Fast Reroute [100, 124] which can provide sub 50 milliseconds convergence.

LS IGPs enable advanced services The ability of LS IGPs to distribute information in the entire routing domain enables advanced services such as traffic engineering (e.g. with MPLS RSVP-TE [7] or with multi-topology routing [109, 108]).

LS IGPs are standardized LS IGPs are standardized and therefore can be used in multi-vendor environments. On the contrary, proprietary DV protocols (e.g. EIGRP) mandate the use of a single vendor. Having a multi-vendor environment is beneficial for at least two reasons: reduced costs and increased reliability. Indeed, different studies have shown that introducing a second vendor can reduce the capital expenditures by at least 30% [41, 42]. Furthermore, multi-vendor environments also reduce the probability of having network-wide failure due to software bugs affecting a single vendor [76].

1.3 Interdomain routing

The role of an interdomain routing protocol is to exchange reachability information between different domains so as to achieve global, Internet-wide connectivity. In today's Internet, BGP is the only used interdomain routing protocol [118]. As such, BGP is commonly known as "the glue that holds the Internet together". From an abstract point of view, BGP is a *path-vector* protocol. Each BGP route contains, in addition to the destination, the list of the ASes that the route has traversed. BGP is also a *single-path* protocol. Whenever a router learns multiple routes for a given destination, it selects one of them that it selectively propagates. Finally, BGP is *policy-based*. BGP policies allow each BGP router to independently prefer some routes over others as well as filter them. BGP policies are mainly used to enforce business relationships. Being a *path-vector* and a *single-path* protocol enables BGP to scale, while being *policy-based* enables BGP to be flexible and to implement highly complex policies. We now briefly review how BGP works.

In BGP, a route consists in one or more IP prefixes along with a set of attributes related to the path used to reach these prefixes. BGP routes are exchanged over TCP connections known as BGP sessions. There are two types of BGP sessions: external BGP session (eBGP session) and internal BGP session (iBGP session). eBGP sessions are used at the border of the routing domain, between routers belonging to different ASes. iBGP sessions are used between routers belonging to the same routing domain. Two routers connected by an eBGP (resp. iBGP) session are known as external (resp. internal) BGP peers or BGP neighbors.

Organization of a BGP router A BGP router performs three main tasks (Fig. 1.3). First, it collects and maintains BGP routes received from neighboring BGP routers in data structures known as `Adj-RIBs-In`. Second, it selects one best route for each prefix according to a Decision Process (DP) and stores it in a data structure known as `Loc-RIB`. Third, it selectively announces its best route to neighboring BGP routers and stores the announcements in data structures known as `Adj-RIBs-Out`.

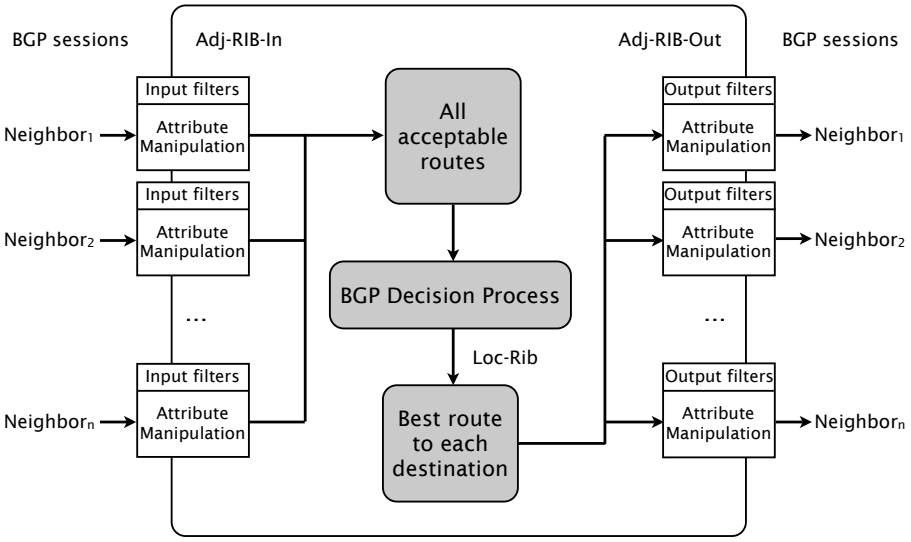


Figure 1.3 A schematic view of a BGP router. The Adj-RIB-In contains the BGP routes received from neighboring routers. The Loc-RIB contains the best route. The Ajd-RIB-Out contains the BGP routes announced to neighboring routers.

To control the BGP routes being learned and advertised, network operators can apply routing filters to each BGP session. Import filters specify which routes should be accepted, possibly after having changed some of the BGP attributes. Export filters specify which routes can be advertised, possibly after modifying some of their BGP attributes. A BGP routing filter is typically composed of a sequence of rules. Each rule is a couple of one predicate and a series of actions. The predicate defines which criteria a route must satisfy in order for the actions to be applied on the route. Actions can be as simple as accepting or denying the route. They can also be used to change the attributes of the route.

BGP routing filters are mainly used to: (i) enforce transit policies, (ii) perform traffic engineering and, (iii) sanitize BGP routes. Transit policies are used to constraint the propagation of BGP routes according to the business relationships between the AS and its neighbors. There are two typical business relationships [58]: *customer-provider* and *peer-to-peer*. The policy for advertising routes among customers, peers and providers is usually as follows. Customer routes are distributed to all BGP neighbors. Peer and provider routes are propagated to customers only. Transit policies

are not limited to the above business relationships and more specific policies exist [94, 121]. Traffic engineering policies are used to influence how BGP ranks and selects routes towards a specific destination. Finally, sanitization policies are used to prevent some routes to be learned or to be advertised to BGP neighbors. For instance, every self-respecting network operator configures BGP routing filters to make sure that routes towards invalid prefixes are not learned, nor advertised [177].

BGP decision process Best route selection is performed on each BGP router according to the 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 until there is only one route left. The evaluation of steps 1–4 is the same at every iBGP router since those steps refer to global attributes. Hence, all routers in the same ISP will eventually select routes that are equally preferred according to steps 1–4 (provided that policies are not applied on iBGP links [28]). Conversely, steps 5–9 involve metrics that have local significance for each BGP router. We now describe each decision step.

1. Prefer paths with the highest LOCAL-PREF.

   The LOCAL-PREF attribute encodes the degree of preference of a route. It is attached to each eBGP learned route and is announced only on iBGP sessions. The LOCAL-PREF usually reflects the business relationship that the AS has with the announcer of the route [20]. Typically, routes announced by a customer AS are associated with a higher LOCAL-PREF than the routes received from a peer AS which themselves are set with a higher LOCAL-PREF than the routes received from a provider AS.

2. Prefer paths with the shortest AS-PATH length.

   The AS-PATH attribute encodes the sequence of ASes traversed by the route. Consequently, it also indicates through which and how many ASes IP packets will flow if that route is used.[1]

3. Prefer paths with the lowest ORIGIN.

   The ORIGIN attribute reflects how the route was originated. It can take three different values: IGP, EGP, and INCOMPLETE. The preferences are as follows: IGP is preferred over EGP which is preferred over INCOMPLETE.

4. Prefer paths with the lowest Multi Exit Discriminator (MED).

   The MED attribute is used to rank the routes received from the same neighboring AS. Its role is to perform incoming traffic engineering. It is set by the neighboring AS to indicate which peering link should be preferred (provided that multiple links exist) for the prefixes it

[1] Observe that the AS-PATH encodes the reverse forwarding path only if there is no deflection along that path [66].

announces. Notice that the MED evaluation can lead to unpredictable network behavior such as routing oscillations [65].

5. Prefer eBGP learned paths over iBGP learned paths.

   From a business point of view, all the remaining paths are now equivalent. An AS will thus try to optimize the amount of internal resources it uses to forward traffic. As such, it will try to forward IP packets to other domains as quickly as possible. This principle is known as hot-potato routing or early-exit routing [132]. Here, a BGP router will prefer to use an external path over an internal one which requires packets to be sent within the AS.

6. Prefer paths with the nearest BGP NEXT-HOP.

   This is the second step of the hot-potato routing. Here, when choosing among two equally-preferred iBGP routes (i.e, two egress points), a BGP router will prefer to send its traffic to the nearest egress point according to the IGP metric. This step effectively couples the BGP and IGP control-planes. This coupling is known to be the source of multiple problems including forwarding loops and routing oscillations [66]. We will explore some of these problems in Chapter 6 and Chapter 7.

7-8. Deterministic tie-break.

   If there are remaining routes after the evaluation of the above criteria, a series of tie breaking rules are applied until one remains. These rules encompass preferring routes learned from a peer whose identifier is the lowest, and the ones learned from the session associated with the lowest neighbor address. Note that this very last step guarantees unicity as each neighbor address is unique.

iBGP organization The original BGP specification [118] prohibited a BGP router from advertising a route learned over an iBGP session over another iBGP session. This rule effectively mandated a full-mesh of iBGP sessions to guarantee route propagation. However, maintaining a full mesh of iBGP sessions does not scale as it requires $\mathcal{O}(n^2)$ unique sessions where n is the number of iBGP routers. To improve scalability, route reflection [11] was introduced along with the notion of route-reflector. Route-reflectors are special BGP routers which are allowed to advertise iBGP learned routes on some iBGP sessions. Actually, the iBGP sessions of each router are split into three sets: *clients*, *peers* and *route-reflectors*. Each BGP router propagates its best route according to the following rules: routes learned from a peer or from a route-reflector are relayed to clients only, whereas all other routes are reflected to all iBGP neighbors. In a fully-meshed iBGP network, all iBGP routers are peers. A *cluster* consists of one or more route-reflectors and all their clients. In addition to changing the original propagation rules, route-reflection also added two iBGP attributes: the

Scope	*Step*	*Criterion*
		Prefer routes ...
global	1	with higher `local-preference`
	2	with lower AS-PATH length
	3	with lower ORIGIN
	4	with lower MED
local	5	learned via eBGP
	6	with lower IGP metric to NEXT-HOP
	7	with lower ROUTER-ID
	8	coming from the BGP neighbor with the lowest IP address

Table 1.1 Whenever a BGP router learns multiple routes for the same destination (IP prefix), it selects one of them by using its decision process. The decision process is a set of rules: whenever there are ties for a rule, the next rule is applied until there is only one route left.

ORIGINATOR-ID and the CLUSTER-LIST. The ORIGINATOR-ID carries the identifier of the originator of the route in the local AS. The CLUSTER-LIST carries the list of clusters through which the route has been reflected. It is used to avoid routing loops as a route-reflector will discard any route in which its own cluster is found in the CLUSTER-LIST. Furthermore, the last steps of the decision process are modified to take into account these two attributes. Table 1.2 describes the BGP decision process in presence of route-reflection.

While route-reflection enables an iBGP topology to scale, it does so in a way that prevents routers from discovering the complete set of eBGP-learned routes. Consider the network topology depicted in Fig. 1.4. Circles represent routers having no clients, while diamonds represent route-reflectors. In this network, four routers learn an equally-preferred route for prefix p_1 and can thus act as egress points. The list along each route reflector represents the know egresses ranked with respect to the IGP distance. In an iBGP full-mesh, all the routers would be aware of the four routes. However, in this route-reflection topology, r_3 is only aware of e_1 and e_3 as it now relies on the previous decisions of its clients r_1 and r_2 to learn routes. Previous studies confirm that route-reflection leads to poor route diversity compared to an iBGP full-mesh [134]. This reduced visibility along with the coupling between BGP decisions and the underlying IGP (step 6 of the decision process) can create many routing and forwarding anomalies, including persistent routing oscillations and forwarding loops [66]. We will explore these problems and their consequences on the BGP reconfiguration process in Chapter 5, Chapter 6 and Chapter 7.

Scope	*Step*	*Criterion*
		Prefer routes ...
global	1	with higher `local-preference`
	2	with lower AS-PATH length
	3	with lower ORIGIN
	4	with lower MED
local	5	learned via eBGP
	6	with lower IGP metric to NEXT-HOP
	7	**with lower ORIGINATOR-ID**
	8	**with lower CLUSTER-LIST length**
	9	coming from the BGP neighbor with the lowest IP address

Table 1.2 When route-reflection is used, the last steps of the decision process are modified.

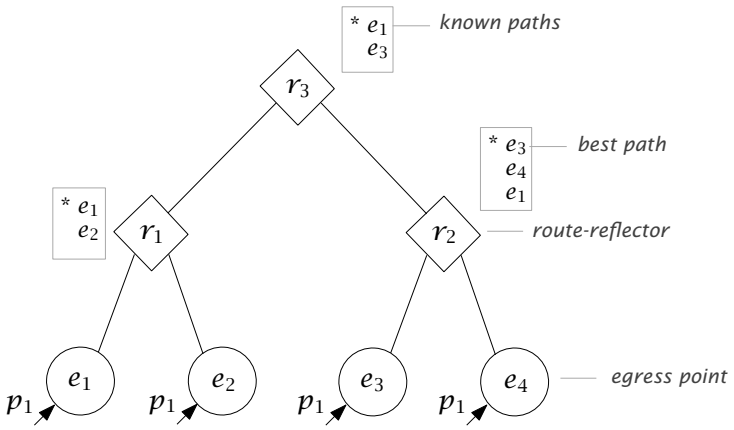


Figure 1.4 An iBGP route-reflection topology. With route-reflection, iBGP routers are hierarchically organized in clients and route-reflectors. Route-reflectors relay iBGP routes to and from their respective clients. Route-reflection also reduces the visibility of the BGP routes. In this topology, r_3 is only aware of the two routes announced by its clients while actually four routes are globally known.

Part II

Reconfiguring intradomain routing protocols

Chapter 2

A general characterization of IGP reconfiguration challenges

2.1 Introduction

As the network grows or when new services have to be deployed, network operators often need to perform large-scale IGP reconfigurations [68]. IGP reconfigurations can indeed improve the behavior of the entire network in terms of (i) manageability (e.g., introducing a hierarchy), (ii) performance (e.g, reducing the convergence time), (iii) stability (e.g., reducing the visibility of some events), and (iv) security (e.g., reducing the potentiality of attacks). Reconfiguring the IGP might also be a pre-requisite to implement value-added services (e.g., a link-state IGP must run prior to deploy MPLS traffic engineering as it is used to feed the traffic engineering database).

Despite its usefulness, large-scale IGP migrations are often avoided until they are absolutely necessary, as confirmed by many private communications with operators. This situation hampers network evolvability and innovation as some configuration changes are not made even if they could have improved the performance of the entire network. At least three different reasons explain this reluctance. First, network operators are concerned with the potential disruption of reconfiguring their IGP. Indeed, the IGP (e.g., IS-IS [99], OSPF [92], EIGRP [2], etc.) plays a critical role in the network as it enables end-to-end reachability between any pair of devices. Therefore, any problem happening during the reconfiguration can cause extensive service-affecting downtimes. Also, many signaling and routing protocols rely on an IGP to work (e.g. BGP, LDP, RSVP and PIM, etc.) and can therefore be affected during the IGP reconfiguration leading to even larger downtimes. Second, network operators typically lack appropriate tools and techniques to perform large, highly distributed changes to the configuration of their networks. Indeed, best current practices (e.g., [191, 68]) and even recent research efforts (e.g., [115]) restrict to specific sub-cases or target specific protocols, respectively. Third, network operators experience difficulties in understanding what is happening during the reconfiguration

as complex interactions may arise between upgraded and non-upgraded devices.

Before aiming at solving the IGP reconfiguration problem, we believe it is important to characterize precisely what type of anomalies can appear in any given IGP reconfiguration scenario. Knowing the type of reconfiguration anomalies and when they can occur is a critical information for network operators as it enables them to take preventive counter-measures. The main goal of this chapter is to answer the following question: *Given an IGP reconfiguration scenario, what types of reconfiguration anomalies can happen ?* To that extent, we propose a general classification which exposes the unknown relationship between the type of anomalies and the type of IGPs (namely, Link-State (LS) or Distance-Vector (DV)) involved in the IGP reconfiguration.

Surprisingly, we prove that forwarding loops can occur during an IGP reconfiguration *only* if the initial and the final IGP are LS. Forwarding loops arise when reconfigured and non-reconfigured routers forward traffic to each other in a conflicting manner. Henceforth, no forwarding loop can happen in a reconfiguration scenario which involves a DV protocol. Regarding traffic shifts, we show that any type of IGP reconfiguration can lead to traffic shifts in the intermediate state. Traffic shifts arise when routers start to use intermediate paths for forwarding packets which correspond neither to the initial forwarding paths, nor to the final ones. Traffic shifts are problematic as they increase the risk of creating congestion. As IGP reconfiguration strategy, we consider the best current practice [68, 179, 165, 191] which consists in running the initial and the final IGP configurations in different IGP processes and then switch from one to the other on a per-router basis. This reconfiguration strategy is usually referred to as “ships-in-the-night” (SITN).

The rest of the chapter is structured as follows. Section 2.2 presents our model, the reconfiguration strategy and formally defines the IGP reconfiguration problem. Section 2.3 describes the classification of the reconfiguration anomalies according to the family of the involved IGP. Finally, Section 2.4 concludes the chapter.

2.2 An abstract model for IGP reconfiguration

In this section, we aim at capturing IGP configurations and forwarding behavior of routers in a model that abstracts protocol-specific details.

We formally define an *IGP configuration* as a tuple (p, G, D, c). The tuple reflects configuration knobs available to operators. p is the identifier of an IGP protocol, e.g., EIGRP, OSPF or IS-IS. m is the mode in which the protocol is configured, namely flat or hierarchical. m is used essentially when dealing with IGPs that can be hierarchically organized (e.g. OSPF and IS-IS). For any IGP p, its *logical graph* $G = (V, E)$ is a weighted directed graph that represents the IGP adjacencies among routers participating in p. Each node in V represents an IGP router, and each edge (u, v) in E represents

an IGP adjacency between routers u and v. c models protocol-specific configuration. c typically encompasses the link weights. For link-state IGPs, c can further model if the logical graph G is hierarchically organized or not. Finally, $D \subseteq V$ is the set of IGP *destinations* for packets that flow in the network. A *destination* is a network prefix which can be originated (i.e., initially advertised) by one or more routers in V. In the following, 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[1]. Moreover, we always consider IGP is configured in such a way that every destination is reachable (i.e., G is connected and no arbitrary route filtering is performed), otherwise obvious forwarding anomalies can happen.

A *route* from a router r to a destination d learned via a protocol p is a simple path $P = (r \dots d)$ on G (the logical graph associated to p) which starts at r and ends at the router advertising d. Packets destined to one router $d \in D$ follow what we call a forwarding path. A forwarding path, or simply *path*, P from s to d is a path $P = (s\ r_1\ \dots\ r_k\ d)$ on G in which r_i, with $i = 1, \dots, k$, 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 associated to the corresponding edges. We denote with $dist(s, d)$ the weight of the shortest path from s to d.

Routers can and often use multiple IGPs [88] at the same time. This is especially true during the reconfiguration. For each configured IGP, a router spawns an independent routing process. This process maintains a protocol-specific routing table, commonly called *Routing Information Base* (RIB). A RIB entry is formally denoted by $rib(u, d, p, t)$, which holds the best-route router u knows from protocol p for destination d at time t. Whenever two or more routes for the same prefix are available in different IGPs, a router will select the one with the lowest *Administrative Distance* (AD). Table 2.1 lists the default AD values of the major routing protocols for two main router vendors. AD values can be manually overridden by network operators, even on a per-destination basis. After being selected, the route with the preferred AD is installed in the router-wide forwarding table, commonly called *Forwarding Information Base* (FIB). Thus, FIB entries possibly come from different IGPs. Contrary to RIB entries, FIB entries directly determine packet forwarding. We define the *next-hop* function $nh(u, d, t)$ which, given a router u and a destination d, associates the set of next-hops (i.e., FIB entries) that u uses at time t to send traffic to d. Notice that $|nh(u, d, t)|$ is not guaranteed to be equal to 1, since routers can use multiple paths to reach a given destination (e.g., in presence of ECMP). Fig. 2.1 represents how information coming from different routing processes are handled by an IP router.

[1] It is possible to generalize to the case of multiple routers originating the same destination, by adding to V a single virtual router that announces the destination to those routers (see Chapter 3).

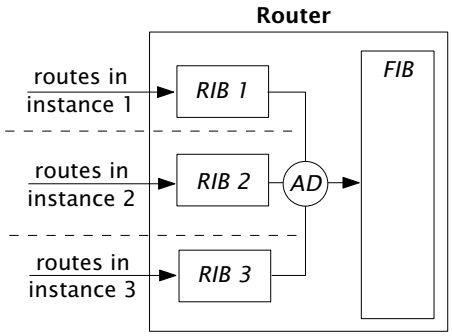


Figure 2.1 Model of a router running multiple IGPs. A specific routing table (RIB) is maintained for each protocol. The best routes of each protocol are installed in the router-wide forwarding table (FIB). When a route for a prefix p is available in more than one protocol, the one with the smallest AD is installed.

	Cisco	Juniper
Connected interface	0	0
Static route	1	5
EIGRP summary	5	-
OSPF internal	-	10
IS-IS L1 internal	-	15
IS-IS L2 internal	-	18
BGP external	20	-
EIGRP internal	90	-
OSPF all	110	-
IS-IS all	115	-
RIP	120	100
OSPF external	-	150
IS-IS L1 external	-	160
IS-IS L2 external	-	165
EIGRP external	170	-
BGP all	-	170
BGP internal	200	-

Table 2.1 Default Administrative Distance (AD) values for two different routing vendors [200, 185]. Routes learned by a protocol with the lowest AD are preferred.

To model the AD setting, we say that $p = pref(u, d, t)$ if the configuration of router u at time t is such that protocol p has the lowest AD for destination d. Similarly, we say that $p' = used(u, d, t)$ if the FIB entry on router u for destination d at time t was learned via protocol p'. Observe that $pref(u, d, t)$ depends on the router static configuration while $used(u, d, t)$ depends on the dynamic information learned by the router. Hence, it is possible that $pref(u, d, t) \neq used(u, d, t)$. This happens, for example, when a protocol has been configured with the lowest AD but does not offer to u any route to d at time t.

2.2.1 The IGP reconfiguration problem

In this thesis, we consider that the IGP reconfiguration process is carried out using a known migration technique called "ships-in-the-night" (SITN) routing [68, 179, 165, 191, 60], widely available on today's commercial routers. In SITN, the old and the new IGP configurations are running concurrently in separate routing processes. Fig. 2.2 depicts the reconfiguration of one router. First, the final IGP is introduced along with the initial one. The AD value of the final IGP is set such that it is strictly higher than the AD of the initial IGP. In particular, we set the AD of the routing process running the final IGP configuration to 255. This setting ensures that no route coming from that process is installed in the FIB [8, 200]. Then, the preference is progressively switched from the old to the new IGP by adjusting the AD values one router at a time. Once all the routers in the network have been migrated, the initial IGP is removed as it is not used anymore.

In the following, we say that a router is reconfigured when it prefers the new IGP over the initial one for all destinations. While AD can be configured on per-destination basis, we do not discriminate between per-router and per-destination reconfigurations, unless specified otherwise, as our main theoretical results apply to both of them.

An IGP reconfiguration carried out with SITN can be seen as a finite number of reconfiguration steps where each step consists in reconfiguring a router. For this reason, we consider that the time is discrete. We assume that the reconfiguration process starts at time $t = 0$ and ends after f steps (i.e., at time $t = f$).

At each reconfiguration step $1 \leq i < f$, only a subset of the routers has been reconfigured, hence different routers might prefer different IGPs for a given destination. To model such inconsistent AD settings, we define the concept of *actual path* $\pi(u, d, t)$ as the path[2] actually followed by packets sent by u towards d at time t resulting from a recursive concatenation of next-hops entries along the path. When several next-hops are available to reach a destination (e.g., in presence of more specific or less specific prefixes), the most specific entry is chosen. Formally, we define the actual path $\pi(u, d, t)$ from u to d at time t as the path $(v_0\ v_1\ \ldots\ v_k)$, such that

[2]For the sake of simplicity, we assume no ECMP in the rest of the chapter as it has no impact on our results.

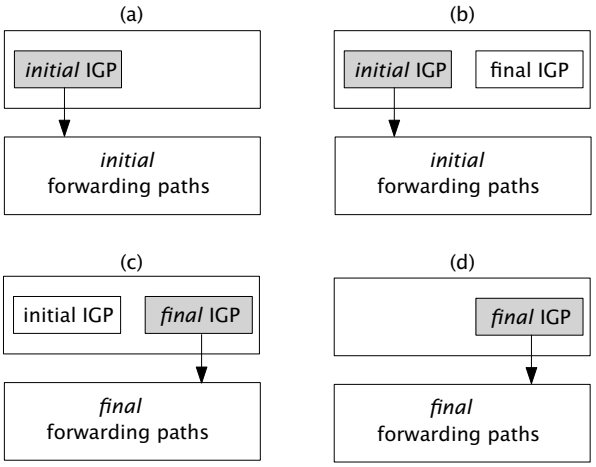


Figure 2.2 Details of the SITN IGP reconfiguration of one router. (a) The initial IGP configuration is the only one dictating the forwarding paths being used. (b) The final IGP configuration is then introduced in another IGP process. (c) Once the final IGP has converged, the preference is given to the final IGP so that it starts dictating the forwarding paths being used. (d) Once all routers have been migrated, the initial IGP is removed as it is not used anymore.

$v_0 = u$, $v_k = d$ and $\forall i \in \{0, \ldots, k-1\}$ $v_{i+1} \in nh(v_i, d, t)$. Observe that $v_k = d$ only if $\pi(r, d, t)$ does not contain a loop.

2.3 A general classification of IGP reconfigurations anomalies

In this section, we present our general classification in two steps. First, we define the *persistent* forwarding anomalies that can occur during an IGP reconfiguration. Observe that we do not consider *transient* forwarding anomalies that could arise during the normal IGP convergence process. Then, we explain when these anomalies can occur by highlighting the impact of the dissemination mechanisms used by the reconfigured protocols.

2.3.1 IGP reconfiguration anomalies

During an IGP reconfiguration, the possibly mismatching AD values that each router assigns to a given IGP can result in two kinds of persistent forwarding anomalies, namely: *forwarding loops* and *traffic shifts*.

A forwarding loop occurs when the actual path from a source s to a destination d contains a loop in which some routers use different protocols, giving raise to conflicting routing decisions. Fig. 2.3(a) depicts a reconfiguration scenario in which a forwarding loop can occur. With a slight abuse of notation, we denote with $\pi(*, d, t)$ the actual path used

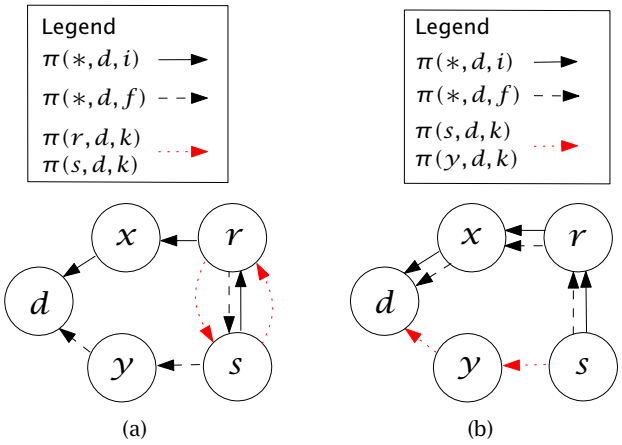


Figure 2.3 Abstract examples of (a) a forwarding loop and (b) a traffic shift.

by each router at time t. In this example, the path $(s\ r\ x\ d)$ is used initially, while the path $(r\ s\ y\ d)$ is used finally. At time k, we have that $used(s,d,k) \neq used(r,d,k)$. Moreover, $nh(s,d,k)$ contains r while, at the same time k, the $nh(r,d,k)$ contains s. Hence, the actual path from both r and s to d contains the loop $(s\ r\ s)$.

A traffic shift occurs if, during the reconfiguration, the actual path from a source s to a destination d changes more than once before reaching its final value $\pi(s,d,f)$. An abstract example of such a traffic shift is represented in Fig. 2.3(b). Both the initial and the final actual paths of s are $(s\ r\ x\ d)$. In the example, however, the actual path of s becomes $(s\ y\ d)$ at time $t = k$. Such traffic shift can occur because the logical graphs of the reconfigured protocols at time k are different from the final ones as a consequence of the reconfiguration of some routers (e.g., some routers are not participating in any of the two protocols anymore). Traffic shifts are fundamentally different from forwarding loops. Indeed, while multiple traffic shifts can cause forwarding loops, forwarding loops are not necessarily caused by traffic shifts.

While forwarding anomalies can happen during IGP reconfiguration, routing anomalies such as *loss of visibility* (LoV) and *routing oscillations* cannot. LoVs happen when the protocol converges to a state in which some routers have no route to some destinations. Routing oscillations happen when the protocol never stabilizes to a stable state. Routing anomalies cannot happen because IGPs are policy-free protocols which do not admit arbitrary route filtering and prescribe each router to apply the same decision process. In particular, each router is guaranteed to have a route to every destination in *at least* one of the two IGP protocols involved in the SITN. Also, IGPs (both LS and DV) are known to always converge to a stable state [63]. Contrary to IGPs, policy-based protocols such as BGP are known to be subject to routing oscillations [66]. In Chapter 5, we show that, in given configurations, BGP is also subject to LoV.

	LS	DV
LS	*forwarding loop* traffic shift	no forwarding loop traffic shift
DV	no forwarding loop traffic shift	no forwarding loop traffic shift

Table 2.2 General characterization of the migration anomalies that could happen in SITN IGP reconfiguration according to the family of protocols (DV and LS) involved in the migration. Counter-intuitively, forwarding loop can happen only in the LS to LS reconfiguration scenario.

2.3.2 The impact of the route dissemination mechanism

We now show that the type of forwarding anomalies possibly occurring during a IGP reconfiguration depends on the route dissemination mechanism adopted by the reconfigured protocols.

The dissemination mechanism of an IGP defines how routes are learned and distributed by routers participating in the protocol. We distinguish between the *independent route announcement* used by LS IGPs and the *dependent route announcement* used by DV IGPs. In a LS IGP, routers advertise the routes independently of whether they are installed in the FIB or not by the protocol. Indeed, in a LS IGP, routers reliably flood routes for connected subnets to all the routers in the network[3]. On the contrary, in a DV IGP, routers advertise only the routes installed in the FIB by the DV itself. This fundamental difference in the dissemination mechanisms can be attributed to the different algorithms used by each family of IGP to compute its all-pairs shortest path as pointed out in [3]. Indeed, LS IGPs rely on Dijkstra-like algorithms which require a global view of the topology, while DV IGPs rely on Bellman-Ford algorithms in which each router computation relies on the information propagated by its neighbors.

Property 2.1 is a direct consequence of the dissemination mechanism used by DV IGPs.

Property 2.1. *Let p be a DV protocol. For any router r and any destination d such that $p = used(r,d,t)$, then r learns a route to d from $nh(r,d,t)$.*

The general characterization of the anomalies that could happen in an IGP reconfiguration is reported in Table 2.2. We distinguish between two cases: (i) when all the protocols involved in the reconfiguration are LS and, (ii) when at least one of them is DV.

When two LS protocols are involved in an IGP reconfiguration, both forwarding loops and traffic shifts can arise. Indeed, due to the LS dissemination mechanism, all the routers have complete routing information in both protocols at any configuration step. However, routers can disagree on the IGP used to forward traffic, since some of them can be reconfigured

[3] We consider routes for connected subnets as routes belonging to any routing protocol.

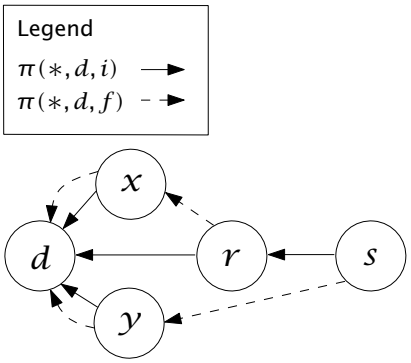


Figure 2.4 Example of a LS to LS reconfiguration in which a traffic shift occurs during the migration if r is migrated before s as the actual path of s becomes $\pi(s, d, t) = (s\ r\ x\ d)$ which corresponds neither to the initial path, nor to the final one.

while others cannot. Such disagreements can cause deflections along the actual path, which in turn are responsible for forwarding loops and traffic shifts. As illustration, consider again the Fig. 2.3(a) in which a LS protocol is used both initially and finally. In this topology, any ordering in which r is reconfigured before s creates the forwarding loop $(s\ r\ s)$ towards d. Regarding traffic shifts, consider the example depicted in Fig. 2.4 in which a LS protocol is again used both initially and finally. In this topology, a traffic shift is generated if r is migrated before s. Let t be the time at which r is migrated. We have that $\pi(s, d, t) = (s\ r\ x\ d)$ which is neither the initial path $\pi(s, d, i) = (s\ r\ d)$, nor the final one $\pi(s, d, f) = (s\ y\ d)$.

Surprisingly, whenever a DV IGP is involved in the IGP reconfiguration, no forwarding loop can occur, only traffic shifts. The absence of forwarding loop is guaranteed even if the routes in the initial and in the final configurations are completely different, e.g., if link weights are completely changed from one configuration to the other.

We prove the statement (see Theorem 2.1) in two steps. Lemma 2.1 proves that in order for a DV route to be propagated over a chain of routers, every router in the path must consistently use the same DV IGP.

Lemma 2.1. *If at a given time t a router r uses a route offered by a DV IGP p to d, then all the routers in $\pi(r, d, t)$ use p.*

Proof. Let $\pi(r, d, t) = (r\ x)P$. If $x = d$, then P is empty, and the statement trivially follows. Otherwise, by Property 2.1, r must have learned $\pi(r, d, t)$ from x. Also, by definition of a DV protocol, x must prefer p and use route P to reach d, otherwise x would not have advertised P to r. Hence, $P = \pi(x, d, t)$. The statement follows by iterating the same argument until the destination is reached. □

Intuitively, Lemma 2.1 states that the forwarding path of a router r that uses a DV IGP to reach destination d contains only other routers which

also use the same DV IGP to reach d. We can group all routers that use the DV IGP p to reach d at time t in a set $cone(p, d, t)$ that we call *DV-cone*. A consequence of Lemma 2.1 is that the actual path of all routers in the DV-cone is internal to the DV-cone itself. Moreover, the set of actual paths of routers in the DV-cone always form a tree thanks to the loop-freeness property of DV protocol. Consistent routing to the destination is therefore guaranteed as soon as a router in the DV-cone is reached. Leveraging this intuition, Theorem 2.1 proves the impossibility of having forwarding loops when a DV protocol is involved in the reconfiguration.

Theorem 2.1. *If a reconfiguration involves a DV protocol, then no forwarding loop can occur.*

Proof. Consider the migration from p_1 to p_2 and assume that p_1 is DV. For each router r and each destination d in the network, one of the following cases applies at every time t.

- r has no route to d.
- all the routers in $\pi(r, d, t)$ use p_1.
- all the routers in $\pi(r, d, t)$ use p_2.
- $\pi(r, d, t)$ contains at least one router that uses p_1 and another router that uses p_2. By Lemma 2.1, since p_1 is DV, $\pi(r, d, t)$ must be a concatenation of path PQ, where all the routers in P use p_2 and all the routers in Q use p_1.

In all the cases, $\pi(r, d, t)$ contains no forwarding loop. Observe that the proof is symmetric, henceforth it also holds if p_2 is DV, yielding the statement. □

Theorem 2.1 also holds when route summarization is introduced or removed during the IGP reconfiguration.

While the presence of a DV IGP guarantees the absence of forwarding loops during the reconfiguration, it does not guarantee the absence of traffic shifts. For example, consider again the scenario depicted in Fig. 2.3(b), and assume that the initial IGP p_1 is DV. Independently of whether the final IGP p_2 is LS or DV, a traffic shift occurs if r is the first router to be reconfigured. Immediately after that, s starts using y to reach d (i.e., $\pi(s, d, t) = (s\ y\ d)$). Indeed, s prefers a route propagated via p_1 over the route $(s\ r\ x\ d)$ received via p_2 since $pref(s, d, t) = p_1$. Intuitively, r leaves the DV-cone of d when it is reconfigured. A traffic shift follows since s was using r and it is still in the DV-cone as it learns another DV path via y. Also, observe that the reconfiguration ordering $(y\ s\ r\ x\ d)$ does not generate any traffic shift. This highlights how the applied reconfiguration ordering affects the number of traffic shifts that occurs as it determines the way in which the DV-cone changes during the reconfiguration. We will leverage this intuition in Chapter 4 when we will study the DV to LS reconfiguration scenario.

2.4 Conclusions

In this chapter, we have described a general characterization of the type of anomalies that can happen in *any* IGP reconfiguration relying on SITN. Our classification sheds light on the unknown dependency between the type of anomalies (loops and traffic shifts) and the dissemination mechanisms used by the involved IGPs. Surprisingly, we proved that forwarding loops can appear *only if* two LS protocols are involved. We also showed that traffic shifts can appear in any reconfiguration scenario.

For network operators, knowing in advance the type of reconfiguration problems that could happen in a given IGP reconfiguration scenario is particularly relevant since it enables them to take preventive countermeasures. We describe such measures for the two most relevant IGP reconfiguration in the following chapters. In Chapter 4, we show how to avoid the creation of forwarding loops in LS to LS reconfiguration. In Chapter 3, we show how to mitigate the effect of traffic shifts in DV to LS reconfigurations.

Although we considered only existing IGPs in this chapter, the generality of our classification allows us to support the theoretical study of reconfigurations between possibly any pair of routing protocols. As such, we envision the framework to be used for introducing protocols tailored for the specific needs of network operators [63].

Chapter 3

Lossless reconfiguration of link-state IGPs

3.1 Introduction

Nowadays, most network operators choose to rely on link-state (LS) IGPs and not on distance-vector IGPs, especially the ones working in an Internet Service Provider [52]. From a practical point of view, LS reconfigurations are therefore extremely relevant.

Typically, network operators target three aspects of their LS IGP when they perform large-scale migrations: (i) the protocol itself, (ii) the IGP's logical organization and (iii) the IGP's dissemination mechanism.

Changing the protocol Although belonging to the same family, each LS protocol provides a different features set [196]. As the network and the underlying requirements evolve, operators may want to change the adopted IGP to benefit from another features set. For instance, if the adopted IGP suffers from spoofing or injection attacks, network operators may want to switch to an IGP which, by design, prevents such attack from happening [179, 204]. Moreover, network operators may want to move to an IGP which is independent from the address family (e.g., OSPFv3, IS-IS), so as to run only one IGP to route both IPv4 and IPv6 traffic [165, 204]. Network operators may also need to change their IGP in order to integrate equipments which are not compliant with the one originally adopted [163]. Finally, network operators may change their IGP because of the familiarity with a given protocol [205] or after a merging of two or more networks in which a common IGP has to be deployed (see Section 3.10).

Changing the logical organization When the number of routers exceeds a given critical mass (e.g., the maximum amount of routes maintainable by the smallest routers), network operators often introduce a hierarchy within their IGP in order to limit the control-plane stress [32, 133]. By introducing a hierarchy, operators have also more control on route propagation by

tuning the way routes are propagated from one portion of the hierarchy to another [68]. In contrast, operators may also want to remove a hierarchy in order to ease the support of some traffic engineering extensions [122].

Changing the announcement mechanism Network operators might 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 more than 10,000 prefixes [86]. Route summarization also helps improving the stability by limiting the visibility of local events [112].

Some reconfiguration scenarios combine several of these aspects such as the migration from a hierarchical OSPF to a flat IS-IS [179]. Also, operators may be forced to revert back to a previous IGP configuration to meet new technical requirements [202].

In Chapter 2, we proved that LS to LS reconfigurations are subject to both traffic shifts and forwarding loops. Regarding traffic shifts, their negative effects can be mitigated by performing the reconfiguration when the network is lightly loaded (e.g., during maintenance window). In contrast, forwarding loops must be avoided even when the network is lightly loaded as they directly translate to traffic losses.

In this chapter, we aim at enabling *seamless link-state IGP migrations*, that is, progressive modifications of the link-state IGP configuration of a running network without losing packets due to forwarding loops. Since we only consider link-state IGP migrations, we use the terms LS IGP and IGP interchangeably. Our contribution is manifold. First, we analyze various scenarios of link-state IGP migrations. 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. Second, we introduce a methodology that enables seamless IGP migration while minimizing the number of reconfigurations per router. We show that, in real world networks, it is possible to find an ordering in which to reconfigure the routers while guaranteeing no forwarding loop. Although finding such an ordering is a $\mathcal{NP}$-complete problem, we propose algorithms and heuristics, and we show their practical effectiveness on several ISP networks. We also show how our techniques can deal with advanced reconfiguration scenarios by extending them to support link failures and network merging. Third, we describe the design and the evaluation of a provisioning system that automates the whole migration process according to our methodology. Our system generates router configurations, assesses the proper state of the network and updates all the routers in an appropriate sequence. 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.

The rest of the chapter is structured as follows. Section 3.2 provides a background on link-state IGPs and presents our abstract model. Section 3.3 formalizes the IGP migration problem and describes the migration scenarios we tackle. Section 3.4 presents our methodology. Section 3.5 proposes algorithms to compute a loop-free migration ordering. Section 3.6 presents our implementation. Section 3.7 evaluates our migration techniques on both inferred and real world topologies. Section 3.8 explains how to deal with network failures. Section 3.9 defines design guidelines that make IGP migrations easier. Section 3.10 describes how our reconfiguration techniques can support complex reconfigurations scenarios such as network merging. Section 3.11 presents related work. Finally, Section 3.12 concludes the chapter.

3.2 Link-state Interior Gateway Protocols

In this section, we provide additional background on link-state IGPs and slightly extend the model presented in Chapter 2. More specifically, we describe and include in the model two additional features provided by LS IGPs: (i) the ability of organizing the logical graph as a hierarchy and (ii) the ability to summarize route announcements in a hierarchically organized IGP.

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. It is either configured or decided based on the actual lengths of the preferred paths towards that destinations (e.g., picking the highest one [92]).

Modeling peculiarities of LS IGPs is possible by adding virtual nodes to the logical graph G or turning it into a multigraph. 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, one edge $(r\ r_j)$, 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 weighted 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.

By appropriately tuning the next-hop function, the IGP model of Chapter 2 can already capture specific forwarding details of IGP configurations such as the forwarding rules in hierarchical (resp. flat) mode or route summarization. In Section 3.3.1, we provide some examples of next-hop functions, actual path functions and migration loops in different migration scenarios.

3.3 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 3.1. *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 failures 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 3.8, 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. However, we argue that congestion issues are less critical, as they can be strongly mitigated by performing the migration during time slots during which the traffic is low. Also, large ISPs are normally over-provisioned [70, 16], further reducing the risk of congestion.

In this chapter, we focus on issues generated by the IGPs themselves. In Chapter 7, we study the issues due to the presence of additional routing protocols (i.e. BGP) in the network. In the rest of the chapter, we call *router migration* the replacement of the initial next-hop function nh_{init} with the

scenario	IGP configuration changes
protocol	protocol replacement
flat2hier	zones introduction
hier2flat	zones removal
hier2hier	zones reshaping
summarization	summarization introduction/removal

Table 3.1 IGP Migration Scenarios.

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. 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 3.3.1. Only in such cases, we compute and apply separate migration orderings for the troublesome destinations (see Section 3.4). The results of our evaluation (see Section 3.7) suggest that there are very few troublesome destinations in realistic topologies.

Throughout the chapter, we focus our attention on *migration loops*, i.e., 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 arising during protocol convergence as they last until specific routers are migrated (e.g., see Section 3.3.1). Observe that, if $nh_{init} = nh_{final}$, the actual path function π does not change either, hence any router migration ordering is ensured to be loop-free.

3.3.1 IGP migration scenarios

Table 3.1 presents the IGP migration scenarios we address in this chapter. We believe that those scenarios cover most of the network-wide IGP migrations that real-world ISPs can encounter. Each scenario concerns the modification of a specific feature of the IGP configuration. Moreover, different scenarios can be combined if more than one features of the IGP configuration have to be changed. We do not consider the change of link weights as a network-wide migration as ISPs typically change the weights of a few links at a time. Moreover, effective techniques have already been proposed for the graceful change of link weights [116, 53, 56, 57, 126]. Nevertheless, our generalized model and the techniques we present in Section 3.5 are also applicable to reconfigure link weights. Furthermore, since the addition and the removal of links and routers 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.

In the following, we describe the issues that must be addressed in each migration scenario we target.

Protocol replacement

This migration scenario consists in 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 [68]. 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 with 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 Section 3.2). 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. 3.1. In a *flat2hier* scenario, some routers change their next-hop towards destinations $E1$ and $E2$. In particular, the right side of Fig. 3.1 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 during which $B1$ 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 per-router migration. Consider, for example, the topology represented in Fig. 3.2. In this topology, symmetric constraints between $B1$ and $B2$ for traffic destined to $E2$ and $E3$ imply the impossibility of finding a per-router loop-free ordering. We refer the reader to the central and the

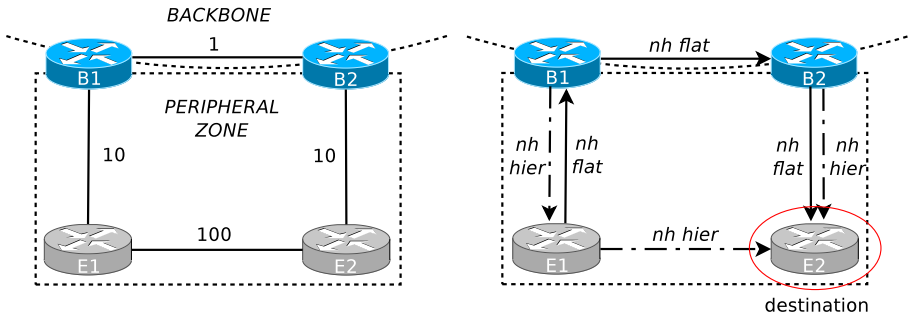


Figure 3.1 Bad Square Gadget. When the IGP hierarchy is modified, $B1$ starts preferring intra-zone path and forwards traffic destined to $E2$ via $E1$. Since initially $E1$ forwards traffic destined to $E2$ to $B1$, $E1$ must be migrated before $E1$ to avoid forwarding loops.

bottom parts of Fig. 3.2 to visualize the next-hop functions in flat and hierarchical modes. While a per-router ordering does not exist, it is easy to see that a per-destination ordering exist for both $E2$ and $E3$.

Similar examples can be found for *hier2flat* and *hier2hier* migrations. Observe that problems in hierarchy modification scenarios are mitigated in protocols such as IS-IS that natively support multiple adjacencies [99]. In fact, multiple adjacencies belonging to different zones decrease the number of cases in which the nh function changes during the migration. For instance, in Fig. 3.1, establishing an additional adjacency between $B1$ and $B2$ belonging to the peripheral zone would avoid the nh function to change between the initial and the final configuration. However, multiple adjacencies do not solve all the problems and migration loops can still arise depending on the initial and the final configurations.

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. 3.3. 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 $B1$ and $B2$ can lead to a forwarding loop between $B3$ and $B4$ for traffic destined to $E2$. Indeed, before summarizing routes, $B3$ and $B4$ prefer to send traffic destined to $E2$ via $B2$. On the other hand, when summarization is introduced, $B1$ and $B2$ propagate one aggregate for both $E1$ and $E2$ with the same weight. Hence, $B3$ and $B4$ change their next-hop since the path to $B1$ has a lower weight than the path to $B2$.

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. 3.3 so that symmetric constraints on the migration order hold between $B3$ and $B4$.

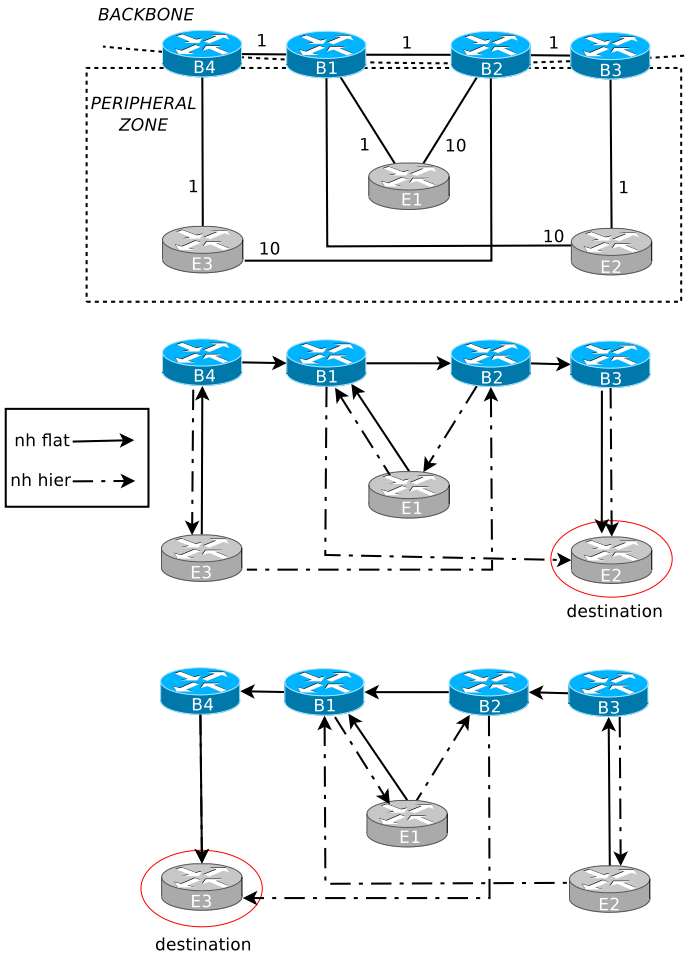


Figure 3.2 Loop Gadget. No migration ordering is loop-free for $flat2hier$ and $hier2flat$ scenarios because of contradictory constraints between $B1$ and $B2$ for destinations $E2$ and $E3$.

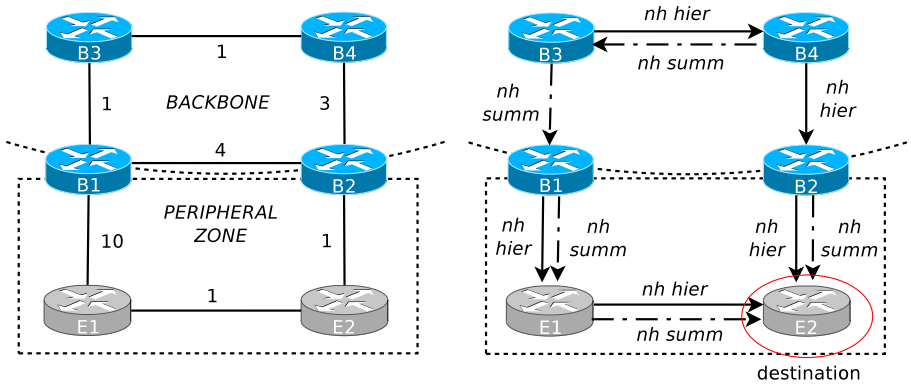


Figure 3.3 Route summarization gadget. When summarization is introduced or removed, a specific migration ordering is needed between $B3$ and $B4$ to avoid forwarding loops.

3.4 Methodology

In this section, we describe the main steps of our methodology (see Fig. 3.4). In the first step, we pre-compute an ordering in which to seamlessly migrate routers with no packet loss. When a per-router ordering does not exist, we first identify the set of problematic destinations for which contradictory ordering constraints exists. Then, we compute a per-destination ordering for each of them. Finally, we compute a per-router ordering for the non-problematic destinations.

Seamless IGP Migration Methodology

1. *Compute a lossless router migration order.* In case no per-router ordering exists, compute a per-destination ordering for the troublesome destinations.
2. *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.
3. *Monitor the final IGP status.* Wait for the convergence of the final IGP configuration.
4. *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.
5. *Remove the initial IGP configuration.* The initial IGP is removed from all the routers in the network.

Figure 3.4 Proposed methodology for seamless IGP migrations.

The actual migration process begins in the second step. In this step, we set the AD of the routing process running the final IGP configuration to 255. Recall that this setting ensures that no route coming from that process is installed in the FIB (see Chapter 2).

In the third step of the migration, we wait for network-wide convergence of 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 (see Table 2.1 for the default AD values). 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 can be done by using tailored route-maps matching the problematic destinations (see [162, 72]). After, we migrate the problematic destinations one by one, by lowering their AD following the pre-computed per-destination ordering. Since a routing entry change could take about $200ms$ before being reflected in the FIB [55], we wait for a given amount of 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 [48].

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.

3.5 Loop-free migrations

In this section, we present the two algorithms we use to compute a loop-free per-router ordering. First, we describe a correct and complete algorithm, the *Loop Enumeration Algorithm*. However, since the problem is $\mathcal{NP}$-complete [140], this algorithm is also inefficient and can take several hours to run on huge ISP networks (see Section 3.7). Then, we describe an efficient heuristic, the *Routing Trees Heuristic* which is correct but not complete. Finally, we also describe how to adapt the algorithms to compute a per-destination ordering to use as fallback when a per-router ordering does not exist.

In the following, we describe our algorithms in the absence of virtual nodes (see Section 3.2). The explicit support of virtual nodes is not needed as virtual nodes never change their respective next-hop function. Indeed, 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.

3.5.1 The Loop Enumeration Algorithm

The Loop Enumeration Algorithm (Fig. 3.5) 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 by using Johnson's algorithm [71]. 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. 3.6, indeed, migrating c_1 before at least one among c_2 and c_3 avoids the loop. In the algorithm, $V_{init,L}$ represents the set of routers that participate in the loop when they are in the initial configuration (line 6), and $V_{final,L}$ 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 13-25), where the variables t_{u_i} represent the migration steps at which routers can be safely migrated (lines 15-20). The binary variables Y_i are used to guarantee that at least one u_i router will be migrated before one v_i router. In particular, whenever a Y_i binary variable is equal to 0, the corresponding t_{u_i} is guaranteed to be less that t_{v_i}. Since the sum of Y_i is guaranteed to be less than $|Y_i|$ (line 23), at least one Y_i will be equal to 0. Additional constraints (lines 26-35) are also generated to ensure that two routers cannot be migrated at the same time, i.e. that all t_{u_i} are distinct. Finally, the algorithm tries to solve the integer linear program and returns a loop-free ordering if one exists (line 37).

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

Theorem 3.1. *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 $u_0 \vee \cdots \vee u_k < v_0 \vee \cdots \vee v_l$ 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 $d \in D$. We now show that L does not arise at any migration step if and only if the constraint is satisfied.

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 $\bar{t}$ such that all the routers in $V_{final,L}$ are migrated while all the routers

1: ***loop_enumeration_run(*** $G = (V, E)$***,*** D***,*** nh_{init}***,*** nh_{final}***)***
2: $CS \leftarrow \emptyset$
3: **for** $d \in D$ **do**
4: $\bar{G}_d = (V, \bar{E})$, with $\bar{E} = \{(u\ v)\}$ such that $v \in nh_{init}(u,d)$ or $v \in nh_{final}(u,d)$
5: **for** each cycle L in $\bar{G}_d$ **do**
6: $V_{init,L} = \{u \in L : \exists v, (u\ v) \in L, v \in nh_{init}(u,d)$ but $v \notin nh_{final}(u,d)\}$
7: $V_{final,L} = \{u \in L : \exists v, (u\ v) \in L, v \in nh_{final}(u,d)$ but $v \notin nh_{init}(u,d)\}$
8: $CS \leftarrow CS \cup \{u_0 \vee \cdots \vee u_k < v_0 \vee \cdots \vee v_l\}$, where $u_i \in V_{init,L}\ \forall i = 0, \ldots, k$, and $v_j \in V_{final,L}\ \forall j = 0, \ldots, l$.
9: **end for**
10: **end for**
11: $LP \leftarrow$ new LP problem
12: $Vars \leftarrow \emptyset$
13: **for** $u_0 \vee \cdots \vee u_k < v_0 \vee \cdots \vee v_l \in CS$ **do**
14: add to LP the following constraints
15: $t_{u_0} - MAX\_INT \times Y_1 < t_{v_0}$
16: $\ldots$
17: $t_{u_0} - MAX\_INT \times Y_l < t_{v_l}$
18: $t_{u_1} - MAX\_INT \times Y_{l+1} < t_{v_0}$
19: $\ldots$
20: $t_{u_k} - MAX\_INT \times Y_{l \times k} < t_{v_l}$
21: $t_{u_0}, \ldots, t_{u_k}, t_{v_0}, \ldots, t_{v_l}$ integer
22: $Y_1, \ldots, Y_{l \times k}$ binary
23: $\sum_{1<i<=l \times k} Y_i < l \times k$
24: $Vars \leftarrow Vars \cup \{t_{u_0}, \ldots, t_{u_k}, t_{v_0}, \ldots, t_{v_l}\}$
25: **end for**
26: **for** $t_i \in Vars$ **do**
27: **for** $t_j \in Vars$ **do**
28: **if** $t_i \neq t_j$ **then**
29: add to LP the following constraints
30: $t_i - MAX\_INT \times Z_1 < t_j$
31: $t_j - MAX\_INT \times Z_2 < t_i$
32: Z_1, Z_2 binary
33: $Z_1 + Z_2 = 1$
34: **end if**
35: **end for**
36: **end for**
37: **return** *solve_lp_problem(LP)*

Figure 3.5 Loop Enumeration Algorithm.

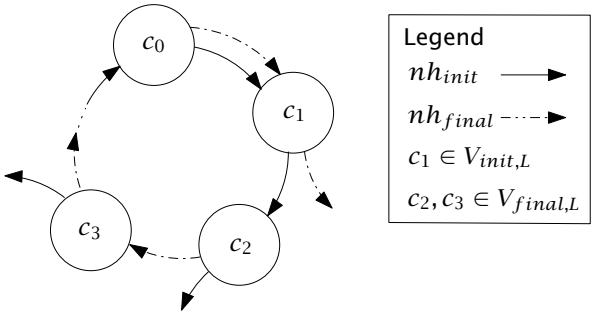


Figure 3.6 Abstract representation of a migration loop.

in $V_{init,L}$ are not migrated. Consider c_0. If $c_0 \in V_{final,L}$, then it is already migrated, i.e., $nh(c_0, d, \bar{t}) = nh_{final}(c_0, d)$, hence $c_1 \in nh(c_0, d, \bar{t})$, by definition of $V_{final,L}$. If $c_0 \in V_{init,L}$, then $nh(c_0, d, \bar{t}) = nh_{init}(c_0, d)$ and $c_1 \in nh(c_0, d, \bar{t})$. Finally, if $c_0 \notin V_{init,L}$ and $c_0 \notin V_{final,L}$, then $c_1 \in nh(c_0, d, t)$ $\forall t$. In any case, $c_1 \in nh(c_0, d, \bar{t})$. Iterating the same argument for all the routers in L, we conclude that $c_{i+1} \in nh(c_i, d, \bar{t})$, with $i = 0, \ldots, k$ and $c_{k+1} = c_0$. Thus, L arises at time $\bar{t}$.

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

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.

3.5.2 Routing Trees Heuristic

The Routing Tree Heuristic is illustrated in Fig. 3.7. 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 was proposed in [57, 53]). 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

1: ***routing_trees_run***$(G = (V, E), D, nh_{init}, nh_{final})$
2: $C \leftarrow \emptyset$
3: **for** $d \in D$ **do**
4: $S_d \leftarrow$ *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: **for** $P = (v_0 \ \ldots \ v_k)$, with $v_k = d$, $(v_i, v_{i+1}) \in E'$, and $predecessors(v_0) = \emptyset$ **do**
8: $last \leftarrow Null$
9: **for** $u \in P \cap \bar{V}_d$ and $u \notin S_d$ **do**
10: **if** $last \neq Null$ **then**
11: $C \leftarrow C \cup \{(u, last)\}$
12: **end if**
13: $last \leftarrow u$
14: **end for**
15: **end for**
16: **end for**
17: $G_c \leftarrow (V, C)$
18: **return** *topological_sort*(G_c)
19:
20: ***greedy_run***$(V, d, nh_{init}, nh_{final})$
21: $S_d \leftarrow \emptyset$
22: $N \leftarrow \{d\}$
23: **while** $N \neq \emptyset$ **do**
24: $S_d = S_d \cup N$
25: $N = \emptyset$
26: **for** $u \in V$, $u \notin S_d$ **do**
27: **if** $nh_{init}(u, d) \cup nh_{final}(u, d) \subseteq S_d$ **then**
28: $N = N \cup \{u\}$
29: **end if**
30: **end for**
31: **end while**
32: **return** S_d

Figure 3.7 Routing Trees Heuristic.

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 $(u \dots v \dots d) \in \pi_{final}(u, d)$, 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 $\tilde{V}_d$ 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 3.1. *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 $(u\ v_0\ \dots\ v_k\ u) \in \pi(u, d, t)$, 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 $w_0, \dots, w_n$ (in both the initial and final IGP configurations) are already in S_d, since each node in $\{w_0, \dots, w_n\}$ is added to S_d at a given iteration before i. Hence, $v_k \notin S_d$ at iteration i, because u is one of the next-hops of v_k and it is added to S_d at iteration i by hypothesis. Iterating the same argument, all routers $v_h \notin S_d$ at iteration i, $\forall h = 0, \dots, k$. As a consequence, the greedy procedure does not add u to S_d at iteration i, which is a contradiction. □

Theorem 3.2. *Let $S = x_1, \dots, x_n$ 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 $\tilde{V} = \{v_1, \dots, v_k\}$, such that $C = (u\ v_0\ \dots\ v_k\ u) \in \pi(u, d, t)$, at a certain time t. By Lemma 3.1, all $v_i \notin S_d$. By definition of the heuristic, all routers v_i are such that $nh(v_i, d, t) = nh_{final}(v_i, d)$, with $i = 0, \dots, k$, 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 $v_i \in \tilde{V}$ 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 neces-

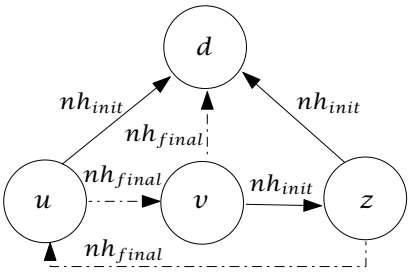


Figure 3.8 In some migration scenarios, the Routing Trees Heuristic generates unnecessary constraints.

sary. 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. 3.8, 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 [47], such cases are rare. In Section 3.7, we show that the heuristic found an ordering in most of our experiments on realistic topologies.

3.5.3 Per-destination ordering

If a per-router ordering does not exist or the Routing Tree Heuristic does not find a solution, a per-destination ordering can be computed. The per-destination ordering is applied to problematic destinations only, that is, to the destinations for which contradictory ordering constraints exist. Per-destination orderings can be computed directly from the ordering constraints identified by our per-router ordering algorithms. Note that it may not be necessary to compute an ordering for every problematic destination, since excluding a destination from the per-router ordering may unlock the problem for a set of other problematic destinations. The number of destinations for which a per-destination ordering is required can thus be minimized. However, our experimental evaluation (see Section 3.7) suggests that potentially troublesome destinations are few in practice, hence the need for minimizing problematic destinations is limited.

3.6 The provisioning system

We implemented a system which computes and automates all the required steps for a seamless migration. The architectural components of our system are depicted in Fig. 3.9. We now describe how data flows through the system (dashed lines in the figure), while stressing the role of each component.

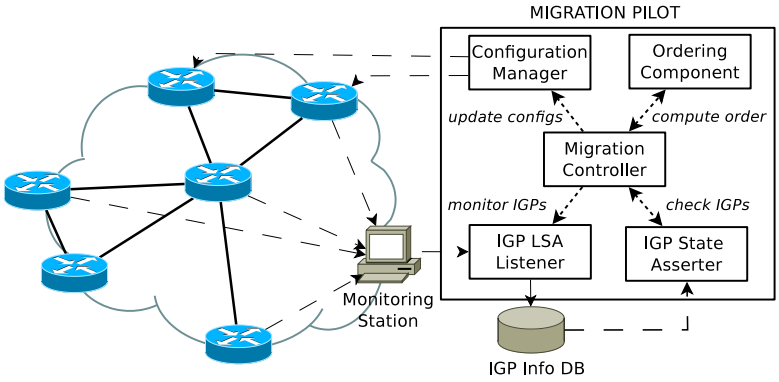


Figure 3.9 Design of the provisioning system. The provisioning system automates all the reconfiguration process. It computes the ordering, monitors the network and pushes the intermediate configurations in the appropriate order.

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 mechanism also enables failure detection. However, while we discuss how to prevent packet loss due to network failures in Section 3.8, 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. Such features have already been proved to be useful in the context of BGP monitoring [144] as a way to collect all control-plane messages with a low resource consumption. The *IGP State Asserter* 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 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 by the *Migration Controller*, which is 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 3.5.1. Then, the Migration Controller runs the IGP LSA Listener. When needed (see Section 3.4), 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 stabil-

ity 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 3.4. 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 [139].

3.7 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.

3.7.1 Data set and methodology

Our data set contains both publicly available and commercial ISP topologies. Concerning publicly available topologies, we used the inferred topologies provided by the Rocketfuel project [129]. 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 actual IGP topologies. We now 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 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 *ZBR*s and routers connected only to *ZBR*s as pure backbone routers. To simulate a *hier2hier* scenario, we 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 [154]. All the tests were run on a Sun Fire X2250 (quad-core 3GHz CPUs with 32GB of RAM).

3.7.2 Ordering algorithms

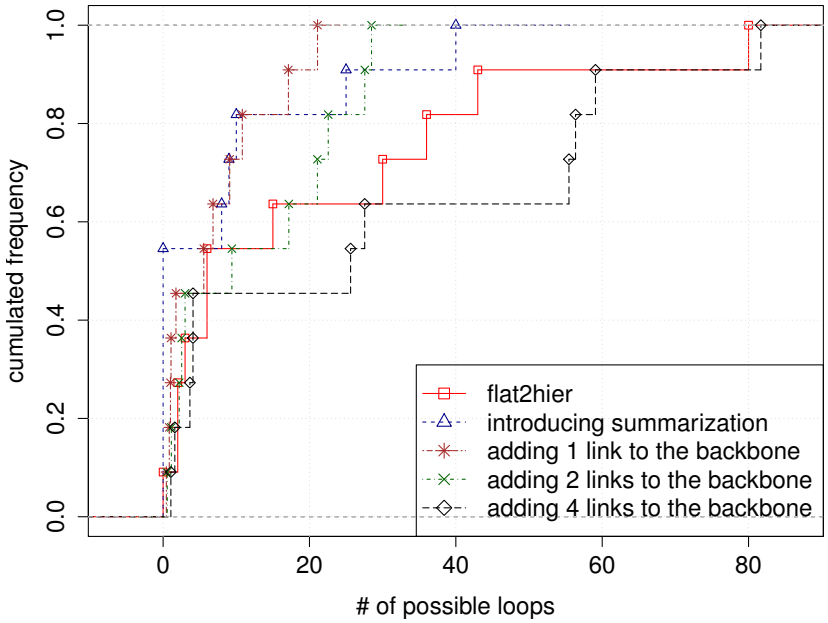


Figure 3.10 CDF of the number of loops that can arise on Rocketfuel topologies. In the worst case, up to 80 different forwarding loops can be created during the reconfiguration.

We first evaluate the usefulness and efficiency of the Loop Enumeration Algorithm and Routing Tree Heuristic. Fig. 3.10 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 reasons, it is desirable that migrations of world-wide networks be carried out on a per-zone basis, i.e., migrating all the routers in the same

zone (e.g., a continent) before routers in other zones. We argue that it is often possible to compute and apply per-zone orderings. Indeed, in both the Rocketfuel and the real-world topologies we analyzed, all the possible loops involve routers in the same zone or backbone routers and routers in a peripheral zone. 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. On the other hand, we found that huge number of problems could arise within $tier1.A$ in the $hier2flat$ scenario where the hierarchy was designed as described in Section 3.7.1. Indeed, more than 2000 loops might arise, involving up to 10 routers. Again, this stresses the importance of the IGP design on the migration outcome. We discuss simple design guidelines that ease IGP migrations in Section 3.9.

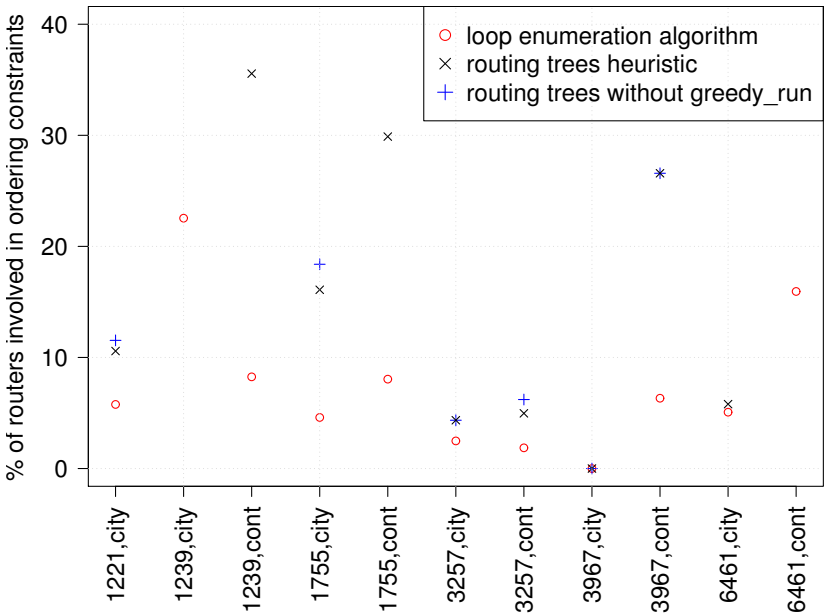


Figure 3.11 Percentage of routers involved in the ordering in *flat2hier* (Rocketfuel topologies). Results for other scenarios are similar.

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. 3.11 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-

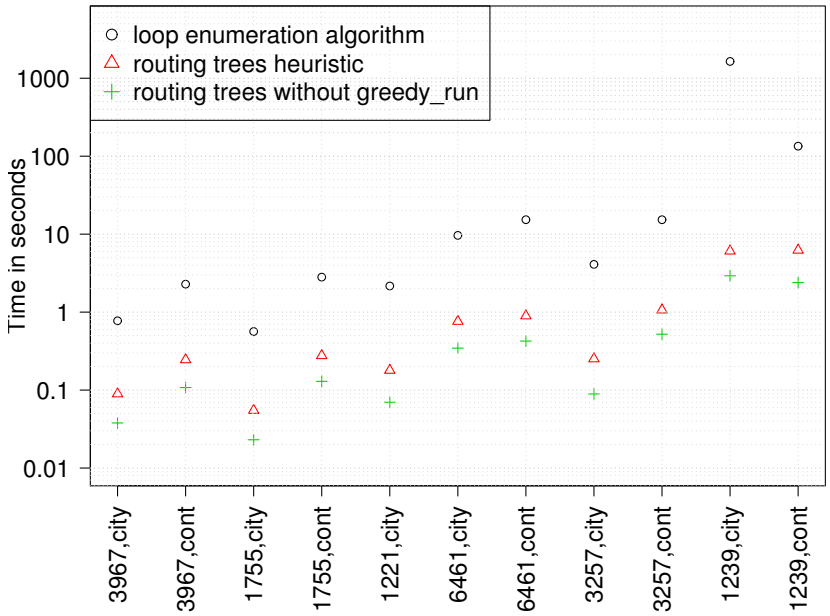


Figure 3.12 Time taken to compute an ordering in *flat2hier* (Rocketfuel topologies). Results for other scenarios are similar.

free ordering in all situations (including all the real-world topologies but $tier1.A$). 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. 3.11 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 3.8). Fig. 3.12 plots the median of the computation time taken by each algorithm over 50 separated runs. 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. 3.12, the low absolute value of the time taken by the Loop Enumeration Algorithm

can be explained by the relatively small size of the Rocketfuel topologies. Nevertheless, for the $tier1.A$ topology, the Loop Enumeration Algorithm did not complete even after of month of computation. To further evaluate the performance degradation of the complete algorithm, we enlarged $tier1.B$'s and $tier2$'s topologies. The operation consisted in replicating several 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.

3.7.3 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 operating system image. In particular, we simulated the migration from a flat IS-IS configuration to a hierarchical OSPF. Geant's topology is publicly available [168]. 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 [169]. 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 3.5.1), we found that 8 different loops towards 5 different destinations could arise on that topology.

To evaluate the cost of not following a proper migration ordering, we counted the number of loops appearing in 1000 random orderings. We observed that more than 50% of the orderings show at least one migration loop for more than 67% of the migration. To further illustrate the effect of not following the ordering, 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. 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. 3.13 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. 3.13 should be ascribed to

the virtual environment). On the other hand, adopting the naive approach of migrating routers in a 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.

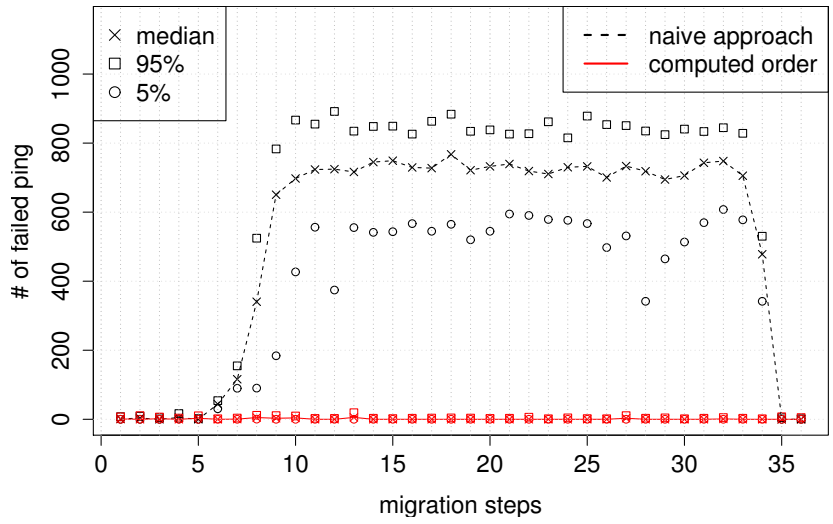


Figure 3.13 Our system guarantees that no packet is lost during migration while long-lasting connectivity disruptions can happen with a naive approach.

Our system also enables faster migrations than known migration techniques [179, 165]. The IGP LSA Listener is able to process IGP messages in a few milliseconds. The performance of this 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 graph, routing stability, and advertisement of the same set of prefixes in both IGPs. Even if the code could be optimized, current performance is good, also considering that the IGP Asserter does not need to be invoked more than once in absence of network failures (see Section 3.4). On average, the Configuration Manager took 8.84 seconds to push one intermediate configuration on a router. The average size of an intermediate configuration is around 38 lines. The entire migration process took less than 20 minutes. On the contrary, a similar real-world Geant migration took several days to be completed [165].

All the intermediate configurations that our system generated in the case study described above are available online [169].

3.8 Dealing with network failures

In this section, we show how to extend the algorithms described in Section 3.3 to deal with network failures.

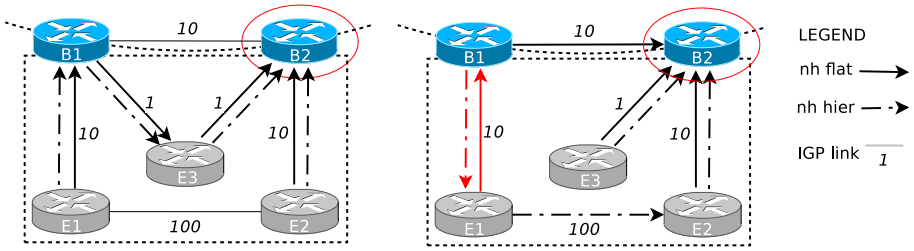


Figure 3.14 Link failures can change the reconfiguration ordering to be followed. In a $flat2hier$ scenario on this topology, a forwarding loop can appear between $E1$ and $B1$ if $B1$ is migrated before $E1$ and link $(B1\ E3)$ fails.

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 loops. Consider, for example, the topology in Fig. 3.14 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 3.14, $E1$ 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 [89]. 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 failure 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. 3.15 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 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. 3.15, 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 all possible single-link failures. Results on the real ISP topologies are similar. For $tier1.2$, we have been able to find a single-failure compliant ordering, while on $tier2.1$, we were able to find an ordering preventing loops for any single-link failure but one. 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 3.4) can be pre-computed for a subset of the destinations thanks to the efficiency of the heuristic approach.

Regarding the computation time, it took less than 2 hours to compute the additional set of constraints on all the topologies but two (namely, $AS1239, \{city, continent\}$). For $AS1239, \{city, continent\}$, it took 46 hours. As mentioned earlier, time efficiency is not critical, since a single-link failure compliant ordering can be computed offline, before performing the migration. Also, the process can be parallelized as every link can be treated separately. Once the additional set of constraints was built, it took less than 1 second in all the topologies to find out an ordering (if any). Given the size of the problem, we did not run our algorithm on $tier1.1$[1].

3.9 Design guidelines

In this section, we state simple design guidelines that can ease the entire IGP migration process, 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 simplifies $flat2hier$ 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 $nh_{init}(v, d) = nh_{final}(v, d)$ also for all routers $v \notin Z$ and $d \in Z$. This implies that no loop can arise

[1] 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.

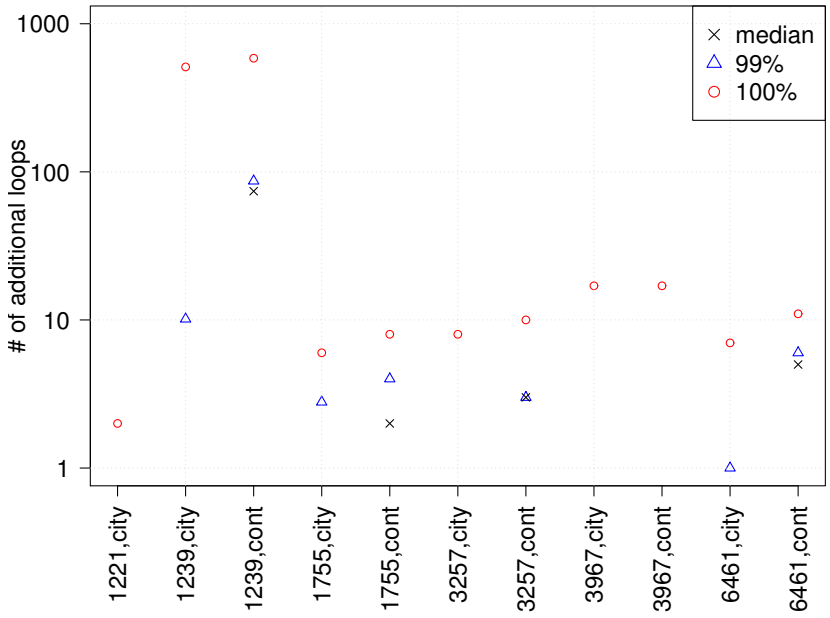


Figure 3.15 Average number of additional forwarding loops created by a single-link failure in *flat2hier* reconfiguration scenarios on Rocketfuel topologies. Missing points are equal to 0.

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 or using OSPF multi-area adjacency extensions [91]) between ZBRs also guarantees that 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 should be 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 [92]). 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.

3.10 Merging link-state IGPs

Since the early 1990s, the telecommunication industry has consolidated itself trough numerous networks merging and acquisitions [23, 152]. This trend is still ongoing. As an example, in 2011, AT&T tried to buy T-Mobile USA from Deutsche Telekom [187, 167], while Level3 and Global Crossing announced a company merging process [188]. While studying all the reasons behind a network merging is out of the scope of this thesis, one reason is to lower the operational costs thanks to economies of scale. However, before being able to reap such benefits, a considerable amount of work is often needed to integrate heterogenous networks. In this section, we make a first step in the direction of seamless network merging. In particular, we show how our reconfiguration techniques (Section 3.5) can be extended to perform seamless IGPs merging.

Two different approaches can be pursued when merging IGPs. The first approach is to keep the two IGPs as separate as possible and only tweak the interface between them. This approach is extremely easy with respect to the complexity of the reconfiguration process. However, this approach typically results in a network composed of a mosaic of different IGPs, which is hard to understand, maintain, and debug [88, 82, 14]. Also, to connect these IGPs together, network operators have no choice but to rely on unsafe primitives, like route redistribution [84] which is known to be prone to misconfiguration, forwarding loops and routing oscillations [83]. Indeed, while BGP could be used at the interface, it fails to realize several design objectives often encountered in enterprise networks [82].

The second approach consists in completely merging the two IGPs, such that the resulting IGP presents the typical properties of an IGP that would have been designed from scratch. This approach helps to better match the network requirements by fully exploiting the available resources and simplify both its design and management. Unfortunately, a heavy reconfiguration campaign is often necessary, since the two IGPs are likely to implement very different design approaches and guidelines. Re-designing the merged IGP may also involve modifications like reshaping parts of the network, decommissioning redundant nodes, adding or removing links, etc.

To make the most of a network merging, seamless approaches to complete network merging are highly desirable in practice. However, just as for regular reconfiguration scenarios, network operators lack methodologies and tools to plan and carry a disruption-free merging process. Therefore, each company facing a merging process has to elaborate its own method-

ology without guarantees on the impact it will have on traffic [172]. Consequently, severe traffic disruptions can arise during the merging, possibly resulting in the violations of Service Level Agreements of every involved networks.

In the following, we study the problem of merging networks from a routing point of view. Our aim is to guarantee *the absence of packet losses* throughout the entire merging process. We first define the network merging problem. Then, we identify the main building blocks that are required to enable seamless network merging. We also describe how to implement them by extending the reconfiguration techniques described in Section 3.5. Finally, we illustrate the validity of our approach trough simulation.

The network merging problem

We formulate the *seamless merging problem* as follows: Given n initial networks, the associated IGP configurations $(I_1, \ldots, I_n)$, and the corresponding next-hop functions $(nh_1, \ldots, nh_n)$. Install the final IGP configuration I_{fin} and the resulting next-hop function nh_{fin} in the network resulting from the merging of these n initial networks. We say that a merging process is seamless if no forwarding loop is experienced.

Observe that, even if the physical topology may change during the merging process, we only consider changes in the IGPs since forwarding is determined by the IGPs. We also consider that traffic flows in the merged network (i.e., nh_{fin}) are given as input by the network operators. Indeed, we assume that operators compute traffic flows according to some capacity planning tool and design best practices. Computing new traffic flows and optimizing the merged topology for high-level network objectives is out of the scope of this thesis.

A unified routing for a unified network

To enable seamless IGP merging, we describe a procedure composed of three main logical steps. Each step is composed of some abstract operations which we call *building blocks*.

- ***Step 1*: Render the networks to be merged compatible**. In this step, networks to be merged are kept separated, but they are prepared for the merging operation by changing and tuning the IGPs running on them. Traffic flows do not change. We distinguish two sub-steps.
  - **Compatible protocols and selection process**. If needed, the protocol running in each initial network is changed to match the routing protocol installed in I_{fin}. For example, if I_{fin} is running IS-IS, OSPF routing instances are moved to IS-IS. In addition to the protocol, the route selection process of I_{fin} (i.e. flat or hierarchical) is imposed on all the networks to be merged.
  - **Compatible IGP weights**. IGP weights are made compatible accross routing instances (equivalent ranges).

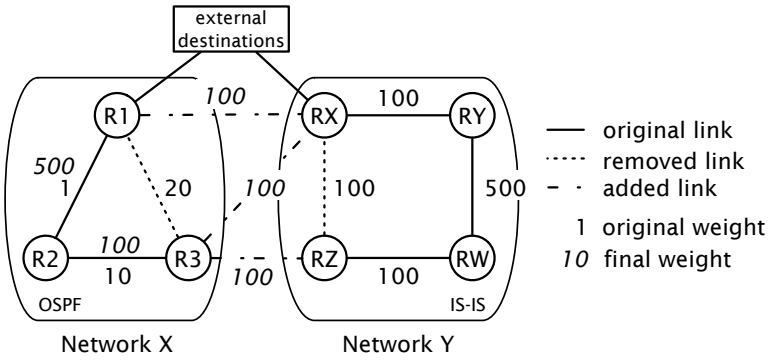


Figure 3.16 Merging scenario in which network X and Y are merged together

- ***Step 2*: Connect and reshape logical graphs**. The logical graphs are merged into the final logical graph in I_{fin}. Observe that the final logical graph can differ from the union of the logical graphs built by the routing instances to be merged. Indeed, adjacencies can be added, replaced, or removed to any of the routing instances to be merged. As a result, a single routing instance is generated starting from several ones.
- ***Step 3*: Tweak IGP weights in the merged network**. If needed, the IGP weights are changed so as to match I_{fin}.

We now describe how to implement these building blocks. An example of network merging between networks X and Y is depicted in Fig. 3.16. In both networks, a link-state IGP is used to propagate routing information for both intradomain and interdomain destinations. $R1$ and RX act as gateway to reach external destinations by redistributing a default route within the IGP. When several networks merge together, it is unlikely that they use the same IGP or, if they do, the same logical organization. For instance, a flat OSPF can be deployed in X, while Y can use a hierarchical IS-IS configuration. Even when the IGPs to be merged share a similar physical design and the same logical organization (e.g. both networks use a flat IGP), the weight ranges configured on the links could be very different as IGP weights are often used to implement traffic engineering objectives [50]. By analyzing three real-world large-size networks, we confirmed that the weight ranges are quite different, ranging from $[1:2{,}000]$, to $[1:10{,}000]$ and $[1:100{,}000]$. In Fig. 3.16, X uses weights ranging from 1 to 20, while the link weights configured in Y belong to the interval $[1:500]$. Moreover, we assume that the final weight range is $[1:500]$, as it would probably be if Y acquired X. Final weights, if changed from the initial weights, are in italics in the figure. In addition to the modified link weights, Figure 3.16 highlights links that are added and removed during the merging process. In particular, three links are added: $(R1, RX)$, $(R3, RX)$ and $(R3, RZ)$, while two links are removed: $(R1, R3)$ and (RX, RZ).

We combine two techniques to implement the three merging steps. First, we leverage the reconfiguration techniques developed earlier (Section 3.5) which enables IGP routing instances to be seamlessly modified. Second, we develop reweighting techniques for the computation of IGP weights given a set of constraints (e.g., range of weights or traffic flows to implement). The reconfiguration techniques implement two building blocks: *compatible protocols and selection process*, and *connect and reshape of logical graphs*. The reweighting technique implements the remaining two building blocks: *compatible IGP weights* and *tweak IGP weights in the merged network*.

Regarding the reconfiguration techniques, we extend them to support the reshaping of the IGP topology. To support the addition of a link in the final IGP, we assign an infinite weight to this link in the initial IGP while, in the target IGP, the link weight is set to its final weight. The opposite holds for removing a link. The final link weight is set to infinity while there is no weight change in the initial IGP. After the convergence of the IGPs, we obtain the initial and final next-hop functions. We are then able to compute an appropriate ordering in which to migrate routers by simply applying the Loop Enumeration Algorithm (Fig. 3.5) or the Routing Tree Heuristic (Fig. 3.7). By combining link additions and removals, any reshaping of the IGP topology is supported, including the addition and the removal of a node.

We also extend the reconfiguration techniques to avoid the creation of interdomain loops. Indeed, in addition to intradomain loops, interdomain loops can arise during the merging process. Consider again Fig. 3.16. Before merging, $R1$ is reaching $R3$ via $R2$. While, after merging, $R1$ is reaching $R3$ via RX, as the direct link to $R3$ is removed. Hence, if $R1$ is migrated before RX, $R1$ starts forwarding traffic to RX before RX being able to use the newly added links. Therefore, RX will send the traffic on its default route. Eventually, this traffic will reach $R1$, effectively creating an interdomain loop. To prevent interdomain loops, we model the Internet connectivity with a virtual node. The virtual node can be used by routers of one network to reach external destinations including the destinations in another network to be merged (before new links are added to connect them). By considering the virtual node as a normal node, our algorithms automatically compute a loop-free ordering encompassing Internet destinations.

Regarding the *reweighting* techniques, we formulate the problem of identifying a set of link weights, compliant with given objectives, as an Integer Linear Program, in which each variable represents the weight associated to a link. In order to set up *compatible routing policies* (step 1 of the merging process) in the networks to be merged, we adopted the same approach proposed in [31], and we modified the Dijkstra algorithm to identify a set of linear constraints sufficient to preserve the All-Pairs Shortest Path (APSPT) on the graph to be reweighted. We aim at preserving the APSPT because, at this step of the merging process, we only need to impose the same range of weights on all the IGPs to be merged, without changing the traffic flows.

We extended the technique described in [31] for (i) supporting integer weights only, (ii) reweighting the graph by minimizing the weights being used and, (iii) reweighting the graph using values in a given candidate set. These improvements are aimed at preserving the semantics that operators associate to different link weights (e.g., primary, backup, etc.). Keeping this semantic in the merged network is a relevant objective as link weights are often used by network operators to reason about traffic flows. In particular, providing constraints on weights that are configured is desirable in case of a network acquisition, where it is preferable to use the range of weights of the acquiring network in the final merged network.

While our reweighting approach is useful to maintain the traffic flows, it can also be used to tweak the routing policies in the merged network, in the last step of the merging process. Indeed, traffic flow objectives in the merged network can be translated into additional linear constraints. To compute these additional constraints, well-known IP traffic engineering tools can be used [50].

The full IGP merging procedure follows. Routing instances to be merged are made compatible and linked together (step 1 and 2) by using the reconfiguration techniques. In particular, the final protocol is deployed in each of the networks to be merged. The configuration of the routing instance in each network is the same as the final one, except for link weights. Indeed, link weights used in this step are tuned by the reweighting technique, such that the initial APSPT is preserved, and the same range of values is used. Links to be added in the merged network are configured with an infinite weight in the initial protocol.

By using the reconfiguration technique, the initial protocol is progressively replaced by the final one in all the routing instances. As for links to be added, links to be removed in the merged network are configured with an infinite weight in the final protocol. If needed, new routing policies are then imposed in the merged network, by computing the new weights with the reweighting technique and seamlessly changing them using the reconfiguration technique. Once the process is over the initial configuration can be safely removed from all the networks as it is not used anymore.

Evaluation of the merging techniques

We now evaluate the merging techniques and show their practicality. Since we already evaluated the reconfiguring techniques in Section 3.7, we focus on the reweighting techniques. Our data set is composed of the IGP topologies of five networks, namely, Abilene, Geant, and Network1, Network2 and Network3. IGP topologies of Network1, Network2 and Network3 have been privately provided by network operators, while the others are publicly available. Table 3.2 provides more details about our dataset. Notice that the number of IGP weights being used is relatively higher in Abilene and Geant as both of them define their weights based on the measured delay between the routers.

Network	# of nodes	# of links	# of distinct weights
Geant	22	68	16
Abilene	9	26	13
Network1	> 550	> 1300	33
Network2	> 150	> 600	54
Network3	> 150	> 350	20

Table 3.2 Dataset characteristics

We evaluated the performance and the scalability of the reweighting technique by considering the merging of each pair of networks in our dataset. To simulate the merging of two networks, we interconnected them with a number of links proportional to the square root of the number of nodes in the smallest network. We further assigned to each additional link a random weight ranging between 1 and the maximum weight used in both networks. Finally, we reweighted each merged network preserving the APSPT. Notice that more complex scenarios in which operators want to change traffic flows in the merged network are possible, but we expect that these scenarios will not have a big impact on the performance of the reweighting technique.

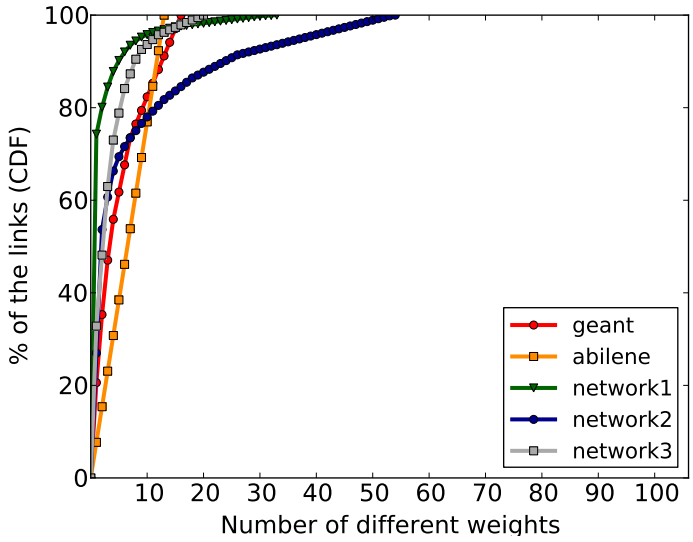


Figure 3.17 A few number of weights account for the majority of the links.

Figure 3.17 shows the cumulative distribution function of the number of links sharing the same weight. In the three real-world networks, 70% of the links shares 5 weights or less. These few number of weights are likely to identify the main classes of links (e.g., primary, backup, access, etc.). Observe that, since our reweighting technique relies on LP, it allows op-

erators to statically define the translated weights for some initial weights. This would help in preserving the semantic in different weight assignment.

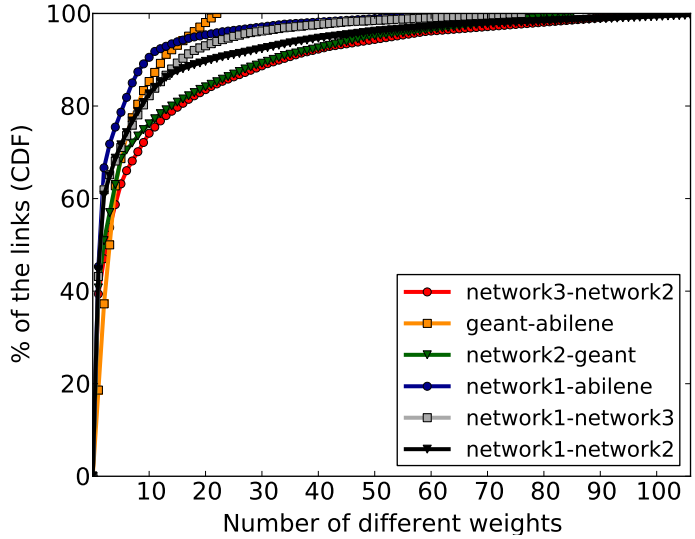


Figure 3.18 Translated weights exhibit similar distribution of the original ones.

Figure 3.18 shows the cumulative distribution function of the number of links sharing the same weight in the merged and reweighted networks. Even if more weights are typically needed after the merging process, the total number of weights is still manageable. More importantly, distribution of final weights is similar to that of weights in the original networks. Thus, by limiting the number of translated weights, the reweighting technique is able to keep most of the semantics associated to the links.

Figure 3.19 plots the median (over 30 separated runs) of the computation time taken by the reweighting technique in function of the number of links in the merged network. All the tests were run on a Sun Fire X2250 (quad-core 3GHz CPUs with 32GB of RAM) using Gurobi [161] as linear solver. In all the cases, the reweighting process was able to find a new weight assignments and took less than 2 minutes to complete. As a second group of experiments, we fixed the *candidate weights*, i.e., a set of weights that should be preferably used in the translation process and we measured the computation time when increasing the number of candidate weights. On average, reweighting is faster when fixing the candidate weights. Indeed, additional constraints are given to the LP solver, thus decreasing the size of the search space, at the cost of increasing the risk for not finding a weight assignment. Also, we found that the number of candidate weights did not impact the solving time.

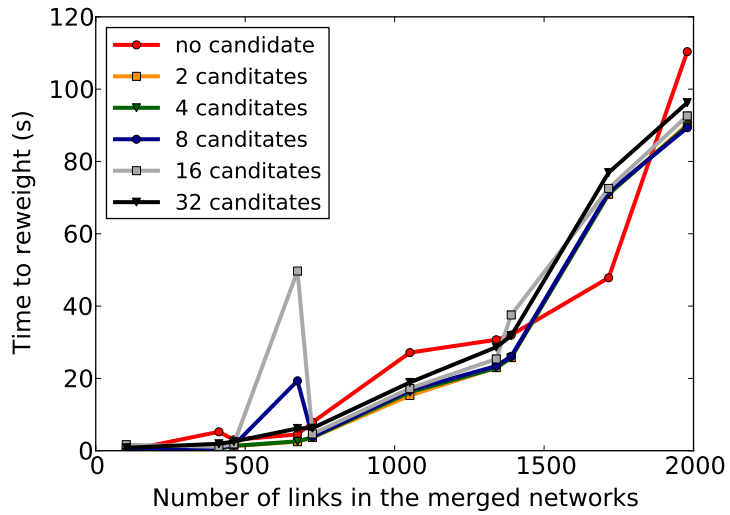


Figure 3.19 All networks can be reweighted in less than 2 minutes.

3.11 Related work

Seamless IGP operation and maintenance have been the focus of several previous studies. For example, several protocol extensions have been proposed [123, 93, 125] to gracefully restart a routing process. However, few research effort has been specifically devoted to network-wide seamless IGP migrations, and current best practices [191, 68] are only rules of thumb which do not apply in the general case, and do not guarantee lossless reconfiguration processes.

In [116] and [115], 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. They 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., [74]).

In [77], Keralapura *et al.* study the problem of finding the optimal way in which devices and links can be added to a network to minimize disrup-

tions. Beyond addressing topological changes, our techniques can be used to address several other migration scenarios.

In [112], Rastogi *et al.* study the problems of (i) selecting the aggregates to advertise at each ZBR and (ii) assigning weights to these aggregates so as to to minimize the error in the shortest path computations. The authors describe an optimal dynamic programming algorithm for the first problem and describe a randomized search strategy for the second problem after having proved that it is $\mathcal{NP}$-hard in the general case. When introducing summarization in an IGP, their techniques could be used in complement of ours as they allow to limit the amount of changes between the initial and the final next-hop functions. Limiting the amount of changes in the next-hop functions can help speeding-up the computation time of finding an ordering or even allow to find one if none existed originally.

In [27], 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 hold in general. For example, migrating $E1$ before $B1$ in Fig. 3.1 creates a forwarding loop in a $hier2flat$ scenario.

In [4], 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 as it requires to duplicate the entire control-plane.

Recently, some techniques [151, 75] 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 [117], 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 [56]. Fu *et al.* [57] and Shi *et al.* [126] 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 [84], we believe that relying on a ships-in-the-night approach (when possible) makes the entire migration process easier and more manageable.

3.12 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 strategy that enables operators to perform network-wide changes on an IGP configuration seamlessly, rapidly, and without compromising routing stability. Our strategy 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.

In addition to support for regular reconfiguration scenarios, we explained how to extend the reconfiguration techniques to support two relevant advanced scenarios: (i) the case of reconfiguring an IGP in presence of single-link failure and (ii) the case of merging two IGPs together. While our techniques do not currently minimize link congestion, it can be avoided with similar extensions, i.e., adding constraints to the migration ordering research space. Our vision is that network-wide migrations could become a basic operation enabling the seamless replacement or reconfiguration of any network protocol. To that extent, we believe that the development of general reconfiguration protocol is an interesting open problem raised by this work.

Chapter 4

Towards disruption-free distance-vector to link-state IGP reconfiguration

4.1 Introduction

While DV protocols have been progressively abandoned by major ISPs [52, 141], they are still heavily used by enterprise networks [88]. We believe one explanation of this situation is that migrating from a DV to a LS can be disruptive. Indeed, while forwarding loops cannot appear during the reconfiguration process, traffic shifts can appear in every intermediate step (see Chapter 2). Traffic shifts can be disruptive as they increase the likelihood of abruptly raising the traffic sent along a path. Such traffic changes, in turn, can lead to network congestion and therefore traffic losses when an under-provisioned path starts to be used. Moreover, slight variations of the IGP can negatively impact overlaying protocols such as BGP [130].

In this chapter, we develop reconfiguration techniques that limit these traffic shifts during ships-in-the-night reconfigurations. Our contribution is threefold. First, we show that a significant amount of traffic shifts can occur when DV to LS reconfigurations are naively performed. These traffic shifts can happen even when the initial and the final forwarding paths are perfectly equivalent. Second, we describe a sufficient and necessary condition to avoid traffic shifts when a per-destination ordering is followed. Using this condition, we prove that there always exist a per-destination ordering free from traffic shift. Third, since reconfiguring on a per-destination basis is not practical, we propose a simple heuristic to limit the traffic shifts when a per-router ordering is followed. Our heuristic is efficient even if it is not guaranteed to satisfy the sufficient and necessary condition. Indeed, it avoids the majority of the traffic shifts in all the tested scenarios.

The rest of the chapter is organized as follows. Section 4.2 quantifies the amount of traffic shifts that can be created when performing DV to

LS reconfiguration scenarios. Section 4.3 describes several reconfiguration strategies to lower and even completely avoid these traffic shifts. Finally, Section 4.4 concludes the chapter.

4.2 Quantifying reconfiguration problems

In this section, we study the amount of traffic shifts that can appear in typical DV to LS reconfiguration scenarios. For the sake of simplicity, we consider reconfigurations scenarios in which the logical graph and the link weights are equal in the DV and the LS IGPs. Consequently, routers FIB entries are the same in the initial and in the final configuration.

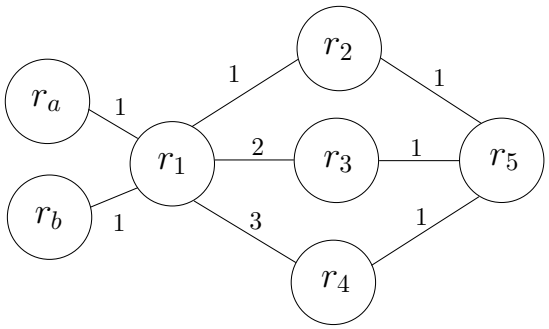


Figure 4.1 In this simple reconfiguration scenario, up to 36 traffic shifts can be created when replacing a DV IGP by the corresponding LS IGP.

Fig. 4.1 depicts a DV to LS reconfiguration where traffic shifts are likely to appear. Consider router r_1 and destination r_5. In both the initial and in the final IGPs, r_1 uses r_2 to reach r_5. If r_2 is the first router to be migrated, it will stop announcing the destination r_5 to r_1 in the DV as it now uses the LS route to reach r_5. Since r_1 is not migrated and still learns the destination r_5 via r_3 and r_4, it will now send its packets towards r_3 (the shortest remaining DV path). Similar considerations apply if r_3 is migrated next, since $r1$ will then shift to $r4$. Finally, when r_4 is migrated, r_1 does not receive any DV path to r_5 and will start using its final LS path via r_2. Two traffic shifts have thus been generated. Naive reconfiguration strategies are likely to generate a lot of traffic shifts. In this simple example, following a random ordering creates 22 traffic shifts on average and 36 traffic shifts in the worst case. In contrast, no traffic shift occurs towards any destination if the order (r_b r_a r_1 r_3 r_4 r_2 r_5) is followed.

To evaluate the amount of traffic shifts created in DV to LS migrations, we simulated reconfiguration scenarios using both publicly available topologies and a commercial network topology. Regarding publicly available topologies, we considered the inferred topologies provided by the Rocketfuel project [129]. In the following, we report the results on AS1221 (104 nodes, 302 edges) and AS3967 (79 nodes, 294 edges), which

respectively represent the best and the worst case in terms of the number of traffic shifts. Similar results and considerations hold for the other topologies. The commercial topology contains approximately 150 nodes and more than 400 edges. On this data set, we performed several experiments each time considering a different router reconfiguration ordering. Each router is reconfigured following SITN migration by increasing the AD of the DV protocol above the AD of the LS protocol, as best current practices dictate [191, 69, 73]. We repeated each experiment 30 times.

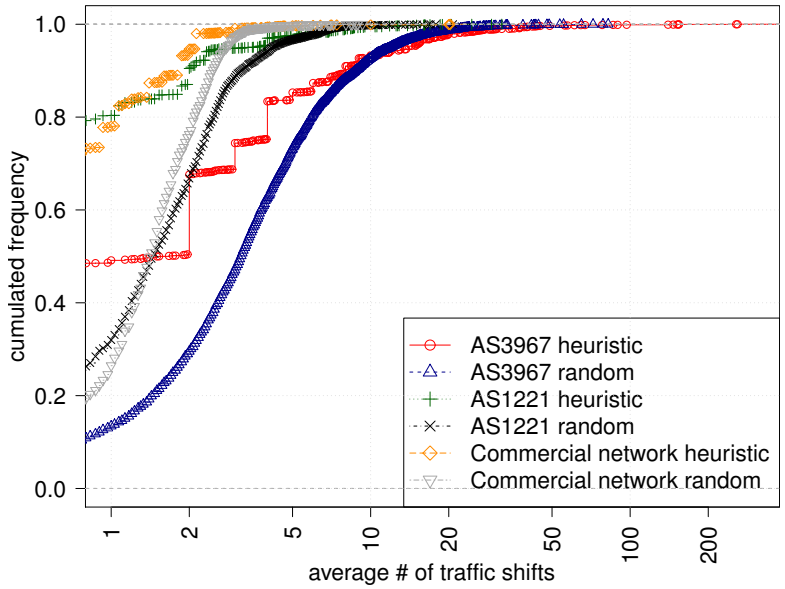


Figure 4.2 Cumulative distribution function (CDF) of the average amount of traffic shifts created between each pair of routers when performing the migration by increasing the AD of the initial protocol.

Fig. 4.2 shows, for each pair of routers, the average amount of traffic shifts created during the reconfiguration process expressed as a CDF (curves labeled random). For AS1221 (resp. AS3967), only 25% (resp. 10%) of the pairs of routers do not experience traffic shifts, while 50% of them experience more than 1.5 (resp. 3.2) traffic shifts during the migration. Similar results hold in the commercial network where only 20% of the pairs do not experience traffic shifts. Each traffic shift increases the likelihood of abruptly raising the traffic sent along a path leading to network congestion when an under-provisioned path starts to be used.

4.3 Limiting routing anomalies

In this section, we describe how the insights provided by our theoretical classification described in Chapter 2 can be leveraged to limit traffic shifts in DV to LS reconfigurations. We work in two steps. First, we

present a necessary and sufficient condition for a reconfiguration free from traffic shifts, and proof that a per-destination ordering free from traffic shift always exists. Second, since per-destination reconfigurations are time-consuming and cumbersome, we devise a heuristic to compute a per-router ordering. We evaluate our heuristic and show that it avoids the majority of the traffic shifts in all the tested scenarios.

In our formal analysis of the DV to LS scenario, a fundamental role is played by the DV-cone concept (Section 2.3). The DV-cone for a destination d at time t corresponds to all the routers that use the DV IGP p to reach the destination d at time t. During a DV to LS reconfiguration, the DV-cone progressively shrinks for each destination: in the initial routing state, the DV-cone of every destination includes all the routers in the network, while in the final state it is empty. Reconfiguring a single router r removes it from the DV-cone of several destinations. In addition, reconfiguring r could remove other routers r' from the DV-cone of several destinations, if r' has no other neighbor providing it with a route in the DV protocol. For instance, r_1 is the only router providing a DV route for $r_2, \ldots, r_5$ to r_a and r_b in the network depicted in Fig. 4.1. Therefore, migrating r_1 forces r_a and r_b to also use a LS path to reach $r_2, \ldots, r_5$ since they do not learn a DV path anymore. That is, migrating r_1 also removes r_a and r_b from the DV-cone towards $r_2, \ldots, r_5$.

From Lemma 2.1, we know that the actual paths function π in every DV-cone forms a tree (see Chapter 2). Hence, no traffic shift occurs if the router reconfiguration order is consistent with the topological order implied by the tree structures of all DV-cones. This condition is actually a necessary and sufficient condition (Theorem 4.1).

Theorem 4.1. *Let p be a DV IGP and $\bar{p}$ be an LS IGP such that the shortest path trees computed by each protocol are equal. Given any destination d, no traffic shift occurs during a reconfiguration from p to $\bar{p}$* if and only if *each router r exits the DV-cone of p before any of its successors in $\pi(r, d, 0)$.*

Proof. First, we show that if a router x exits the DV-cone of p after one router y in $\pi(x, d, 0)$, then we have a traffic shift. Let t_x and $t_y < t_x$ be the time at which x and y exits the DV-cone, respectively. For each time t such that $t_y < t < t_x$, x is in $cone(p, d, t)$, hence still uses p. By Lemma 2.1, $used(x, t) = p$, and $\pi(x, d, t)$ is completely internal to the DV-cone. Since y is not in $cone(p, d, t)$ anymore, it is not using p. Hence, $\pi(x, d, t) \neq \pi(x, d, 0)$ and $\pi(x, d, t) \neq \pi(x, d, f)$ which corresponds to a traffic shift.

We now prove that if routers are migrated in a consistent order with respect to the topological order imposed by the DV-cones (i.e., *before* any of its successors, for all the destinations), then no traffic shift occurs. Consider that router x is migrated for the destination d at time t. The actual path $\pi(x, d, t)$ can be written as PQ, where P is a (possibly empty) path outside the DV-cone and $Q = (y \ldots d)$ is a path inside the DV-cone. By hypothesis, P must be a sub-path of $\pi(x, d, f)$. Also by hypothesis, all routers in the $\pi(y, d, 0)$ are still in the DV-cone of d, hence Lemma 2.1

implies that $Q = \pi(y, d, 0)$. Since the initial and final paths are equal, $PQ = \pi(x, d, f)$, yielding the statement. □

Corollary 4.1. *Given a destination d, if each router is reconfigured before all the routers in its actual path to d, then no traffic shift occurs for d.*

Corollary 4.1 states that if routers are reconfigured on a per-destination basis, then it is always possible to reconfigure the network without traffic shifts. Intuitively, the corollary guarantees that whenever a router is reconfigured, the final LS forwarding path is used as all its successors have already been reconfigured.

While following a per-destination ordering can guarantee the absence of traffic shifts, it also makes the whole process cumbersome. Moreover, the network runs in a transient state for a much longer time. An obvious way to speedup the reconfiguration is to follow a per-router ordering which is recommended by best current practices [73, 191]. However, the price to pay is the loss of guaranteed correctness. Indeed, a per-router ordering might not always exist as ordering constraints for different destinations can be contradictory with respect to each other. As illustration, an example of network in which no per-router ordering exists is depicted in Fig 4.3. Since migrating r_5 and r_6 does not create any traffic shift, we consider the state of the network at time t where both of them have been migrated. At time t, reconfiguring any of r_1, r_2, r_3 or r_4 creates at least one traffic shift. Indeed, reconfiguring r_1 creates traffic shifts for r_5 as routers start to use the path provided by r_3. The same holds when reconfiguring r_4 considering as destination r_6. Similarly, reconfiguring r_2 creates traffic shifts for r_3 since r_1 will start using its direct link with r_3 to reach it. The same holds when migrating r_3 considering as destination r_2.

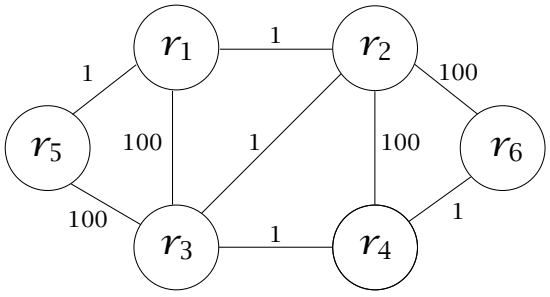


Figure 4.3 Shifty gadget. In this DV2LS reconfiguration scenario, no per-router ordering is free from traffic shifts. In particular, migrating any of r_1, r_2, r_3 or r_4 creates at least one traffic shift.

In addition to not always exist, a per-router ordering is not easily computable since the necessary and sufficient constraints identified from Theorem 4.1 are likely to be contradictory. Hence, we propose the Minimal Traverse heuristic defined in Fig. 4.4. The heuristic iteratively computes a router reconfiguration ordering by picking one router at the time. At each

iteration, the heuristic tries to select the router with the lowest likelihood of currently providing DV routes to other non-reconfigured routers.

```
minimal_traverse_heuristic(G = (V, E))
ordering ← [ ]
H = (V', E') ← G
while V' ≠ ∅ do
    // min_traverse_node(H) returns one of the node
    // of H which is involved in the least amount of shortest paths
    u ← min_traverse_node(H)
    append(ordering, u)
    H ← remove_node(u, H)
end while
return ordering
```

Figure 4.4 Minimal Traverse heuristic.

The intuition behind this heuristic directly follows Theorem 4.1. However, the heuristic is not guaranteed to satisfy the condition expressed in Theorem 4.1 since there is no one-to-one mapping between the number of DV route traversing a router r and the number of routers to which r provides a DV route.

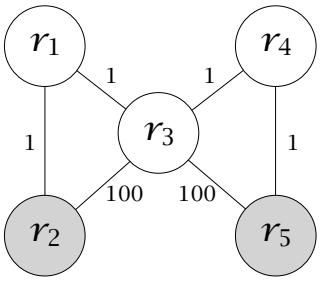


Figure 4.5 Unpopular gadget. In this DV2LS reconfiguration scenario, the Minimal Traverse heuristic is not able to find a per-router ordering free from traffic shifts while there exists one (i.e., $r_2\ r_5\ r_3\ r_1\ r_4$).

To illustrate the sub-optimality of the heuristic, consider the reconfiguration scenario depicted in Fig 4.5. In this network, the heuristic fails to find an ordering free from traffic shifts, while one exists: $(r_2\ r_5\ r_3\ r_1\ r_4)$. Indeed, the heuristic computes the following ordering: $((r_2|r_5), (r_1|r_4), r_3)$ by ranking the nodes according to the number of paths they are involved in. While migrating r_2 and r_5 is perfectly safe, migrating r_1 (resp. r_4) creates a traffic shift for destination r_2 (resp. r_5) as r_3 starts using its direct path instead of the detour path. Although r_3 is the most heavily used router, no traffic shift is created when it is migrated even if it is the very first router to be migrated. The reason is that r_3 is a graph separator for the DV logical graph. A *graph separator* is a set of graph vertices whose removal divides the graph into two or more connected components. During a

DV to LS migration, migrating the routers forming a graph separator is interesting as routers located in the resulting connected components use the LS path to reach each other. Therefore, it helps speeding up the reconfiguration process by transitioning multiple nodes to their final paths at once. As another illustration, consider again the network depicted in Fig. 4.1. In this network, r_1 and r_5 form a graph separator. Indeed, removing r_1 and r_5 implies the DV graph to be completely disconnected, i.e., each resulting connected component is composed of only one node. Therefore, each node is using its final path to reach all the destinations after the migration of only two nodes.

Even if the heuristic is not guaranteed to find a traffic shift ordering when one exists, it is polynomial with respect to the size of the graph and performs well (see Fig. 4.2). For AS1221, 73% (resp. 99%) of the pairs experience 0 (resp. 8) traffic shifts. Similarly, for AS3967, 47% (resp. 99%) of the pairs experience 0 (resp. less than 34) traffic shifts. For the commercial network, 70% (resp. 99%) of the pairs experience 0 (resp. less than 3) traffic shifts.

While the orderings computed by the heuristic are clearly better than random ones with respect to the amount of traffic shifts created, some pairs of routers are still subject to a significant number of them. To further improve the performance, we refined our approach by slightly deviating from the traditional SITN technique. In particular, we evaluated the performance of the heuristic when the reconfiguration of a router consists in completely removing its support for the DV protocol instead of just tweaking the AD values. Removing the DV is safe since the final protocol is LS. Therefore, no route loss is ensured as a LS is guaranteed to always provide a route to all the routers in the network.

While the traditional SITN approach allows network operators to backtrack to the initial state in case of problems happening during the migration, the modified approach is less prone to traffic shifts. Indeed, when a router r is reconfigured by increasing the AD value assigned to the DV IGP, r is removed from the DV-cone of all destinations, except r itself. Conversely, when a router r is reconfigured by removing the DV completely, r is removed from the DV-cone of all destinations, *including* r itself. By Theorem 4.1, it is ensured that no traffic shift can occur anymore for traffic destined to r.

While removing the DV is better from the traffic shifts point of view, it could create more convergence issues since removing the protocol on a router is equivalent to a router failure. Depending on the DV protocol, such a failure can result in a count-to-infinity problem. However, EIGRP—the DV IGP used in the majority of enterprise networks [88]—does not suffer from such a problem [59].

Fig. 4.6 reports the improvement obtained by removing the DV protocol in the reconfiguration of each router. For AS1221, 94% (resp. 99%) of the pairs experience 0 (resp. 2) traffic shifts. Similarly, for AS3967, 72% (resp. 99) of the pairs experience 0 (resp. 17) traffic shifts. In the commercial network, almost all the possible traffic shifts are avoided since 98% of the

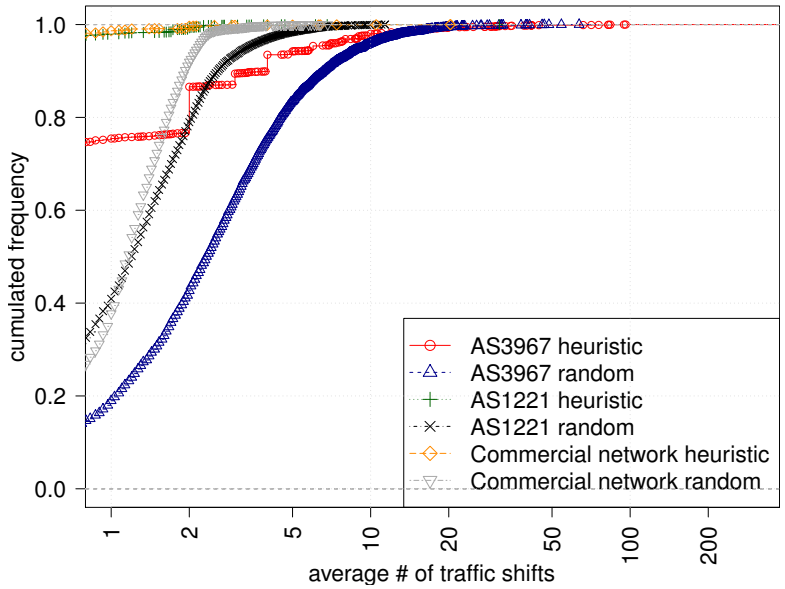


Figure 4.6 CDF of the average amount of traffic shifts created between each pair of routers when performing the migration by removing the initial protocol. Results for other scenarios are similar.

pairs do not experience any. The number of traffic shifts is similar when the DV IGP is not removed (see Fig. 4.2) and a random ordering is followed.

Even when performing the migration by removing the DV, a few pairs of routers still experience a large number of traffic shifts as exacerbated by the tail of the distributions. To avoid those traffic shifts, a per-destination ordering could be enforced only for these problematic pairs. Such a hybrid reconfiguration strategy effectively combines the guaranteed correctness of the per-destination ordering with the speed and management ease of a per-router ordering.

4.4 Conclusions

Despite being prone to numerous problems, DV IGPs are still heavily used today. One explanation is that transitioning a DV IGP to a LS IGP can be disruptive as traffic shifts can appear during any reconfiguration step. Indeed, as our simulations show, numerous traffic shifts do happen when naive reconfiguration strategies are followed.

To limit the occurrence of traffic shifts during the reconfiguration, we provide network operators with two different reconfiguration strategies. The first strategy consists in performing the migration on a per-destination basis. Although cumbersome from the network management point of view, we proved that in per-destination reconfiguration there always exists a reconfiguration ordering which avoids all traffic shifts. The

second strategy consists in performing the migration on a per-router basis. This strategy trades correctness for practical effectiveness. Indeed, a per-router ordering does not always exist and is harder to compute. To cope with this situation, we designed a heuristic and confirmed its practical effectiveness on both inferred and real network topologies.

Regarding the perspectives, quite a lot of work remains to be done. First, we would like to study the computational complexity of deciding whether a per-router ordering exists. Also, we would like to leverage the notion of graph separator in order to speed up the reconfiguration process. We also believe that extending the framework to encompass formal analyses of transient issues is an interesting open problem. Finally, we would like to study whether the concepts of this chapter could also limit the traffic shifts in LS to LS scenarios.

Part III

Reconfiguring interdomain routing protocols

Chapter 5

iBGP configuration correctness

5.1 Introduction

To reconfigure an iBGP configuration such that it remains correct in every intermediate state, it is first needed to define what a correct configuration is. This is the main goal of the chapter. In particular, we define iBGP configuration correctness in presence of route reflection, the iBGP scaling mechanism used in most BGP networks.

With respect to an iBGP full-mesh, route reflection trades scalability for correctness as it is prone to both routing and forwarding anomalies. These anomalies are due to reduced visibility and the interaction between iBGP and the underlying IGP [66]. Routing anomalies consist in routing oscillations that prevent iBGP from settling to a stable state. Forwarding anomalies consist in packet deflections in which different routers choose different egress points for the same destination prefix. By combining multiple packet deflections, forwarding loops can be created.

In the last decade, the research community has devoted significant effort to prevent such anomalies from happening. Griffin and Wilfong [66] formalized the absence of routing and forwarding anomalies by introducing two fundamental properties of iBGP configurations: signaling and forwarding correctness. *Signaling correctness* ensures that a BGP network will always converge to a stable routing state. *Forwarding correctness* ensures the absence of packet deflections along any forwarding path. More recently, several authors proposed solutions to guarantee signaling and forwarding correctness, either by enforcing special properties on the iBGP configuration (e.g., [114, 17, 18]) or by modifying the iBGP protocol itself (e.g., [49, 95]).

In this chapter, we show that iBGP route propagation rules also play a fundamental role in ensuring the correctness of iBGP configurations in the presence of route reflection. We give simple examples in which traffic blackholes can be created by the combined effect of iBGP route propagation rules and the iBGP route selection algorithm. Our examples show that distinct destination prefixes cannot always be analyzed separately. To model the absence of propagation anomalies, we define a new correctness

property, called *dissemination correctness* and show how it fills the gap between signaling and forwarding correctness. Since we find that verifying dissemination correctness is computationally intractable, we propose sufficient conditions to enforce it and use them to provide network operators with iBGP design guidelines. In particular, we find that the absence of a special type of iBGP sessions named *spurious OVER* sessions guarantees dissemination correctness. Although uncommon, spurious OVER sessions are sometimes added in real-world networks [44, 102, 193, 179]. Indeed, they can be used by network operators to fix forwarding issues and improve route diversity, as suggested in some recent research works (e.g., [104, 105]). Spurious sessions are also likely to appear during the reconfiguration process. Trough a thorough review of the state of the art, we show that most previous work overlook dissemination correctness, incorrectly assuming that signaling correctness implies dissemination correctness.

This chapter is organized as follows. Section 5.2 introduces the model and the notations. Section 5.3 reviews the known correctness properties as well as the known sufficient conditions to enforce them. Section 5.4 illustrates the importance of the iBGP propagation rules. Section 5.5 formally defines the dissemination correctness property. Section 5.6 proposes two sufficient conditions for enforcing dissemination correctness. Section 5.7 revises related work. Finally, Section 5.8 ends the chapter.

5.2 A model for iBGP configuration

We now present a tailored version of the well-known "Stable Paths Problem" (SPP) model [62] that we use to model iBGP topologies in the rest of the thesis.

To model an IGP (logical) graph, we reuse the model described in Chapter 2 (see Section 2.2). In this model, an IGP graph is a weighted directed graph $I = (V, E)$ with a weight associated to each edge $(u, v) \in E$. Without loss of generality, we assume that the IGP graph is symmetric. 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, $u \leftarrow v \Leftrightarrow v \rightarrow u$, and $u \leftrightarrow v \Leftrightarrow v \leftrightarrow u$. Route reflection topologies are usually organized in a hierarchy where there are no cycles consisting of UP sessions only. In a iBGP 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-reflector.

Due to the iBGP route propagation rules, not every path can be used to distribute a BGP route announcement. 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. This means that a valid signaling path matches regular expression UP*OVER?DOWN* [18]. Formally, we denote with $\mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u$ the set of valid signaling paths, and $\forall u \in V$, $\mathcal{P}^u$ is the set of valid signaling paths starting at u and ending at an egress point, i.e., a border router of the ISP. The empty path ϵ represents destination unreachability, and is always valid at any vertex in V.

For each $u \in V$, the preference level of paths in $\mathcal{P}^u$ is expressed by a *ranking function* $\lambda^u : \mathcal{P}^u \to \mathbb{N}$. 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 $\Lambda = \{\lambda^u | u \in V\}$. For each router $u \in V$, λ^u completely reflects the BGP decision process (see Table 1.1). As in SPP, unreachability is the last resort, that is, $\forall P \in \mathcal{P}^u, P \neq \epsilon$: $\lambda^u(P) < \lambda^u(\epsilon)$. Also, λ^u respects IGP metrics, hence $\forall P_1, P_2 \in \mathcal{P}^u$, $P_1 = (\ldots e_1)$, $P_2 = (\ldots e_2)$, then $dist(u, e_1) < dist(u, e_2) \Rightarrow \lambda^u(P_1) < \lambda^u(P_2)$.

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 5.5 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).

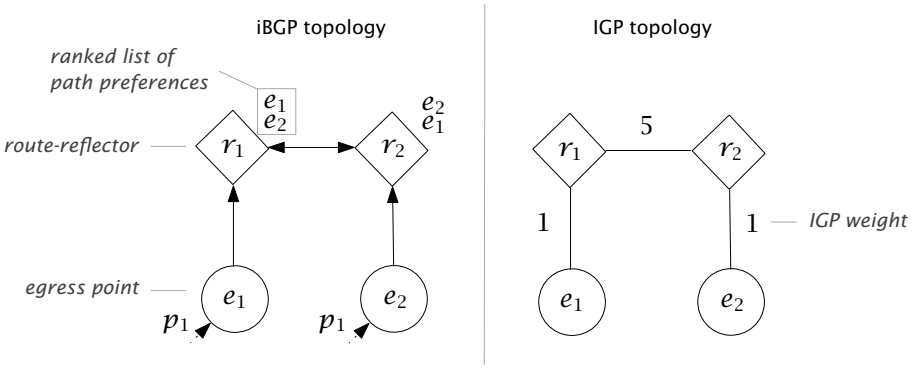


Figure 5.1 Graphical notations used in the thesis when referring to an iBGP configuration

We now describe the graphical conventions used in the thesis when referring to iBGP configurations (see Fig. 5.1 for an example). 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 valid signaling 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. 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. For instance, in Fig. 5.1, we denote with the list $(e_1\ e_2)$ the fact that r_1 prefers the path $(r_1\ e_1)$ over $(r_1\ r_2\ e_2)$. Moreover, 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. Regarding the IGP topology, lines represent IGP links and labels represent the IGP weight assigned to a link.

5.3 Known correctness properties and sufficient conditions

In this section, we review the two best known iBGP correctness properties, namely: *signaling correctness* and *forwarding correctness* [66] as well as sufficient conditions to enforce them.

5.3.1 Signaling and forwarding correctness

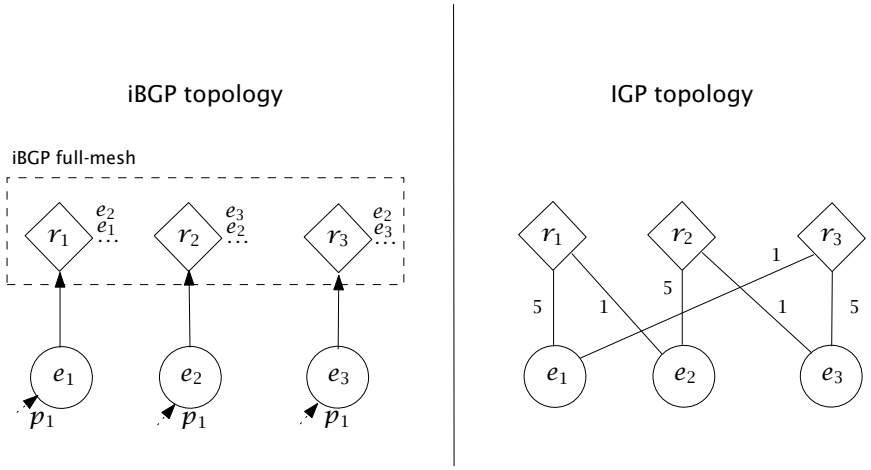


Figure 5.2 An example of BGP configuration in which no stable solution exists (taken from [66]).

A BGP configuration is said to be *signaling correct* if it is free from signaling anomalies. Signaling anomalies [62, 66, 65, 64] or *routing oscillations* occur when BGP routers are unable to converge to a stable routing state. Oscillations can delay BGP convergence for a possibly indefinite

amount of time, wasting resources and negatively impacting traffic. A signaling correct configuration guarantees that BGP will eventually converge to a single predictable stable state. In iBGP, signaling anomalies are due to the interaction with the underlying IGP, and can be further classified into two categories: those induced by partial lack of visibility due to the route reflection topology and those induced by the particular semantics of the MED attribute. In this thesis, we focus on the former type of oscillations as techniques similar to those presented in [65] can be used to deal with MED-induced oscillations. An example of permanent routing oscillation due to route reflection is depicted in Fig. 5.2. In this network, each router r_i prefers the path propagated by its neighbor P_{i+1} over its direct path P_i where subscripts have to be interpreted modulo 3.

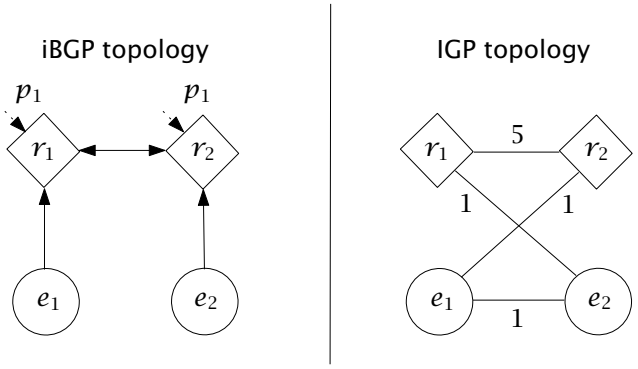


Figure 5.3 An example of BGP configuration in which a forwarding loop exists in the stable state between e_1 and e_2 (taken from [66]).

A signaling correct configuration is *forwarding correct* if it is free from forwarding anomalies. Forwarding anomalies [66, 17] occur when two or more routers make inconsistent forwarding choices in the stable state. Besides inducing suboptimal forwarding and overcomplicating network management and troubleshooting, forwarding anomalies can also disrupt traffic by causing packet deflections and forwarding loops. An example of forwarding deflection leading to forwarding loop is depicted in Fig. 5.3. In this network, e_1 (resp. e_2) only learns p_1 via r_1 (resp. r_2). A forwarding loop is created as e_1 reaches r_1 via e_2. However, once the packet arrives at e_2, it is forwarded towards r_2 (according to e_2's iBGP information) but, to reach r_2, e_2 is using e_1, hence causing the loop.

5.3.2 Sufficient conditions for ensuring correctness

The following set of sufficient conditions guarantees that an iBGP topology B is both signaling and forwarding correct [66]:

1. B does not contain any cycle consisting of UP sessions only ;
2. any route-reflector prefers paths announced by its clients over paths announced by non-clients ;

3. for any pair of routers u and v, there exists a valid signaling path P such that $P = sp(u, v)$, i.e., P coincides with the IGP shortest path between u and v.

In particular, Conditions 1 and 2 ensure that the iBGP configuration is signaling correct, while Condition 3 guarantees forwarding correctness. Although interesting from a theoretical perspective, such conditions are particularly challenging to be applied in real-world topologies. For example, Condition 3 practically forces the BGP topology to be totally congruent to the IGP one, in such a way that even a full-mesh of iBGP sessions [118] is not compliant.

Another known sufficient condition for guaranteeing safety and signaling correctness is the absence of cyclic dependencies among routing preferences, also called *Dispute Wheel* [62]. Formally, a dispute wheel $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ is a triple consisting of a sequence of k routers $\vec{\mathcal{U}} = (u_0 \dots u_{k-1})$ (*pivot* nodes), together with two sequences of non-empty valid signaling paths $\vec{\mathcal{Q}} = (Q_0 \dots Q_{k-1})$ (*spoke paths*) and $\vec{\mathcal{R}} = (R_0 \dots R_{k-1})$ (*rim paths*), such that for each i:

1. each rim path R_i goes from u_i to u_{i+1}
2. each spoke path Q_i goes from u_i to an egress point e_i for prefix p
3. each pivot node u_i prefers path $R_i Q_{i+1}$ over Q_i

where subscripts are intended modulo k with $k = |\vec{\mathcal{U}}|$. A dispute wheel is thus such that each node u_i prefers the path through u_{i+1} over its direct path. In the network depicted in Fig. 5.2, we have $\vec{\mathcal{U}} = (r_1\ r_2\ r_3)$, $\vec{\mathcal{Q}} = ((r_1\ e_1)\ (r_2\ e_2)\ (r_3\ e_3))$ and $\vec{\mathcal{R}} = ((r_1\ r_2)\ (r_2\ r_3)\ (r_3\ r_1))$. During the oscillation, each u_i alternately selects $R_i Q_{i+1}$ and Q_i when $R_i Q_{i+1}$ is not available.

5.4 iBGP deceptions: more sessions, fewer routes

In this section, we illustrate the dissemination problem that can arise in route reflection topologies. As an illustration, consider prefix p_1 in the network depicted in Fig. 5.4. Because of the "prefer eBGP over iBGP" rule of the BGP decision process (step 5, see Table 1.1), e_1, e_2 and e_3 will all select their external route as best. These routers will then advertise their best route to all their iBGP neighbors, namely b (for e_1 and e_2) and r (for e_3). b will collect routes from its clients, select its best route, and propagate it to its neighbors. Because of the "prefer closer egress" rule of the BGP decision process (step 6, see Table 1.1), b will select e_2 because it is closer than e_1. Therefore, b will advertise e_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 e_3 and router a will select e_2.

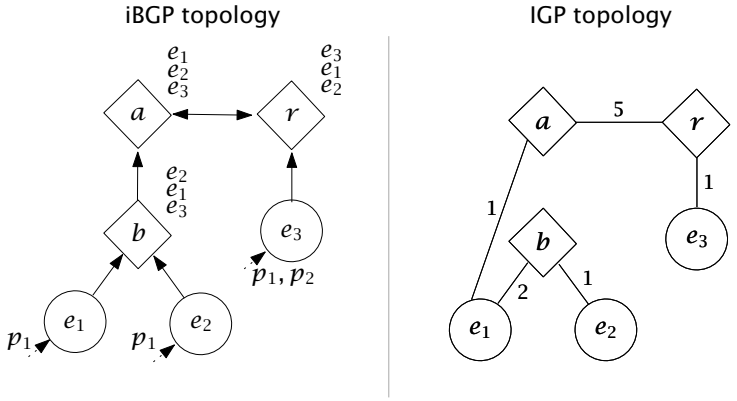


Figure 5.4 An example of BGP configuration which exhibits visibility issues. Indeed, a prefers e_1, but it never learns the route since b only reflects e_2.

Although it prefers e_1, a has no knowledge of it as it only receives route e_2 (resp. e_3) from b (resp. 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 forwarding 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 [66].

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 already been proposed in [104, 105], e.g., to support recently proposed techniques for reducing iBGP convergence time [48]. Indeed, quantitative studies have shown that route reflection leads to very poor route diversity [134]. 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 [17].

Adding OVER sessions to an iBGP topology may have undesirable side effects. Consider the iBGP network in Fig. 5.5 (OVER-RIDE) which is a simplified version of the one in Fig. 5.4. An additional OVER session exists between routers a and e. Since e is the only egress point for prefix p, a will prefer the route that it learns on the OVER session because of the length of the cluster-list is shorter (step 8 of the BGP decision process, see Table 1.1). Since its best route is learned from a peer, a does not propagate it to r, so r end up having no route to prefix p. Either of the two cases can occur depending on r has a less specific route or not. First, if r knows a

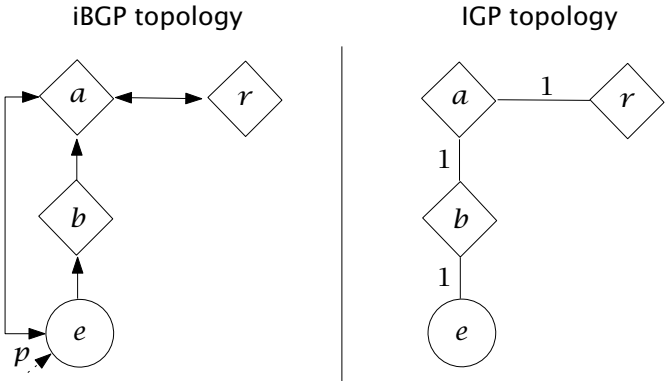


Figure 5.5 The OVER-RIDE gadget. Due to iBGP propagation rules, a does not propagate routes that it receives on its OVER session with e to r. Consequently, r ends up without any route to e while a valid signaling path exists: $(e\ b\ a\ r)$.

less specific route which includes prefix p (e.g., a default route), it will use that route for traffic destined to p, possibly generating forwarding deflections and loops. This implies that it is not safe to assume that prefixes are independent in iBGP. Second, if r has no less specific route for p, it will create a traffic blackhole. Both kinds of anomalies are only due to the iBGP topology. Indeed, the IGP topology is irrelevant in this case because there is only one egress point for p. For this reason, the OVER-RIDE complies with the conditions of [17], yet it is subject to anomalies. Even worse, such anomalies could be triggered by external events, e.g., if an egress point fails.

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 to deploy OVER sessions, in order to lower memory overhead and update churn, as only a subset of reflected routes is announced on OVER sessions.

5.5 Unveiling iBGP dissemination correctness

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

Definition 5.1. *Given an iBGP 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 $x \notin T_B$ or $y \notin T_B$.*

Spurious sessions are not frequent in today's ISP networks. Vendor guidelines also suggest to not deploy them [156]. Nevertheless, spurious sessions have been proposed to solve visibility issues [104, 105], and previous work showed that large ISPs sometimes use them [44, 102]. In fact,

spurious OVERs will appear in any ISP which defines a full-mesh of iBGP sessions among the clients of a route-reflector (see [193, 179] for examples). Another scenario in which spurious OVERs are likely to appear is iBGP reconfigurations. For example, current best practices to replace an iBGP full-mesh with route reflection [68] suggest to progressively introduce UP sessions before removing the full-mesh. Hence, OVER sessions initially in the full-mesh are likely to become spurious in intermediate configurations. Similar considerations hold when reconfiguring from route-reflection to full-mesh (e.g., to improve path visibility for load-balancing purposes). As illustration, 100 out of the 120 possible (83%) per-session orderings create dissemination anomaly when reconfiguring the OVER-RIDE gadget without the initial spurious OVER between a and e (Fig. 5.5) to a full-mesh.

Route dissemination deceptions

As discussed in Section 5.4, the OVER-RIDE 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 works are based on that intuition. In particular, [104, 105] assume that adding an OVER session can only improve route visibility, while [17, 18] 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 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 5.2. *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 to prefix p in the stable state, for any non-empty set of egress points for p.*

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. As an illustration, the OVER-RIDE gadget is signaling correct, but not dissemination correct. Also, a dissemination correct topology is not guaranteed to be forwarding correct. An an illustration, the network depicted in Fig. 5.3 is dissemination correct (all routers receive a route for p_1), but not forward-

ing 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

Besides affecting dissemination correctness, a single spurious OVER can prevent an iBGP topology to be either signaling or forwarding correct.

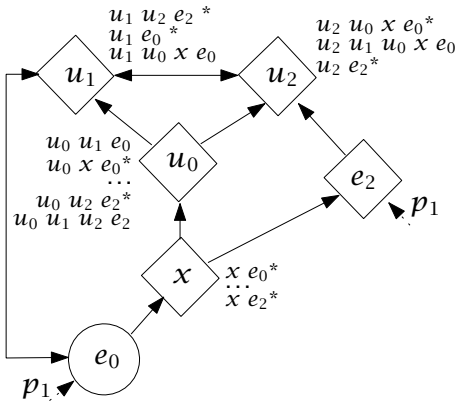


Figure 5.6 Spurious OVER can create routing oscillations.

Consider Fig. 5.6. Every router is equipped with a list of valid signaling paths, sorted in decreasing order of preference. Observe that (u_1, e_0) is a spurious OVER session. We now show that iBGP cannot converge in this 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. 5.6. Observe that the path preferences highlighted in the figure can result from the standard BGP decision process (Table 1.1) if the IGP topology is such that $dist(x, e_0) < dist(x, e_2)$, $dist(u_0, e_0) < dist(u_0, e_2)$, $dist(u_2, e_0) < dist(u_2, e_2)$, and $dist(u_1, e_0) = dist(u_1, e_2)$. 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 lower neighbor address criteria.

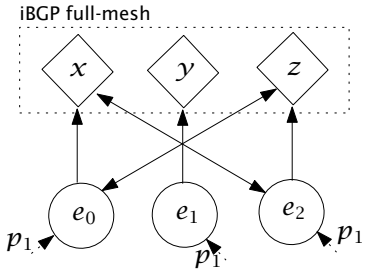


Figure 5.7 Spurious OVER can create forwarding loops.

Forwarding correctness can also be affected by the presence of spurious OVER sessions. Consider the topology in Fig. 5.7, 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 arises for p_1.

5.6 Guaranteeing dissemination correctness

In [145], we show that the problem of deciding whether a signaling correct iBGP topology is dissemination correct is co$\mathcal{NP}$-hard and therefore, computationally intractable. Indeed, a reduction from the well-known 3-SAT complement problem [101] can be built in polynomial time. We also show that the problem of deciding if the addition of a single session can affect the dissemination correctness of an iBGP topology is also co$\mathcal{NP}$-hard.

In this section, we describe how to get around the computational complexity by proposing two different sufficient conditions on the network configurations that guarantee dissemination correctness. We then discuss their practical applicability.

5.6.1 Sufficient conditions for dissemination correctness

Each of the following conditions guarantees a signaling correct iBGP topology B to be dissemination correct.

1. *prefer-client*: all iBGP routers in B prefer routes propagated by clients (on a UP* path) to any other route.

2. *no-spurious-OVER*: B contains no spurious OVER.

In order to prove our results, we need the following lemma.

Lemma 5.1. *Given a signaling correct iBGP topology B, if for any prefix p at least one router in the top layer T_B selects a route for p that was learned over an UP* path, then B is dissemination correct.*

Proof. Consider any prefix p, and let $\bar{r} \in T_B$ be the router that selects a route $\bar{R}$ to p which was learned over an UP* valid signaling path $(e \dots \bar{r})$ (possibly $e = \bar{r}$). By iBGP route propagation rules, $\bar{r}$ propagates route $\bar{R}$ to all routers in T_B. Since B is signaling correct and all routers in T_B receive at least one route for p, all routers in T_B will eventually select a route. Independent of the neighbor from which the best route was learned, routers in T_B 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. □

In the following theorems, we prove that *prefer-client* and *no-spurious-OVER* guarantee dissemination correctness.

Theorem 5.1. *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 r in T_B selects a route to p over an UP* path. Then, the statement follows because of Lemma 5.1.

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 $e_p \in T_B$, then $r = e_p$. Otherwise, there must exist a router r_1 such that $r_1 \leftarrow e_p$, by definition of T_B. 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 $r_1 \in T_B$ then $r = r_1$. 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 $r_1 \notin T_B$. 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 r in T_B that learns a route from one of its clients. Because of iBGP propagation rules, that route must be learned over an UP* path. □

Theorem 5.2. *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_B 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 $e_p \in T_B$, then the statement follows by Lemma 5.1. 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.

- $r_1 \in T_B$ and r_1 learned R' from one of its clients. By the iBGP propagation rules, R' must be learned over an UP* path.
- $r_1 \in T_B$ and r_1 learned R' from a peer r_2. In this case, r_2 must have received R' over an UP* path, otherwise it would not have propagated it to r_1.
- $r_1 \notin T_B$ and r_1 learned R' from one of its clients. Then, r_1 forwards route R' to all its route-reflectors.
- $r_1 \notin T_B$ 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 $r_1 \notin T_B$.

In the first two cases, the statement follows by Lemma 5.1. 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_B for which one of the first two cases applies, yielding the statement. □

5.6.2 Applicability of the sufficient conditions

We now discuss how the sufficient conditions presented above can be 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 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 $P = (r\ s \dots e)$ 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. 5.8, which is the simplest redundant route reflection topology designed according to

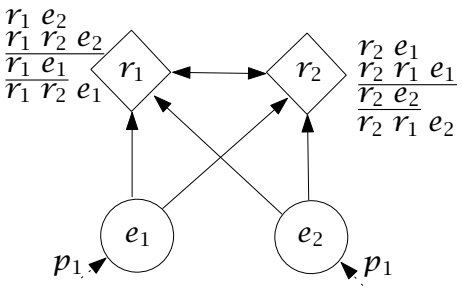


Figure 5.8 Redundant topologies hardly satisfy the *prefer-client* condition.

current best practices [156, 198]. Clients e_1 and e_2 are connected to both route-reflectors r_1 and r_2 which belong to different clusters. Both e_1 and e_2 are egress points for prefix p_1. Even in such a simple scenario, the *prefer-client* condition does not hold, whatever the IGP topology is: underlined paths highlight violations of the *prefer-client* condition. 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 received by e_2 to all the routes received by e_1, because of the "prefer closer egress" rule of the BGP decision process (step 6, see Table 1.1). Hence, r_1 prefers routes learned over path $(r_1\ r_2\ e_2)$ to those over path $(r_1\ e_1)$. This violates the *prefer-client* condition. A similar violation happen if $dist(r_1, e_1) < dist(r_1, e_2)$. 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 [11].

Guideline A. *In redundant iBGP configurations, redundant route-reflectors must belong to the same cluster in order to be able to enforce the prefer-client condition.*

Observe that current best practices for cluster design [198] do not comply with Guideline A.

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 whole 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 discussed in Section 5.5. In such cases, UP sessions can be deployed instead of spurious OVERs, without adversely affecting dissemination correctness.

Guideline B. *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 allow route propagation in one direction only.

5.7 Related work

In this section, we discuss how dissemination correctness relates to previous works on iBGP correctness properties and topology design. We find that dissemination correctness was often overlooked, so extra conditions (see Section 5.6) are needed to keep the validity of the results.

Regarding propagation issues, the recent work from Sobrinho and Quelhas [128] is probably the closest to ours, even if its focus is different. Indeed, the authors aim at computing how many links should be removed in order to disconnect two nodes in a network governed by a policy-based route vector protocol. To that extent, the authors study the difference between usable connectivity and route connectivity in the eBGP world. They show that the existence of a usable path from a to b does not imply that a has a route to b. This corresponds to our finding that having a iBGP valid signaling path between two nodes is not a sufficient condition to guarantee dissemination correctness. They also show that adding a link to the network can reduce the visibility of some of its nodes. The authors argue that route connectivity differs from usable connectivity because of the lack of *export-isotonicity*. Export-isotonicity means that if a node exports a route to a neighboring node, then it also exports to that same neighbor any more preferred route. Interestingly, our OVER-RIDE gadget (Fig. 5.5) is not export-isotonic. Indeed, a exports to r the routes it learns from b, but not the route it directly learns from e (more preferred). Although similar, our work differs from theirs as we show that dissemination problems are not confined to eBGP and can also happen in iBGP. Also, we are interested in determining the connectivity of a network in the stable state making no assumption on the iBGP topology. As such, our complexity results as well as our sufficient conditions could also be applied to their study.

Signaling and forwarding correctness have been introduced and analyzed by Griffin *et al.* in [66]. 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 [66], we find that the proposed sufficient conditions also guarantee dissemination correctness, since they encompass the *prefer-client* condition as formulated in Section 5.6. However, as discussed in Sections 5.2 and 5.6, these conditions are very constraining for real-world networks.

In [114], 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 [114] 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 *prefer-client* condition, it is not enough to prevent dissemina-

tion anomalies caused by multiple valid signaling paths to the *same egress point*, as the OVER-RIDE demonstrates. Moreover, Fig. 5.7 shows an example which matches the conditions of [114] but is not forwarding correct.

In [49], Flavel and Roughan propose to modify the BGP decision process so that the length of the `cluster-list` is compared before the IGP weight towards the egress. Such a variant of iBGP is proved to always converge. However, no guarantee is given for dissemination correctness. Actually, the OVER-RIDE is a simple example where the modified iBGP protocol cannot provide all routers with a route for every prefix. However, by following the guidelines described in this chapter as well as their iBGP modification, one can achieve both signaling and dissemination correctness.

In [17, 18], 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 an example, the OVER-RIDE is fm-optimal but not dissemination correct.

In [104, 105] 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 5.6 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., in the OVER-RIDE). 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 [147], 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 [147] guarantees signaling and dissemination correctness, but cannot guarantee forwarding correctness. Also, conclusions drawn in [44] are similarly affected. Indeed, configuring a top layer full-mesh (as prescribed by Theorem 4.1 in [44]) guarantees a valid signaling path for each pair of iBGP routers, but does not imply dissemination correctness.

Despite the concept of dissemination correctness had not been formalized before, we find that some results in the literature guarantee it as a side effect.

Modifications to the iBGP protocol as proposed in [95] and fine tuning of attributes in iBGP messages as proposed in [28] 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.

BGP Add-Paths [149] 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 according to 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 [137]. Raszuk *et al.* [113] propose to add special route-reflectors in order to distribute multiple routes. Unfortunately, since this technique relies on additional route-reflectors, it does not guarantee the advertisement of all the AS dominant routes, and thus it is not sufficient for dissemination correctness. Packet encapsulation is suggested in both cases to solve forwarding anomalies when not every AS dominant route is propagated. Observe that both proposals are still in the development stage.

5.8 Conclusions

iBGP route reflection provides network operators with good scalability at the cost of possibly introducing routing and forwarding anomalies. In this chapter, we show that iBGP route dissemination anomalies are also possible, triggering unexpected side effects like traffic blackholes and forwarding loops. Moreover, the ability of iBGP to correctly distribute routing information within an ISP can be affected by the addition of even a single iBGP session. This is particularly relevant as prior contributions proposed to fine tune iBGP by adding extra sessions. Hence, we introduce the concept of dissemination correctness to model visibility issues caused by iBGP route propagation rules. We study the computational complexity of checking dissemination correctness and provide sufficient conditions to enforce it in real-world configurations.

We thoroughly review previous work and discuss how existing results relate to dissemination correctness, finding that some contributions need to be revisited. 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., [149, 98, 25]) for decoupling route propagation from route selection in iBGP.

Chapter 6

Lossless BGP reconfiguration with BGP Ships-In-The-Night

6.1 Introduction

During the life of a network, 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, the organization of iBGP sessions may need to be modified, e.g., by replacing the full-mesh of iBGP sessions dictated by the original BGP specification [118] with a route reflection [11] configuration. Also, iBGP configuration changes can be triggered by changes to the underlying Interior Gateway Protocol (IGP) (see Part 2). Unfortunately, IGP configuration adjustments can affect iBGP routing choices, possibly leading to routing and forwarding inconsistencies [66], as well as undesired side effects on internal and external traffic flows [9]. IGP changes may thus require iBGP configuration changes. Similarly, the eBGP configuration need to be changed. 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. In some networks, routing policies are changed on a daily basis [85]. Recent examples also include the so-called "peering wars" that led to the de-peering of large ISPs [173].

The impact of changes to either iBGP or eBGP configuration is hard to predict. The main reason is that local changes on one BGP router can affect routing information as viewed by remote routers in a domino effect in which intermediate routers and possibly message timings play a critical role. Nowadays, 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., [198, 68, 156]), but they typically focus on simple reconfiguration cases. Moreover, current best practices barely take into account the possibility of creating routing and forwarding anomalies during the migration process.

In this chapter, we address the problem of deploying a new BGP configuration in an ISP with no negative impact on the traffic. We consider both eBGP and iBGP configuration changes. The contribution of this chapter is threefold. First, we show that long-lasting routing and forwarding anomalies can occur during BGP reconfigurations even when the initial and the final BGP configurations are anomaly-free. We simulated BGP reconfigurations in a Tier-1 network observing that a significant number of anomalies persists for large parts of the reconfiguration process. Such theoretical and practical insights expose the fragility of correct BGP configurations, as different kinds of anomalies can be triggered even by simple changes on a single BGP session. Second, we study the problem of finding an ordering of configuration changes which guarantees an anomaly-free migration process. Unfortunately, we show that finding such an order is a computationally intractable problem. We also present simple cases in which such an order does not exist at all. Third, we propose a generic framework that enables lossless BGP reconfigurations. Our solution is based on the possibility of current routers to support independent and isolated control and forwarding planes. We describe a possible implementation of our framework, and we present a working prototype. We show the effectiveness of our approach through a use case and we study its scalability.

Beyond addressing current needs of the operators, our approach can be leveraged to achieve additional agility and flexibility, which, in turn, leads to competitive advantages to ISPs. For example, the ability to frequently change eBGP configuration enables ISPs to adapt routing policies to observed traffic trends and turn off network devices during idle time (e.g., during the night). By rapidly and safely switching preference of routes received from their eBGP neighbors, ISPs can also reduce their transit costs, and take full advantage of services (e.g., Equinix Direct [178]) aiming at more flexible establishment of upstream connectivity.

The rest of this chapter is organized as follows. Section 6.2 states the BGP reconfiguration problem, and discusses its practical relevance highlighting deficiencies of simple approaches and current best practices. Section 6.3 presents examples in which an operational reconfiguration ordering does not exist. Section 6.4 explains our proposed solution. Section 6.5 reports results of our case study. Section 6.6 presents related work. Finally, Section 6.7 concludes the chapter.

6.2 Seamless BGP reconfigurations

In this section, we define the BGP seamless reconfiguration problem. By analyzing historical configuration changes deployed in a Tier-1 ISP, we show that the problem has practical relevance. Moreover, we show that applying incremental approaches and current best practices [198, 68] incurs the risk of introducing migration-induced anomalies. Finally, by means of

simulations, we quantify the disruptions generated by existing approaches in simple migration scenarios.

6.2.1 Problem statement

We define a BGP configuration C as a tuple (B, I, Υ) consisting of an iBGP topology B (as defined in Chapter 5), an IGP topology I (as defined in Chapter 2), and a function Υ that maps each destination prefix to a set of border routers receiving an eBGP route to this prefix. In the following, we refer to border routers receiving an eBGP route to a prefix as *egress points* for that prefix. Unless otherwise specified, we always refer to routes that are equally preferred according to the first four decision steps (see Table 1.1). Observe that each element of a BGP configuration influences routing decisions taken by routers. Indeed, the iBGP topology regulates what routes are known by each router, the IGP topology affects route preference at different routers, and Υ determines what routes are available to each prefix. Observe that function Υ encodes both the eBGP topology and eBGP policies. We assume that each iBGP topology B complies with current design best practices:

1. B is a hierarchy where each router is assigned to one layer ;
2. routers in the top-layer have no route-reflectors and are all peers ;
3. routers not in top-layer have at least one route-reflector.

We say that a BGP configuration C is *anomaly-free* if no signaling, forwarding and dissemination anomalies occur for any destination prefix. In this chapter, we consider that the combination of egress points for any destination (i.e., Υ) is fixed, and we show that reconfigurations are hard even when this assumption holds. Note that, in the worst case, each prefix is learned from a different subset of border routers. In this case, we are consistent with previous work on configuration correctness [66, 145]. Since we study BGP reconfigurations, we deal with BGP configurations that change over time. Whenever it is not clear from the context, we will use convenient indices. For example, C_t is the BGP configuration at time t. We define two special indices i and f that refer to the initial and the final time in the reconfiguration, respectively.

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. Also, we assume that the underlying IGP configuration does not change during the reconfiguration. We disregard reconfiguration where router configurations are modified on a per-prefix basis. Indeed, given the size of current BGP RIBs (around 450,000 prefixes [157]), per-prefix reconfigurations incur severe penalties in the speed and the ease of management of the migration process.

We further define a reconfiguration as *seamless* if for any migration step j, with $i \leq j \leq f$

- C_j is anomaly-free;

- C_j is not subject to unintended traffic shifts.

An *unintended traffic shift* is a change in the best path selected by a router to a given prefix in which the egress point is neither the initial nor the final one. We also talk about unintended traffic shift when a router switches between the initial and the final egress points multiple times. Unintended traffic shifts can be disruptive for at least three reasons: (i) they can disrupt traffic engineering policies (e.g., forcing traffic to exit from other continents), (ii) they can adversely impact costs (e.g., swelling traffic flows on transoceanic links) and, (iii) they increase the likelihood of congesting some links (e.g., under-provisioned backup links). Personal communications with operators confirm that avoiding unintended traffic shifts is one of the most important requirements identified by network operators. Observe that unintended traffic shifts are peculiar to the reconfiguration problem which explains why they have not been studied in prior work.

If a reconfiguration is not seamless, routing and forwarding anomalies occur in intermediate configurations. 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 [142]. Migration anomalies contrast with short-lived protocol-dependent issues, like those occurring transiently during protocol convergence.

6.2.2 Frequency of BGP reconfigurations

To illustrate the frequency of BGP configuration changes, we analyzed the BGP configurations of approximately 20% of the routers of a Tier-1 ISP, from April 2010 to July 2011. The 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. Overall, we detected 1,337 BGP configuration changes. Among these changes, sessions additions and removals were the most common. Sessions additions happened 5,828 times, encompassing 976 eBGP sessions and 4,852 iBGP sessions. Session removals were less frequent but still not rare, as they happened 236 times for eBGP sessions and 1,440 times for iBGP sessions. At each router, eBGP sessions were typically added in groups, while iBGP sessions were mostly added in pairs of redundant sessions with two route-reflectors. By looking at route-map names, we also registered 41 changes of inbound eBGP policy and 77 modifications of outbound eBGP policy.

Finally, we collected less frequent miscellaneous changes, encompassing the promotion of a router to the role of route-reflector (11 times), AS number modification on an eBGP peer (8 times), and address family enabling (3 times) and disabling (5 times) on eBGP sessions. These results

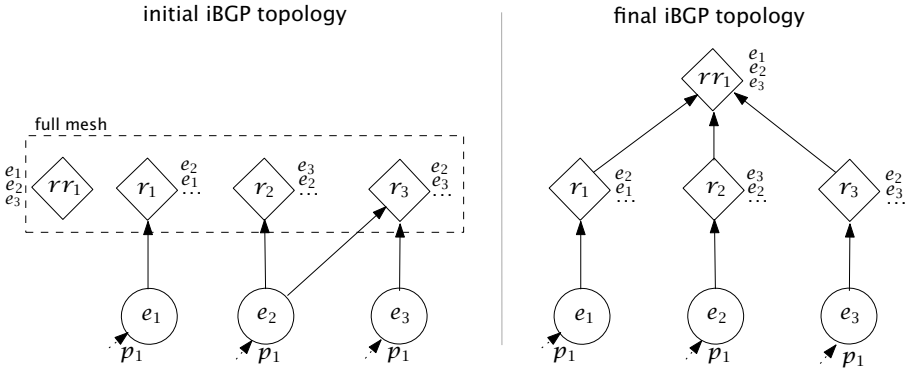


Figure 6.1 An example in which the bottom-up strategy, suggested by the current best practices, creates routing oscillations during the reconfiguration.

testify that reconfigurations of already established BGP sessions are also performed by operators, even if less frequently than the addition or the removal of BGP sessions.

6.2.3 Current best practices provide no guarantee

Currently, network operators can only rely on a few rules of thumb to perform BGP reconfiguration. Such rules of thumb only concern simple topological changes, like the replacement of a fully-meshed iBGP topology with a two-layer route reflection hierarchy [198, 68]. In the following, we show that current best practices to migrate from a full-mesh to a route-reflection hierarchy provide no guarantee on the absence of migration anomalies. To be as general as possible, we consider as current best practice an extension of the procedures proposed in [198, 68] devised according to private discussions with operators. Such an extension consists in reconfiguring routers, one at the time, on a per-layer basis, in a bottom-up fashion (i.e., starting from the bottom layer up to the top one). Each router r is reconfigured by activating all the sessions r has in the final configuration before shutting down all the sessions r maintains exclusively in the initial configuration.

An example of migration oscillation created by following best current practices is reported in Fig. 6.1. In this network, a cyclic preference of routes exists among r_1, r_2, and r_3 (i.e., a dispute wheel, see Chapter 5). However, in the initial configuration, session (e_2, r_3) ensures that r_3 always receives a route from its most preferred egress point, forcing a stable state to be eventually reached. The final configuration is also oscillation-free since rr_1 breaks the cycle of route preferences by steadily selecting the route from e_1.

The bottom-up approach mandates to reconfigure e_1, e_2, and e_3, before all the other routers. However, after the reconfiguration of e_2, session $(e_2\ r_3)$ is removed and the resulting intermediate configuration becomes

subject to routing oscillations. The problem is fixed only when migrating middle layer routers. In contrast, a seamless migration can be achieved by reconfiguring r_3 first. Indeed, after its reconfiguration, r_3 will never learn the route propagated by r_2 anymore, since r_3 has sessions only with e_3 and rr_1, which are guaranteed to always select the route from e_3 and e_1 respectively. This breaks the cycle of route preferences, ensuring no migration oscillation. Similar examples can be found for other migration anomalies, like forwarding loops and unintended traffic shifts.

6.2.4 Quantitative analysis

To quantify the anomalies generated by simple migration approaches, we simulated several BGP reconfiguration scenarios on a Tier-1 network consisting of roughly 100 iBGP routers organized in three layers of route reflection. We simulated two kinds of experiments: iBGP topology changes and eBGP policy modifications.

Study of the effects of iBGP topology changes in a Tier-1

The first kind of experiments consisted in reconfiguring an iBGP full-mesh into the final route reflection hierarchy used by the Tier-1. We considered three ordering strategies: (i) random ordering, (ii) random ordering in which top layer routers are reconfigured at the end and, (iii) bottom-up ordering as dictated by current best practices. We denote these strategies as *RND* (Random), *RBT* (Random But Top), and *BTU* (Bottom-Up), respectively. For each strategy, we run 50 different experiments corresponding to a different ordering. For each experiment, we used C-BGP [111] to compute all the BGP routing tables in the intermediate configurations.

Fig. 6.2 plots the fraction of experiments during which different types of anomalies occur. A data point (x, y) in the graph means that $(100 * y)\%$ of the orderings of a given 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. This makes random orderings clearly not viable in practice. *RBT* migrations were not subject to LoVs, however they were responsible for several migration issues. Indeed, in more than 90% of the experiments, loops occurred during more than 10% of the migration steps. Even worse, unintended traffic shifts occurred during more than 55% of the *RBT* migration process in almost all the experiments. *BTU* performed slightly better than *RBT* on traffic shifts, but, surprisingly, *BTU* creates more forwarding loops than *RBT* on average on the considered network. Indeed, in almost 60% of the experiments, loops were raised during more than 35% of the migration steps. Observe that the selection of a temporary BGP next-hop counts as an unintended traffic shift. We stress that each of these traffic shifts can potentially affect several prefixes, among which the prefixes that drive the vast majority of traffic [80]. Moreover, our experiments show that performance degradation can be long-lasting. As a rule of thumb, assuming that a migration

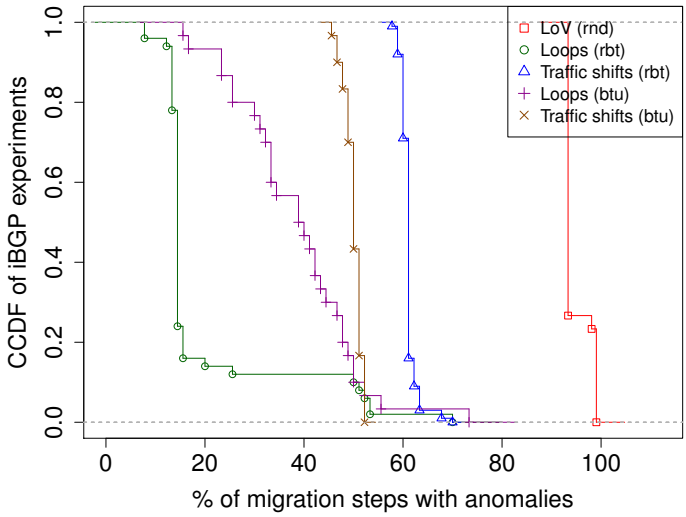


Figure 6.2 Percentage of the migration process affected by anomalies during a full-mesh to route reflection reconfiguration of a Tier-1.

step takes about 3 minutes (e.g., for ensuring BGP convergence), having a loop for 35% of the migration process translates to losing traffic for about 100 minutes.

Study of the effects of eBGP policy modifications in a Tier-1

In the second kind of experiments, we measured the amount of unintended traffic shifts created by changes of eBGP policies. In each of those experiments, we modified the value of the local-preference (LP) assigned to the routes received by a given neighboring AS. This scenario can arise in the case of de-peering or after a change of economical relationship. We now describe the model that we used to measure the unintended traffic shifts.

Let s be the neighboring AS to be renumbered. Let $E = \{e_0 \ \dots \ e_n\}$ be the set of edge routers that maintain an eBGP session with s. Without loss of generality, we assume that only one session is defined between every $e_i \in E$ and s allowing us to identify the session by the edge router. Let $P = \{p_0 \ \dots \ p_k\}$ be the set of the IP prefixes announced by s. Let D be a function that associates to each prefix p, the set of all edge routers that receive a route for p. Let T be the set of all edges routers that learn a prefix in P as $T = \bigcup_{p \in P} D(p)$. Notice that $E \subseteq T$ since other egresses than the one connected to s can learn a prefix in P. We denote with lp_{init} (resp. lp_{final}), the initial (resp. final) LP value applied on all the sessions $e \in E$. Moreover, the lp function associates to each e_i the current LP value being applied. We say that e_i has been renumbered when $lp(e_i) = lp_{final}$. We denote with $(e_1 \ \dots \ e_n)$, the ordering in which the edges are being renumbered and with $1 \leq t \leq n$ the current step of the renumbering process.

To represent the concept of traffic shift, we define the utilization function $util(e_i, p_j)$ that associates to each edge router e_i and to each destination p_j, the percentage of routers u in the network such that $nh(u, p_j) = e_i$ in the stable state. For each edge router $e \in T$, we define the aggregated utilization function as:

$$aggr(e) = \sum_{p \in P} \frac{util(e, p)}{|P|}$$

During an eBGP renumbering process, the aggregate utilization of each edge might vary as some eBGP sessions are renumbered but not all of them. We define the utilization matrix U_{ij} where $u_{ij} = aggr(e_i)$ at the renumbering step j where $1 \leq i \leq |T|$ and $1 \leq j \leq n$. We further denote with ΔU the differentiated matrix obtained by differentiating column $(j + 1)$ and j of U. Each value of ΔU gives the total amount of traffic that has shifted towards (positive variation) or away (negative variation) from the corresponding edge between two consecutive renumbering steps. We further define $TheoryTS$ by differentiating the first and the last column of U. $TheoryTS$ is the variation of traffic between the initial and the final state for each edge. We define the $TransientTS(A, t)$ function which associates to each column t of a differentiated matrix A, the absolute value of the sum of the positive (resp. negative) variations. Applied to ΔU, $TransientTS$ returns the percentage of traffic that has shifted due to each renumbering step. Applied to $TheoryTS$, $TransientTS$ returns the smallest percentage of traffic that can be shifted during the renumbering. Notice that the sum of the positive variations always equal the sum of the negative variations as the traffic being pushed away from a router will always exit via another one (traffic conservation). We can now define the total transient TS, $TotalTS$ that has been created during the renumbering process as:

$$\sum_{0 \leq t \leq |E|} TransientTS(\Delta U, t) - TransientTS(TheoryTS, 1)$$

$TotalTS$ represents the average number of times each router transiently switched next-hop during the entire renumbering process of the eBGP sessions with AS s.

As an illustration, let's consider the renumbering scenario described in Fig 6.3. In this network, all the sessions with AS_2 will be renumbered from $lp_{init} = 120$ to $lp_{final} = 60$. More particularly, we have $E = \{e_1, e_2, e_3\}$, $P = \{p_1\}$ and $T = \{e_1, e_2, e_3, e_4, e_5\}$. We assume that (e_1, e_2, e_3) is the renumbering sequence being followed. The values of U and ΔU are given by:

$$U = \begin{bmatrix} .40 & 0 & 0 & .20 \\ .30 & .60 & 0 & .20 \\ .30 & .40 & 1 & .20 \\ 0 & 0 & 0 & .20 \\ 0 & 0 & 0 & .20 \end{bmatrix} \quad \Delta U = \begin{bmatrix} -.40 & 0 & +.20 \\ +.30 & -.60 & +.20 \\ +.10 & +.60 & -.80 \\ 0 & 0 & +.20 \\ 0 & 0 & +.20 \end{bmatrix}$$

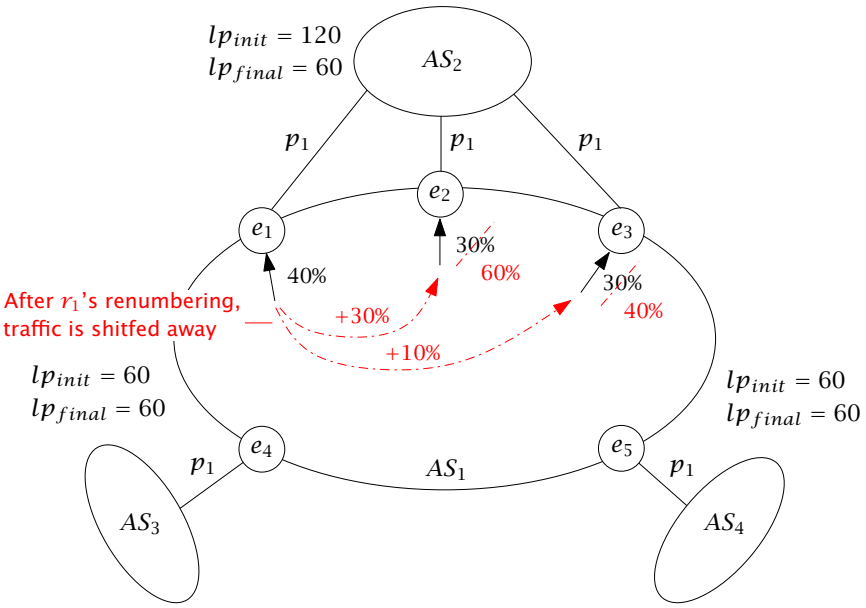


Figure 6.3 Lowering the LP applied on the eBGP sessions with s_2 (e.g. due to a change of the peering agreement) triggers significant traffic shifts. Here, 40% of the traffic destined to p_1 transiently shifts to e_2 and e_3 after having lowered the local-preference of e_1.

Computing the $TransientTS$ on each column of ΔU, we see that 40% of p_1-traffic shift after e_1 renumbering (traffic is spitted on e_2 and e_3), 60% after the renumbering of e_2 (as e_3 is now the only egress used) and 80% after the renumbering of e_3. $TransientTS(TheoryTS, 1)$ is equal to 40%. It means that 40% of the traffic will shift in any case. $TotalTS$ is thus equal to $(40\% + 60\% + 80\%) - 40\% = 140\%$.

Using this technique, we have quantified the unintended traffic shifts created in a Tier-1 network during a LP renumbering. Our data set consisted in the C-BGP [111] model of the Tier-1 along with a dump of the Adj-RIBs-In from the main route reflectors and volume statistics. Based on the volume statistics, we selected 940 prefixes that together were responsible for 80% of the traffic [134]. Then, we identified all the neighboring ASes that were 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 ASes and 250 eBGP sessions. For simplicity, we assumed the Tier-1 to initially apply the same LP on all the eBGP peerings it keeps with the same neighboring ISPs. We used CBGP to compute the routing tables of every router in the network before, during, and after the LP renumbering. We then computed the $TotalTS$ value as described above. 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,

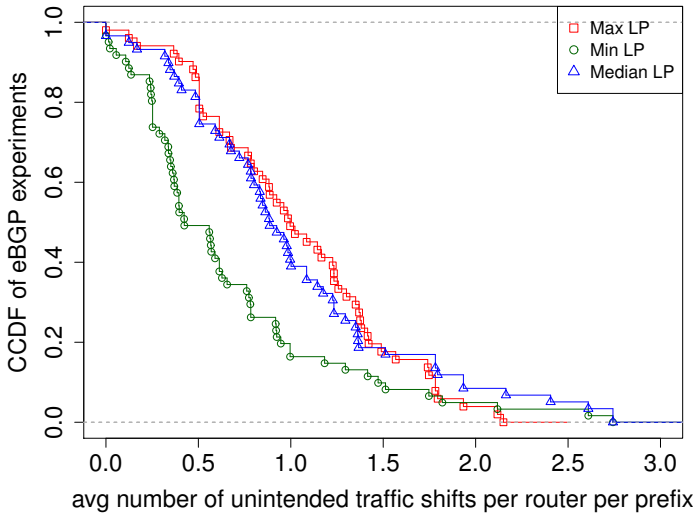


Figure 6.4 Average number of next-hop changes per-prefix each router in a Tier-1 sees during the modification of the LP attribute of all the eBGP sessions with a neighboring AS.

we considered different values for lp_{final}. Namely, we set lp_{final} to the minimum, maximum, and an intermediate (median) value among those found in the Tier-1 configuration. These scenarios correspond to turn a neighboring ISP into a provider, peer, customer, respectively.

Figure 6.4 shows the complementary cumulative distribution of the average number of unintended traffic shifts per router ($TotalTS$). Each point in the plot corresponds to an experiment involving a different neighboring ISP, a different value of lp_{final}, and a different ordering. On average, 50% (20%, resp.) of the routers experience at least 1 (resp., 1.5) unintended traffic shifts for each prefix announced by the ISP considered in the experiment when lp_{final} 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_{final} 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 creating traffic congestion for many migration steps. Additionally, eBGP churn can increase the likelihood of route damping. We expect these results to be a source of concerns for 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 [80]. Observe that lowering the LP to the minimum value creates less traffic shifts on average. In fact, contrary to the other two scenarios, routes affected by setting the LP to the minimum value never attract additional traffic, and

can only be de-selected by routers that preferred them before. Still, in few experiments, the average number of unintended traffic shifts is more than 2.5 per router per prefix.

6.3 An algorithmic approach is not viable

Given that reconfigurations are frequent and that they can have a significant impact on traffic forwarding (see Section 6.2), we would like to compute a migration ordering that ensures seamless reconfigurations. Unfortunately, the results presented in this section show that this is not always possible.

Indeed, despite our assumption of anomaly-free initial and final configurations, finding an operational ordering that guarantees no migration anomalies is computationally hard. Indeed, we proved [143] that finding such an operational ordering is $\mathcal{NP}$-hard in both the iBGP and the eBGP cases, using a polynomial-time reduction from 3-SAT problem. The reduction is based on mapping boolean assignments of a 3-SAT instance into reconfiguration orderings. In Chapter 7, we show that the same reduction can be used to prove the computational complexity of avoiding anomalies in IGP reconfigurations that also consider BGP. It should be noted that the complexity of the reconfiguration problem cannot be derived from the computational intractability of assessing the correctness of a single BGP configuration (see Chapter 5). For instance, sufficient conditions enabling seamless migrations might exist, enabling the design of an efficient algorithm that preserves correctness with no need to inspect each intermediate configuration.

In this section, we present examples in which every operational ordering leads to migration anomalies. We consider both iBGP topology changes, and eBGP policies changes.

6.3.1 iBGP topology changes

From an algorithmic point of view, changing the iBGP topology can be formalized as follows. We refer to an ordering in which to reconfigure iBGP sessions guaranteeing a seamless migration as *seamless ordering*. Recall that we do not consider per-prefix reconfigurations as they are too slow.

Problem 6.1 (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 $B_i \setminus B_f$.*

To be as general as possible, we allow multiple sessions involving the same router to be simultaneously added or removed at each migration step. This 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, admitting simultaneous changes on arbitrary sessions is less realistic, since perfect synchronism between routers must be assumed for both configuration commits and processing of BGP updates at multiple devices. Moreover, allowing simultaneous operations on different routers overcomplicates controlling the reconfiguration, e.g., if a commit fails.

Observe that SOCP does not take into account possible changes in the interdomain routing. Indeed, given an initial configuration $C_i = (B_i, I, \Upsilon)$, Υ is assumed not to change throughout the migration process. In the following, we show that even if eBGP is stable, there are cases in which a seamless ordering does not exist. Moreover, there are cases in which every reconfiguration ordering is not:

1. oscillation-free;
2. LoV-free;
3. deflection-free;
4. free of unintended traffic shifts.

It is simple to extend those examples to cases in which no reconfiguration ordering is free from different kinds of anomalies, e.g., some orderings create migration oscillations while others forwarding loops. In the following, we show two examples in which migration oscillations and migration loops cannot be avoided, respectively. We experimentally confirmed the behavior of each examples by emulating the initial, the final, and the possible intermediate BGP topologies in a virtual environment.

Control-plane anomalies cannot always be avoided

Fig. 6.5 depicts an example in which every reconfiguration ordering creates a permanent oscillation at some reconfiguration step. Observe that both the initial and the final configurations are oscillation-free. Indeed, it is easy to check that the configurations are guaranteed to converge to the stable states reported in Fig. 6.5.

However, an oscillation occurs in every migration ordering. Indeed, since sessions to be added and removed have no routers in common, we have two cases.

- $add(e_1, r_3)$ before $remove(e_x, r_2)$. Immediately after the addition of (e_1, r_3), r_2, r_3, and r_4 respectively prefer paths $(r_2\ r_4\ e_4)$, $(r_3\ r_2\ e_x)$, and $(r_4\ r_3\ e_1)$ for prefix p_2.
- $remove(e_x, r_2)$ before $add(e_1, r_3)$. Immediately after the removal of (e_x, r_2), r_1, r_2, and r_3 respectively prefer paths $(r_1\ r_2\ e_2)$, $(r_2\ r_3\ e_3)$, and $(r_3\ r_1\ e_1)$ for prefix p_1.

In both cases, a cyclic preference of routes prevents iBGP to converge to a stable state [66] for either p_1 or p_2.

In addition to routing oscillations, Fig. 6.6 depicts an example in which all the reconfiguration orderings cause a dissemination anomaly. Observe

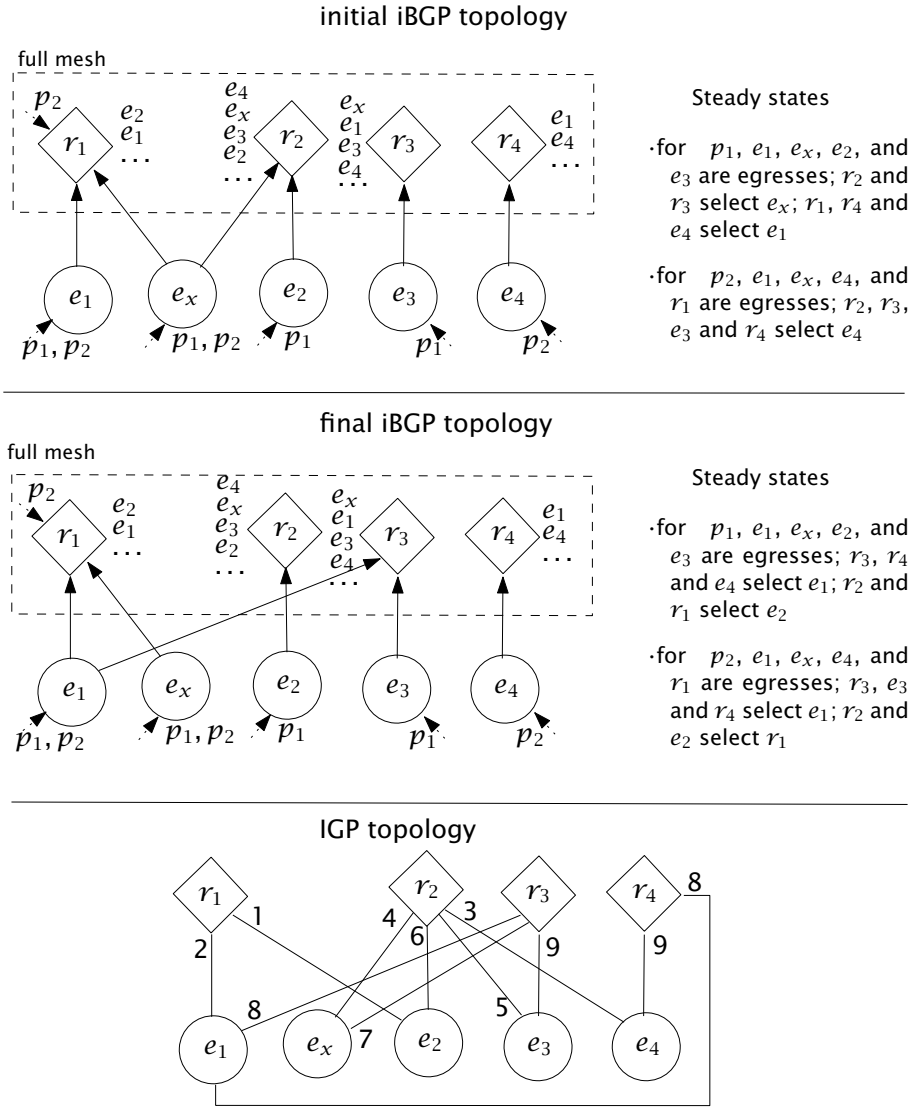
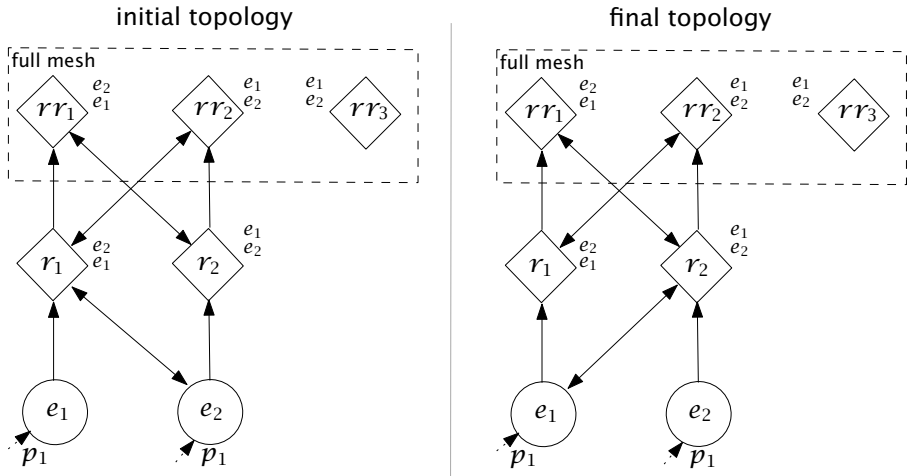


Figure 6.5 Twice-Bad gadget, an iBGP topology change case in which an oscillation-free reconfiguration ordering does not exist.



guaranteed steady state

- $\pi(r_1) = (r_1\ e_2)$
- $\pi(r_2) = (r_2\ e_2)$
- $\pi(rr_1) = (rr_1\ r_2\ e_2)$
- $\pi(rr_2) = (rr_2\ r_2\ e_2)$
- $\pi(rr_3) = (rr_3\ rr_2\ r_2\ e_2)$

guaranteed steady state

- $\pi(r_1) = (r_1\ e_1)$
- $\pi(r_2) = (r_2\ e_1)$
- $\pi(rr_1) = (rr_1\ r_1\ e_1)$
- $\pi(rr_2) = (rr_2\ r_1\ e_1)$
- $\pi(rr_3) = (rr_3\ rr_1\ r_1\ e_1)$

Figure 6.6 Double-Cross gadget.

that all the routers have a route to p in the initial and in the final configuration as highlighted in the bottom part of the figure. However, in any reconfiguration ordering, one of the following cases apply.

- $add(e_1, r_2) < remove(r_1, e_2)$: 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.
- $remove(r_1, e_2) < add(e_1, r_2)$: 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, resp.) will select the route propagated on the OVER session (r_2, rr_1) ((r_1, rr_2), 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.

Data-plane anomalies cannot always be avoided

Similarly to control-plane issues, scenarios exist in which forwarding anomalies occur in every reconfiguration ordering, even if the initial and final configuration are deflection-free. Consider the example in Fig. 6.7. In the initial configuration, all routers but s sends traffic to e_0, 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 the final configuration, all routers but s select the route received 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, one of the following cases apply to the intermediate configuration in every reconfiguration ordering.

- $remove(e_0, r_2)$ before $add(e_1, r_1)$: r_1 and r_2 are forced to select $(r_1\ e_0)$ and $(r_2\ e_1)$ respectively, hence a loop occur between r_1 and r_2 (see the IGP topology).
- $add(e_1, r_1)$ before $remove(e_0, r_2)$: because of path preferences, r_1 and r_2 will select $(r_1\ e_1)$ and $(r_2\ e_0)$ respectively. 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 rise to a loop between rr_1 and rr_2 (see the IGP topology).

In both cases, a migration loop occur.

In addition to forwarding loops, Fig 6.8 depicts an example in which router t is subject to unintended traffic shifts for either p_1 or p_2, whatever the reconfiguration ordering is. In the initial configuration, t steadily selects $(t\ e_5)$ for both p_1 and p_2. Indeed, t receives no path in B_i from r_1 and r_2, since they learn their best routes $(r_1\ e_2)$ and $(r_2\ e_1)$ via an OVER session. On the contrary, in B_f, r_1 and r_2 steadily select $(r_1\ e_1)$ and $(r_2\ e_4)$ respectively for p_1. Thus, t steadily selects $(t\ e_1)$, because of egress point

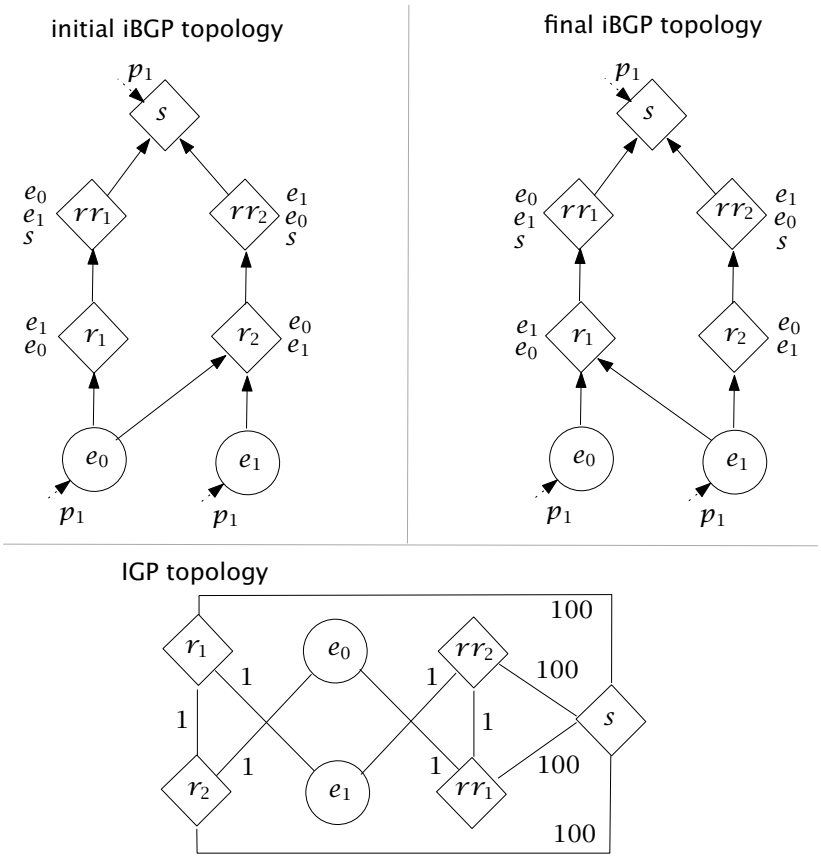


Figure 6.7 Pylon gadget, an iBGP topology change case in which a loop-free reconfiguration ordering does not exist.

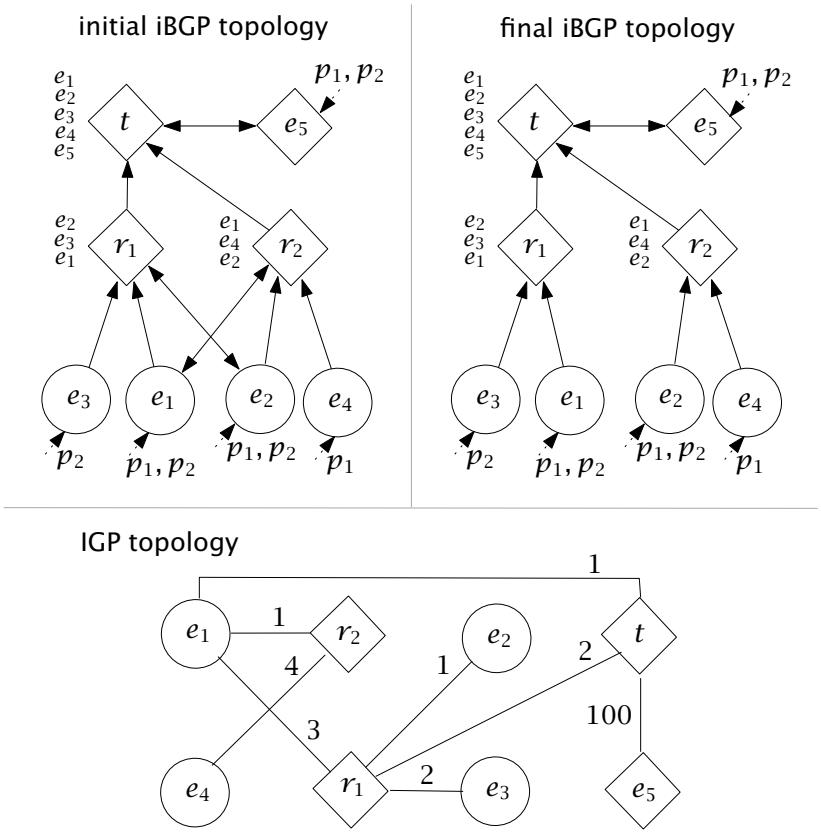


Figure 6.8 Pyramid gadget, an iBGP topology change case in which unintended traffic shifts occur in every reconfiguration ordering.

preferences. For p_2, r_1 and r_2 steadily select $(r_1\ e_3)$ and $(r_2\ e_2)$ respectively, hence t will select $(t\ e_2)$, because of egress point preferences. In any reconfiguration, we have one of the following cases.

- $remove(e_1, r_2) < remove(e_2, r_1)$. In this case, consider prefix p_1. Immediately after $remove(e_1, r_2)$, 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 will steadily select $(t\ r_2\ e_4)$, that is the route announced by an egress point to which t will not send traffic in B_i nor in B_f.
- $remove(e_1, r_2) > remove(e_2, r_1)$. In this case, consider prefix p_2. Immediately after $remove(e_2, r_1)$, 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 in an intermediate configuration.

6.3.2 eBGP policy changes

Similarly to the iBGP topology change problem, the eBGP policy change problem is stated as follows.

Problem 6.2 (Policy Ordering Application Problem (POAP))**.** *Given the initial and the final routing policies, compute an ordering in which to apply the new policies on routers while guaranteeing a seamless migration.*

POAP boils down to studying how intermediate policies affect the set of routes injected in iBGP. Indeed, both the IGP and the iBGP topologies are assumed not to change during the reconfiguration, that is $C_i = (B, I, \Upsilon_i)$ and $C_f = (B, I, \Upsilon_f)$, with possibly $\Upsilon_i \neq \Upsilon_f$. In intermediate configurations, function Υ can also be different from both Υ_i and Υ_f. Hence, our formulation of the problem encompasses the cases in which the set of egress points for a given prefix changes only in the intermediate configurations and also between the initial and the final configurations. Since we assume eBGP stability, the Υ function in intermediate configurations depends only on the BGP reconfiguration ordering.

Changing the eBGP policies potentially unveils control-plane and data-plane anomalies in intermediate configurations. Again, migration anomalies cannot be avoided in some cases. Fig. 6.9 shows an example in which a migration loop cannot be avoided, even if the initial and the final configurations are deflection-free. Consider p_1. In the initial configuration, e_2 and e_3 do not select eBGP routes because of the local-preference settings. Hence, e_1 and rr_1 are the only two egress points for p_1. As a consequence, r_1, r_2, e_2, and e_3 select e_1 because of egress point preferences, while r_3, r_4, and e_4 select rr_1 because they are not aware of other egress points. The

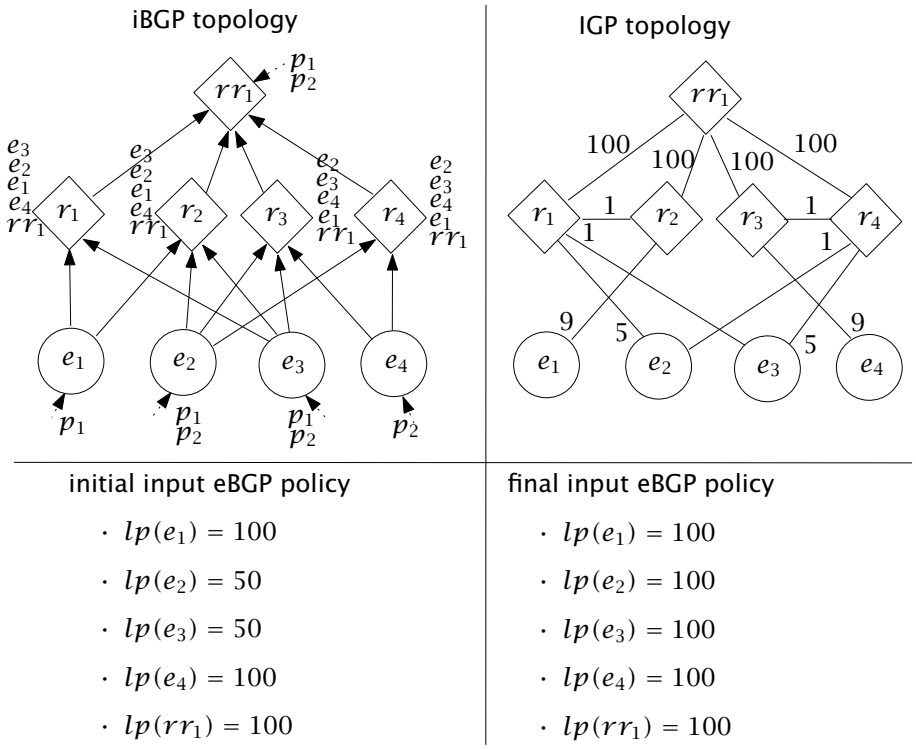


Figure 6.9 Carousel gadget, an eBGP policy reconfiguration case in which forwarding loops occur in every reconfiguration ordering.

IGP topology ensures no deflection. In the final configuration, all e_i with $i = 1, 2, 3$ and rr_1 are egress points for p_1. Also, r_1 and r_2 select e_3, and r_3, r_4, and e_4 select e_2, because of egress point preferences. Since r_1 and r_2 (resp. r_3 and r_4) agree on the egress point to use, no deflection occurs. Similar arguments apply to p_2. However, if e_2 is reconfigured before e_3, then r_2 starts receiving and selecting the route from e_2, because of egress point preferences. However, r_1 still selects the route from e_1 since it is not aware of e_2. Because of the IGP topology, a loop is generated between r_1 and r_2. A symmetric loop occurs between r_3 and r_4 if e_3 is reconfigured before e_2.

In addition to migration loops, all the other kinds of migration anomalies can be created by changing the eBGP configuration, unless the BGP topology is guaranteed to be (signaling, forwarding and dissemination) correct for any possible set of egress points (see Chapter 5).

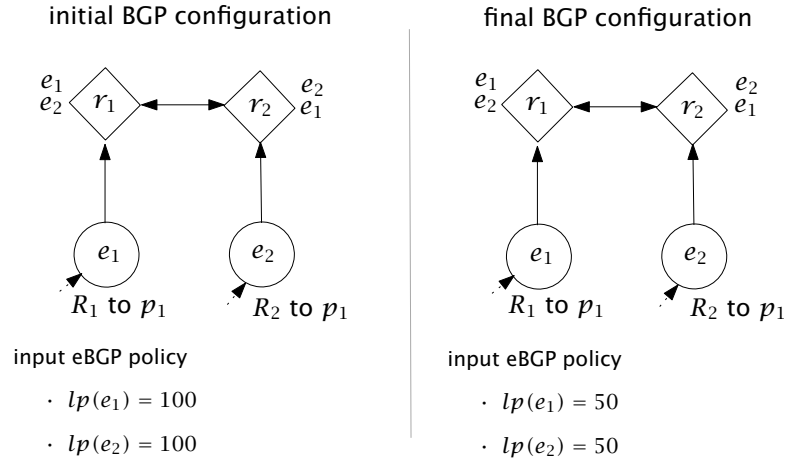


Figure 6.10 Fragile gadget, an eBGP policy reconfiguration case in which unintended traffic shifts occur in every reconfiguration ordering.

In addition, there are cases in which unnecessary traffic shifts cannot be avoided. The example in Fig. 6.10 represents a scenario in which the local-preference of the eBGP routes R_1 and R_2 has to be lowered, e.g., because a change in the commercial relationship with the neighboring ISP. In this case, an unnecessary traffic shift occurs in every migration ordering. Indeed, in both C_i and C_f, traffic towards p_1 is load-balanced among e_1 and e_2, since r_1 and e_1 use R_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 because the route is temporarily assigned a higher local-preference with respect to R_1. Hence, r_1 and e_1 are subject to an unnecessary traffic shift that holds until routing policy is changed on e_2. A symmetrical traffic shift occurs if e_2 is migrated before e_1.

6.4 A general solution for BGP reconfigurations

Section 6.3 shows that seamless BGP 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 enabling seamless migrations. We propose to run two distinct control-planes on all routers in the ISP, as it is normally suggested for IGP reconfigurations (e.g., [68]). The co-existing control-planes run different configurations, i.e., one control-plane runs the initial configuration and the other the final configuration. To avoid control-plane anomalies, the two control-planes are completely isolated, and are already converged to a stable state when starting the reconfiguration. To avoid data-plane anomalies, our solution specify what control-plane must be used network-wide for packet forwarding. We refer to this approach as BGP *Ships-In-The-Night* (SITN).

6.4.1 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 decisions of other routers. Indeed, running the initial and the final configurations in separate control-planes enables each router to compute both the initial and the final BGP routing tables (RIBs). Then, a router reconfiguration just mandates the router to forward 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 router;
- independent propagation of all routes to all routing processes within each router.

In order to simulate the co-existence of multiple routing processes on the same router, we can leverage the Virtual Routing and Forwarding (VRF) feature [159] available on commercial devices. This feature is usually used as a basis for MPLS L3VPNs and BGP multi-topology. VRF 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. Moreover, each namespace can define its own iBGP

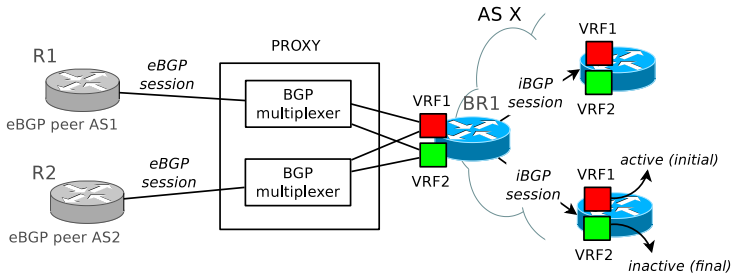


Figure 6.11 Architecture of our solution.

topology, i.e., iBGP peering sessions and routing policies can be configured on a per-namespace basis.

To run two control-planes at the same time, we would use an *initial VRF* with the initial configuration, and a *final VRF* with the final configuration. Unfortunately, because of the one-to-one mapping between interfaces and VRFs, routes learned from external peers are injected in a single VRF. This prevents independent propagation of external routes to all the VRFs, since only the best routes can be leaked from one VRF to another. A workaround to propagate all the external routes to all the VRFs is to configure multiple parallel eBGP peerings. However, this solution is unpractical as it requires coordinated configuration changes on both sides of those peerings.

To ensure correct forwarding, every router on the data path must forward packets 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 which receives the packets. An example of implicit tagging is what is commonly known as VRF-lite [160]. 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.

6.4.2 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. 6.11. The architecture of the proxy is similar to the one of BGP-Mux [136] 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 [136] 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, we envision its functionality to 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, as tagging mechanism, the proxy exploits 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 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 (resp. final) VRF over iBGP sessions that are not in the initial (resp. final) configuration.

A reconfiguration step in BGP SITN simply consists in switching the active VRF at one proxy. This has two important consequences: first, packets from eBGP neighbors start to be forwarded according to the active VRF; second, the proxy starts announcing the routes in the active VRF to its eBGP neighbors. As we have one proxy per border router, the reconfiguration can be performed one border router at a time.

6.4.3 Key benefits

A primary benefit of our solution is that it guarantees seamless BGP migrations, as proved by the following theorem.

Theorem 6.1. *BGP SITN ensures seamless migrations.*

Proof. First, we consider control-plane anomalies. BGP SITN ensures that the two control-planes run network-wide in isolation, meaning that routes received by each router in each control-plane coincide with the routes that the router receive in either the initial or in the final configuration. Also, the selection of the active VRF on each router has no impact on any of the two control-planes. Hence, absence of control-plane anomalies follows from the assumption that both the initial and the final configurations are correct.

Regarding data-plane anomalies, both deflections and unintended traffic shifts are prevented by the tagging mechanism. Indeed, whatever is the active VRF on any proxy, the tagging mechanism (e.g., MPLS or VRF-lite) ensures that every router in the network will use the same VRF to forward the packet. Hence, any data packet will be forwarded over a forwarding path which is either the path used in the initial configuration or the one followed in the final configuration. The correctness of the initial and final configurations ensures no data-plane anomalies. □

Our approach is also suitable to deal with routing and forwarding issues that we have not considered in this chapter for the sake of simplicity. Primarily, our approach requires no extra logic to adapt to control-plane and data-plane changes. Indeed, thanks to the proxy and the isolation principle, both SITN control-planes react to routing changes independently. Also, issues due to specific settings of the `MED` attribute are avoided in our approach because of the isolation of the initial and the final control-planes, that are assumed to be anomaly-free. Finally, our approach prevents transient anomalies, like protocol convergence issues, caused in the incremental approach by the reconfiguration of single routers. Indeed, the two control-planes running in BGP SITN do not need to converge at each migration step, but only before starting the reconfiguration and possibly after external changes like eBGP or data-plane changes.

The main drawback of our approach is the additional load imposed to routers as they have to support two control-planes. In Section 6.5 we show that current ISP-targeted routers can sustain this additional load. For legacy network devices and to narrow the additional iBGP churn due to two control-planes, we propose to divide the reconfiguration in chunks, so that each chunk consists in the reconfiguration of a different group of prefixes. Indeed, the design of the proxy can be easily adapted to support different active VRFs for different groups of prefixes. We evaluate the trade-off between quickness and scalability of our approach in Section 6.5.

6.5 Evaluation

In order to show the feasibility and effectiveness of our solution, we implemented a prototype that can perform seamless reconfigurations. We use our prototype to perform a use case, and we evaluate the scalability our

solution. Finally, we qualitatively compare our approach with alternative proposals.

6.5.1 Implementation

The system is based on an extended version of our provisioning system (see Chapter 8) to which we added support for VRFs and route-maps. At each migration step, our system reconfigures one border router by interacting with the corresponding proxy and switching the active VRF on it.

We implemented the proxy as a standalone script of about 400 lines in Perl. Our prototype proxy has some known limitations: first, it requires the ability to define logical interfaces on the border router; second, it requires a shared layer 2 infrastructure between the proxy, the external neighbor and the border router. However, these limitations could be easily avoided if the proxy component was directly integrated in the router operating system. Given the simple architecture of the proxy, we believe such an integration to be possible on commercial routers.

6.5.2 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 consistently with the layer 2 topology of Geant [168]. 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 [156]. 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 has two route reflectors belonging to the layer immediately above, mimicking redundancy best practices.

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 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 we

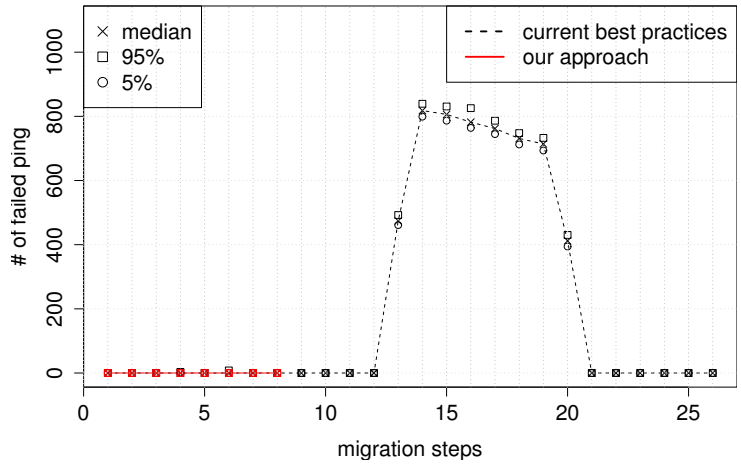


Figure 6.12 Using our system, no packet was lost when converting the Geant network from an iBGP full-mesh to a route-reflection hierarchy. On the contrary, significant traffic losses occurred with current best practices.

added final iBGP sessions to the iBGP configuration. Two route-maps per router ensured correct propagation of routes on the initial and final iBGP topologies. Then, we proceeded one border router at a time. To migrate a border router, we activated the final VRF on the proxy. When the final VRF is used on all the border routers, we remove the initial iBGP sessions, the initial VRFs and both route-maps from the routers. In the second experiment, we followed the current best practices [198, 68]. In particular, for each router to be migrated, we first 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. 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 each router towards each summary prefix throughout the migration process.

Fig. 6.12 reports the median, and the 5th and 95th percentiles of ICMP packets lost during each migration step. The number of migration steps are fewer in our approach than in current best practices, since the new configuration is used network-wide after the reconfiguration of the border routers (only a configuration cleaning step is needed for the other routers in the network).

Using our framework, no packet was lost, 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.

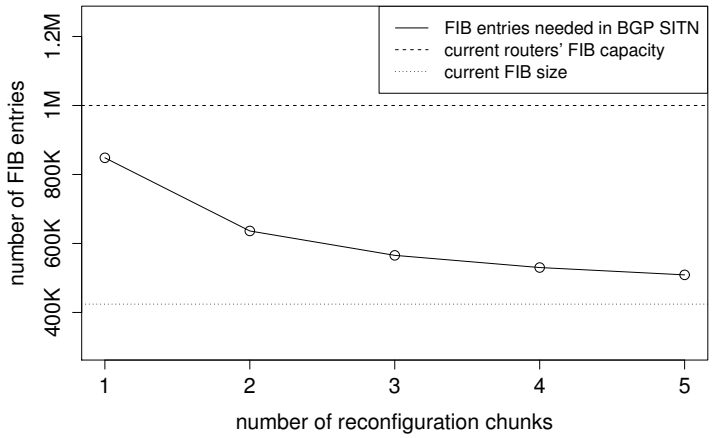


Figure 6.13 Scalability evaluation of BGP SITN with respect to FIB size.

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 shows the advantage of relying on our framework, as it provably avoids packet losses (see Theorem 6.1) that can affect traffic to a significant portion of the full routing table during several reconfiguration steps.

6.5.3 Scalability

We now estimate the overhead of our approach in terms of additional router memory and CPU processing power needed to maintain two control planes. Regarding memory, we focus on the FIB size as routers' RIBs can be easily scaled by adding low cost RAM components. On the other hand, CPU overhead is mostly due to computing the BGP best path twice (once for each control plane), which increases the iBGP churn, i.e., the number of iBGP updates.

Although sharing memory and data structures across multiple VRFs might be a significant performance improvement, we find that routers currently store a separate copy of the RIB and the FIB for each VRF. Hence, activating BGP SITN would double the number of entries in the FIB. This is not a problem for current routers, as shown in Fig. 6.13. Indeed, the FIB capacity of routers was estimated in at least 1 million FIB entries in 2009 [8] (see the horizontal dashed and dotted line in Figure 6.13), while the current FIB size of a typical Internet router is about 425,000 FIB entries as of 28 May 2012 [194]. Moreover, the FIB overhead can be reduced by dividing the prefixes in n groups and migrating one group at a time, as described in Section 6.4.3. This way, at each migration step, the total number of FIB entries will be $(1 + 1/n)$ times the original FIB size, thus reducing the amount of additional memory needed, at the cost of multiplying the number of migration steps by n. The dashed line with circle

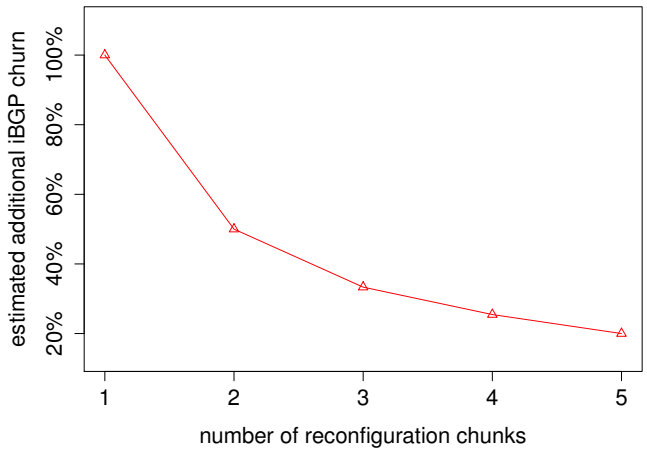


Figure 6.14 Scalability evaluation of BGP SITN with respect to churn.

points in Fig. 6.13 shows that a good trade-off can be achieved for $n = 2$ or $n = 3$.

Very similar considerations hold for BGP churn: by grouping prefixes in n sets, we can trade speed of the migration process for better scalability. We performed the following analysis. Since both the initial and the final configurations are correct, the number of iBGP updates generated by a single eBGP update must be finite. To be independent of the given iBGP topologies, we assumed that the iBGP churn is proportional to the eBGP churn. Observe that this is a pessimistic assumption as several eBGP updates could generate no iBGP update in one of the two control planes or both. Indeed, no iBGP updates are generated when the received eBGP update does not change the best route selection at the egress point. We collected all the eBGP updates from [194] during May 2012, and we estimated the additional churn introduced by SITN with respect to the churn generated in a single control plane. We used a simple greedy heuristic to divide prefixes in n groups: iteratively, we picked the most "churny" prefix not yet assigned to any group and we added it to the group having the least total number of BGP updates. Fig. 6.14 shows the result of such an analysis for a single route collector from [194] (similar results hold for other collectors). If $n = 1$, then the iBGP churn in SITN is the sum of the churn in each control plane. In the worst case, iBGP churn is equal in the initial and final control planes, and the churn increase due to SITN is 100%. However, even such a simple approach can lead to a good trade-off: the red solid line in Fig. 6.14 shows that the additional BGP churn generated by BGP SITN quickly drops as n increases.

Our results suggest that BGP SITN can be deployed in today's networks. Also, operators providing MPLS VPN services already have most of the machinery in place to support BGP SITN. Others should weigh the need for network agility against the cost of configuring two control planes in their

network. We believe the long term gain in network agility can motivate operators to bear the initial deployment cost.

6.5.4 Comparison with alternative solutions

A possible alternative to our solution consists in configuring static routes that match the initial configuration and ensure consistency throughout the migration. Static routes can be eventually removed when the final configuration has been deployed. Unfortunately, this approach has severe drawbacks. First, detaching the control plane from the data plane makes the network unable to react to BGP routing changes (e.g., the withdrawal of a BGP route). Second, our results in Section 6.3 indicate that removing the temporary static routes after BGP has converged can result in forwarding anomalies. Finally, adding a significant number of static routes overcomplicates management and troubleshooting.

In principle, recently proposed techniques to simplify BGP management can be leveraged to facilitate the reconfiguration process. For example, one might think to rely on platforms that centrally compute BGP routes (e.g., [19]), or mechanisms that separate the control plane from the data plane (e.g.,[25]). On one hand, this approach simplifies the understanding of intermediate routing states, and avoids control plane anomalies. On the other hand, however, it suffers from problems intrinsic to centralized approaches, especially scalability and resiliency of the centralized platform, and difficulties to quickly react to data plane changes (e.g., link and router failures). Moreover, deploying the centralized component while avoiding routing and forwarding inconsistencies can be seen as just another instance of the seamless reconfiguration problem.

6.6 Related work

Considerable effort has been devoted to BGP configuration correctness [66, 44, 87] and iBGP topology design [18, 114, 147]. 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.

A general approach to deal with multiple configurations is proposed in [4]. In that work, Alimi *et al.* propose firmware modifications that enable routers to manage a shadow configuration beyond the active configuration used to data traffic forwarding. Shadow and active configurations can be switched using an ad-hoc commit protocol. The entire approach can be seen as a way to implement two BGP control-planes. However, our solution is more lightweight and easier to implement with respect to [4], as it requires no ad-hoc protocol for either tagging packets and committing configuration changes. Also, we justified the need for an additional control-plane to solve the BGP reconfiguration problem by a thorough theoretical study.

Graceful session reset is tackled in [54]. Also, Route Refresh and BGP Soft-Reset capabilities are standardized in [24]. In contrast, we aim at enabling iBGP and eBGP reconfigurations which are not restricted to single BGP peerings and can affect several routers in a network.

Recently, some techniques [151, 75] have been proposed to enable virtual routers or parts of the configuration of BGP routers (e.g., BGP sessions) to be moved from one physical device to another. Their works differ from ours as we aim at changing network configurations.

In [117], 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).

6.7 Conclusions

Network operators regularly change router configurations. BGP reconfigurations are no exception, 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.

In this chapter, we show that routing and forwarding anomalies, possibly resulting in severe losses, can occur during BGP reconfigurations, even when MED is not used and simple policies are deployed. Unfortunately, current best practices 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.

To avoid routing and forwarding anomalies, we study the problem of finding an operational ordering so that all intermediate configurations are anomaly-free. Unfortunately, the problem of deciding whether such an ordering exists is computationally intractable. Also, we show several cases where such an ordering simply does not exist.

To provide a solution in the general case, we propose a solution that provably enables lossless BGP reconfigurations by leveraging existing technology to run multiple isolated control-planes in parallel. We describe an implementation of this framework, evaluate its scalability, and illustrate its effectiveness through a case-study.

Our findings show that achieving lossless BGP reconfigurations is a hard problem in the general case. However, there might exist specific cases that can be performed safely, without relying on multiple control-planes. Understanding what kinds of reconfigurations can be carried out under what assumptions is an interesting open problem.

Part IV

Combining intradomain and interdomain routing protocols reconfiguration

Chapter 7

Impact of IGP reconfiguration on BGP

7.1 Introduction

In a typical ISP network, several network protocols depend on the IGP to discover information about the network topology. For instance, signaling protocols such as the Label Distribution Protocol (LDP) [5] or the Resource Reservation Protocol (RSVP) [7] depend either entirely, or partially on the IGP in order to establish MPLS label-switched paths across the network. Another example is the Protocol Independent Multicast (PIM) [46] used to route multicast traffic which depends on the IGP to build the multicast tree.

Any protocol depending on the IGP can be negatively impacted if the underlying IGP reconfiguration is disruptive. However, even if the underlying IGP reconfiguration is disruption-free, it is not guaranteed that anomalies will not occur in the protocols depending on the IGP. This is especially true for BGP. When selecting among several equally preferred routes, a BGP router will prefer the ones whose next-hop is closer in terms of IGP distance (see step 6 in Table 1.1). Reconfiguring the IGP can thus change the BGP decisions taken by a router. Given that the IGP reconfiguration is progressive, migrated routers will base their BGP decisions on the new IGP, while non-migrated routers will base their decisions on the old IGP. This can lead to situations in which migrated and non-migrated BGP routers disagree about which BGP path to use to reach some destinations. These disagreements can create BGP-induced routing, dissemination and forwarding anomalies.

An example of IGP reconfiguration in which a BGP-induced forwarding loop can be created is depicted in Fig. 7.1. The considered scenario is a flat2hierarchical reconfiguration where routers rr_1 and rr_2 become ZBR. From the BGP point of view, routers rr_1 and rr_2 act as route reflectors for e_1, e_2, and e_i. An equally preferred BGP prefix p_1 is also learned on e_1 and e_2. We adopt the same graphical conventions as in Chapter 6 to describe iBGP topologies. Regarding IGP topologies, we represent a backbone link with a dashed line, and an intra-zone link with a plain one. Unless the con-

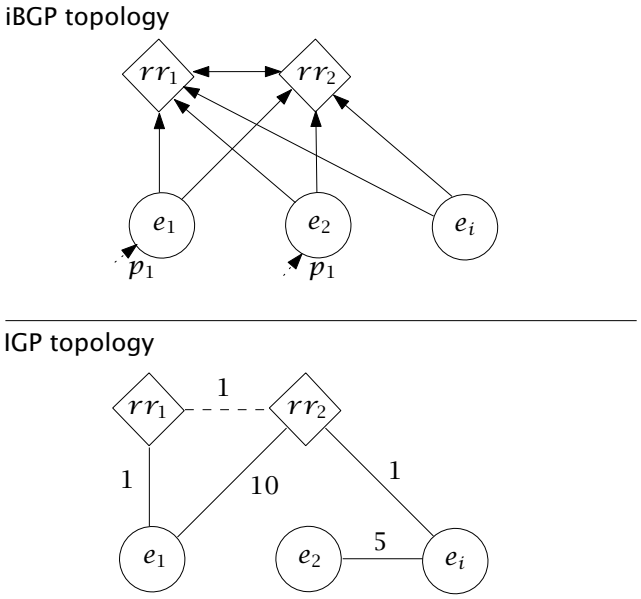


Figure 7.1 Myopia gadget. Migrating the IGP can create BGP-induced forwarding loops. In this flat2hierarchical scenario where rr_1 and rr_2 become ZBR, reconfiguring rr_2 before e_i creates a BGP-induced loop for destination p_1.

trary is explicitly stated, we always assume that the IGP reconfiguration is performed by following the techniques described in Chapter 3 (i.e., SITN).

First, observe that the BGP configuration is loop-free under the initial and the final IGP configuration. In the initial IGP configuration, every router but e_2 selects e_1 as their best egress point for reaching p_1. Indeed, both rr_1 and rr_2 prefer e_1 over e_2 which implies that e_i only learns about e_1. In the final IGP configuration, rr_1 selects e_1 as before, but rr_2 and e_i now select e_2. Indeed, in a hierarchical IGP, we have $dist(rr_2, e_2) = 6$ which is smaller than $dist(rr_2, e_1) = 10$.

Consider now the reconfiguration process. If we consider only IGP destinations, any ordering such that $e_1 < rr_1$ is disruption-free. This ordering is needed to avoid the loop towards rr_2. However, this constraint is not sufficient to prevent the creation of BGP-induced forwarding loops. Indeed, if the ordering is such that $rr_2 < e_i$, a forwarding loop is created towards p_1 as rr_2 prefers $(rr_2\ e_2)$ while e_i prefers $(e_i\ rr_1\ e_1)$. This forwarding loop is due solely to the presence of BGP and the fact that a BGP prefix can be learned from any subset of egress points. This example illustrates the necessity of taking into account both IGP and BGP destinations when computing ordering constraints. In this example, it is easy to see that any ordering such that $e_1 < rr_1 < e_i < rr_2$ is loop-free for both IGP and BGP destinations.

In this chapter, we study the *combined reconfiguration problem* which consists in reconfiguring a link-state IGP in such a way that no anomaly is created at the IGP and at the BGP levels. Our contribution is manifold. First,

we show that reconfiguring the IGP can create any BGP-induced anomalies. Such anomalies can happen even if an anomaly-free IGP ordering is followed and an iBGP full-mesh is used. By simulating different IGP reconfigurations of a Tier-1 network, we found that numerous BGP-induced anomalies can appear and last for a significant part of the IGP reconfiguration. Unfortunately, we also show that a combined reconfiguration ordering does not always exist, even if a per-destination ordering is followed. Furthermore, we show that deciding if a combined reconfiguration ordering exists is computationally intractable ($\mathcal{NP}$-hard). Since an algorithmic approach is not viable, we discuss how to solve the combined problem by leveraging configuration guidelines and by extending the reconfiguration framework presented in Chapter 6.

The rest of the chapter is organized as follows. Section 7.2 formalizes the combined reconfiguration problem. Section 7.3 evaluates the amount of BGP disruption that can be created during typical IGP migration scenarios. Section 7.4 shows that migrating the IGP can trigger any type of BGP-induced anomalies unless a proper ordering is followed. It also shows examples where such an ordering does not exist. Section 7.5 discusses the use of other reconfiguration techniques beyond SITN and shows that they are also likely to create BGP-induced anomalies. Section 7.6 discusses the computational complexity of the combined problem with respect to the complexity of the constituting sub-problems. Finally, Section 7.7 describes how IGP design guidelines and the usage of our reconfiguration framework can help solving the combined reconfiguration problem.

7.2 Problem statement

All the concepts needed for modeling combined reconfiguration were introduced in the previous chapters. In the following, we denote with I_t the IGP configuration (as defined in Chapter 3) at step t and with B_t the corresponding BGP configuration (as defined in Chapter 6) at the same step. As before, we define two special indices i and f to refer to the initial and final configurations. We assume I_i, I_f, and B to be given as input and to be anomaly-free. We refer to a combined IGP and BGP configuration with the couple (I, B). Also, we define a combined reconfiguration scenario as $(I_i, B_i) \rightarrow (I_f, B_f)$. In a pure IGP reconfiguration case, we have $(I_i, B) \rightarrow (I_f, B)$, while in a pure BGP reconfiguration case, we have $(I, B_i) \rightarrow (I, B_f)$.

Intuitively, the *Seamless Combined Migration Problem* (SCMP) consists in progressively reconfiguring the IGP running in a BGP-enabled network without creating forwarding loops for IGP destinations and, at the same time, gracefully accommodate the corresponding changes in the BGP path preferences. We define it as follows:

Problem 7.1 (Seamless Combined Migration Problem)**.** *Given an initial and a final IGP configuration which determine two next-hop functions, compute a router migration ordering, if any, such that, (i) no forwarding loop arises*

for any IGP destination learned by the IGP, and such that, (ii) for any intermediate migration step, the corresponding BGP configuration is:

- *oscillation-free;*
- *LoV-free;*
- *deflection-free;*
- *not subject to unintended traffic shifts.*

7.3 Quantifying the impact of IGP reconfigurations on BGP

In this section, we study the impact of the interactions between graceful IGP operations and BGP by running several experiments on a Tier-1 network. In each experiment, we simulated the reweighting of few IGP links. The IGP reconfiguration is performed using the per-router reconfiguration technique proposed in Chapter 3 which provably avoids forwarding loops to any IGP destination.

The Tier-1 network we considered consists of more than 100 routers and more than 150 links. BGP route reflection [11] is configured on the network, and BGP routers are arranged in a three-layer route reflection hierarchy. In addition to the configurations of all routers, our data set includes a dump of all the BGP routes received by the route reflectors at the top layer.

We simulated three reconfiguration scenarios. In the first scenario, we reweighted 5 links ($\approx$ 3% of all links), in the second scenario we reweighted 10 ($\approx$ 6%) links, and in the third scenario we reweighted 15 links ($\approx$ 10%). These scenarios are meant to capture typical reconfigurations performed by network operators to achieve better traffic engineering while minimizing the number of reweighted links [51]. For each scenario, we performed 30 experiments. In each experiment, we randomly chose the reweighted links. We also randomly chose the new weight to be assigned to the selected links within the set of weights used in the initial configuration.

We used SimBGP [192] to compute the forwarding tables during the reconfiguration. To reduce the number of BGP prefixes used in the simulation, we group BGP prefixes into virtual prefixes. Namely, we univocally map a virtual prefix to a combination of routers that, in our data set, injected BGP routes to the same BGP prefix. We stress that each virtual prefix typically corresponds to several BGP prefixes since several BGP prefixes can be injected by the same set of routers. Overall, we identified 1500 virtual prefixes. Each of them corresponds to 102 real BGP prefixes on average, with a minimum of 1 and a maximum value of 13650. After each router reconfiguration, we waited for BGP convergence and we analyzed the resulting forwarding tables to identify BGP forwarding loops towards virtual prefixes.

We found that numerous BGP forwarding loops can appear during the reconfiguration process. Fig. 7.2 plots the fraction of experiments experi-

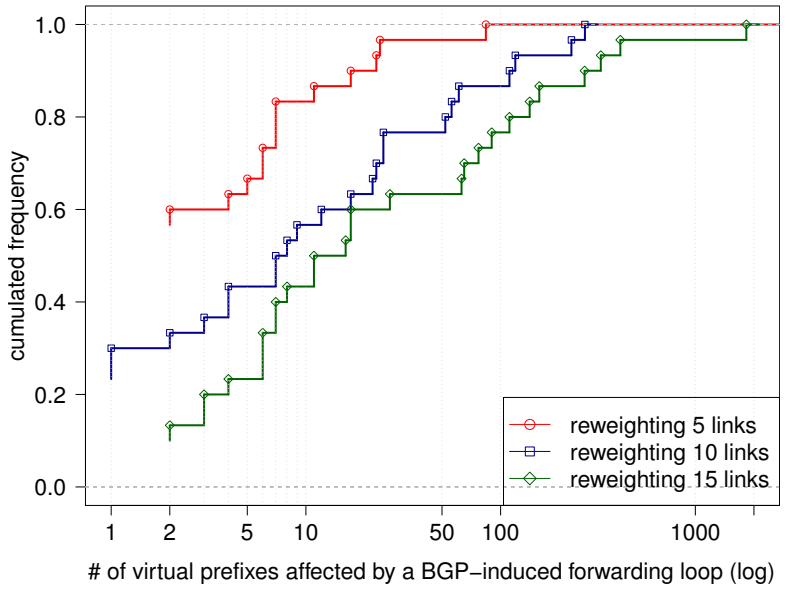


Figure 7.2 Numerous BGP-induced forwarding loops can appear during IGP changes, even when state of the art techniques are applied.

	5 links	10 links	15 links
Average loop duration (% of process)	23.80	17.14	7.61
Average number of routers involved	5.33	5.33	7.90
Total number of routers involved	32.00	42.00	49.00
Maximum size of a loop (# routers)	2.00	2.00	8.00
Average size of RT impacted (%)	1.16	2.48	7.83
Total size of RT impacted (%)	15.00	31.00	97.00

Table 7.1 BGP-induced loops are long-lived, involve multiple routers and impact a significant part of the BGP Routing Table (RT).

encing a given amount of BGP-induced loops. A data point (x, y) in the graph means that $(100 * y)\%$ of the experiments exhibited x forwarding loops. When reweighting 5 links, BGP-induced forwarding loops happened for at most 85 virtual prefixes in the worst case, and for at least 2 virtual prefixes in more than 40% of the experiments. We stress that this is significant as each virtual prefix can potentially map to a large number of actual BGP prefix. When reweighting 10 (resp. 15) links, the likelihood that at least one virtual prefix experienced a BGP-induced forwarding loop was more than 70% (resp. 90%) of the experiments. In the worst case, we observed forwarding loops for more than 1800 virtual prefixes, indicating that several virtual prefixes were affected by different forwarding loops at different stages of the reconfiguration.

Besides potentially affecting a large number of prefixes, loops for BGP destinations can last during several steps of the reconfiguration process. In our experiments, some loops lasted for more than 20% of the entire reconfiguration process (Table 7.1). While all the forwarding loops raised when reweighting 5 and 10 link involved 2 adjacent routers, we found some cases where as many as 8 routers were involved when 15 links are reweighted. For each impacted virtual prefix, we also accounted the number of real prefix impacted. We discovered that a significant fraction of the routing table can be impacted. Indeed, close to 8% of the RT was subject to at least one loop on average when 15 links were renumbered. Finally, we observed that different reconfiguration orderings created BGP loops toward different prefixes. For example, almost all virtual prefixes (97%) were impacted at least once in all the experiments we did for the 15 link reweighting scenario.

7.4 BGP disruptions due to SITN IGP reconfiguration techniques

In this section, we show that migrating the IGP relying on SITN routing can trigger any type of BGP anomalies unless a proper ordering is followed. To compute this ordering, we show that it is necessary to consider the IGP and the BGP destinations *at the same time* and not separately as we did previously. Unfortunately, we also show that there are cases in which such ordering does not exist. In particular, we describe different IGP reconfiguration scenarios in which: (i) any reconfiguration ordering is not deflection-free (loop-free); (ii) any reconfiguration ordering is not oscillation-free; (iii) any reconfiguration ordering is not dissemination correct; and (iv) any reconfiguration ordering is subject to unintended traffic shifts. Without loss of generality, we consider only flat2hierarchical IGP reconfiguration scenarios. We experimentally confirmed that the anomalies occurred in all the following scenarios by performing them in a virtual environment using a major router vendor operating system image.

7.4.1 Unavoidable BGP-induced deflections and forwarding loops

The divorce gadget depicted in Fig 7.3 is an example of reconfiguration scenario in which a safe ordering exists at the IGP and at the BGP level, but they are incompatible. The reconfiguration scenario is such that rr_1, rr_2, rr_3 and $rr4$ become ZBR. The iBGP topology is a full-mesh and e_1 and e_3 act as egress points for prefix p_2. In I_i (resp. I_f), every router but e_3 (resp. e_1 and rr_1) selects e_1 (resp. e_3) to reach p_2.

First, consider the IGP destination e_j. To avoid forwarding loop, the ordering constraint $rr_2 < e_i < rr_3$ must be respected. Indeed, (i) rr_2 switches from using rr_3 to reach e_j to use its direct link, (ii) e_i switches from using rr_3 to use rr_2, and (iii) rr_3 switches from using rr_4 to use e_i.

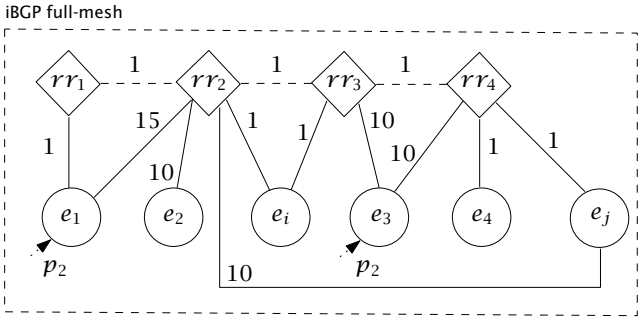


Figure 7.3 Divorce Gadget. No combined migration ordering is loop-free for flat2hierarchical scenarios even if there exists a safe ordering at the IGP and at the BGP level.

However, consider now the BGP destination p_2 where the opposite ordering constraint $rr_3 < e_i < rr_2$ holds. Indeed, (i) rr_2 switches from using rr_1 to use e_i, (ii) e_i switches from using rr_2 to rr_3, and (iii) r_3 switches from using rr_2 to use e_3 directly. Clearly, these two ordering constraints are contradicting with each other and cannot be satisfied simultaneously.

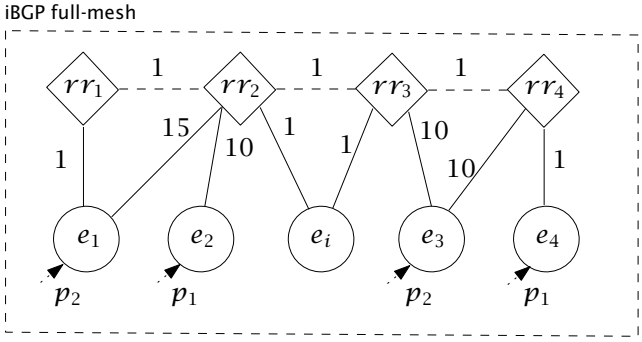


Figure 7.4 Roller-Coaster Gadget. Migrating the IGP can create migration loops that cannot be avoided at the BGP-level even when a full-mesh of iBGP session is used.

In the previous example, an ordering was needed at both the IGP and the BGP level and these two orderings were contradicting. In the following, we show that the reconfiguring IGP can create contradictory constraints at the BGP level only. As example, consider the reconfiguration scenario depicted in Fig 7.4 which is a simplification of Fig 7.3. The IGP reconfiguration scenario is such that rr_1, rr_2, rr_3 and rr_4 become ZBR. The iBGP topology is again a full-mesh. e_2 and e_4 act as egress points for p_1, while e_1 and e_3 act as egress points for p_2. In I_i, every router but e_2 (resp. e_3) selects e_4 (resp. e_1) to reach p_1 (resp. p_2).

Consider now the reconfiguration process. At the IGP level, only one ordering constraint exists: $e_1 < rr_1$ to avoid a forwarding loop towards rr_2. However, at the BGP level, one of the following two cases apply:

1. $rr_2 < rr_3$. In this case, rr_2 steadily selects $(rr_2\ e_3)$ for reaching p_2, while rr_3 keeps selecting $(rr_3\ e_1)$. Depending on the state of e_i, we distinguish two cases. First, if e_i is not migrated, then the forwarding loop $(rr_2\ e_i\ rr_2)$ occurs. Indeed, $nh_{final}(rr_2, p_2) = \{e_i\}$ while $nh_{init}(e_i, p_2) = \{rr_2\}$. Second, if e_i is migrated, the forwarding loop $(rr_2\ e_i\ rr_3\ rr_2)$ occurs. Indeed, $nh_{final}(e_i, p_2) = \{rr_3\}$ and $nh_{init}(rr_3, p_2) = \{rr_2\}$.

2. $rr_3 < rr_2$. In this case, rr_3 steadily selects $(rr_3\ e_2)$ for reaching p_1, while rr_2 keeps selecting $(rr_2\ e_4)$. Depending on the state of e_i, we distinguish two cases. First, if e_i is not migrated, then the forwarding loop $(rr_3\ e_i\ rr_3)$ occurs. Indeed, $nh_{final}(rr_3, p_1) = \{e_i\}$ while $nh_{init}(e_i, p_1) = \{rr_3\}$. Second, if e_i is migrated, the forwarding loop $(rr_3\ e_i\ rr_2\ rr_3)$ occurs. Indeed, $nh_{final}(e_i, p_1) = \{rr_2\}$ and $nh_{init}(rr_2, p_1) = \{rr_3\}$.

In both cases, a forwarding loop is created at the BGP-level during the IGP reconfiguration. Notice that the iBGP topology was a full-mesh. Therefore, even a proper iBGP design is not enough to guarantee safe combined reconfiguration.

In Chapter 3, we showed that it always exists a *per-destination* ordering guaranteeing the absence of forwarding loop. Unfortunately, in combined reconfiguration, this property is not verified anymore. As illustration, consider the flat2hierarchical reconfiguration scenario illustrated in Fig. 7.5 where r_5 and r_2 become ZBR. The iBGP topology consists in a route reflection hierarchy where r_1 acts as the top-level route-reflector. In this topology, r_1, r_2, r_7 and r_8 receives an equally preferred route for prefix p_1. The destination that we consider in this scenario is r_2's loopback address. Recall that, when performing a per-destination reconfiguration, only the final path for r_2 is installed in the FIB. We also assume that each edge router "rewrites" the next-hop of an eBGP learned route with its own loopback address before announcing it in the iBGP topology. A configuration practice commonly used and known as *next-hop-self* [37].

In I_i, r_5 and r_6 receive and steadily select r_2, while r_3 and r_4 only receive r_1 and thus select it. In I_f, both r_5 and r_6 prefer the path propagated by their respective client r_7 and r_8, r_3 and r_4 also select r_7 as it is closer than r_1. In both cases, there is no BGP-induced loop.

Consider now the reconfiguration process. First, observe that r_6 must be migrated before r_5 for avoiding an IGP-induced forwarding loop towards r_2. Indeed, r_5 uses r_6 in I_f while the opposite holds in I_i. However, consider what happens on the BGP decisions when r_6 is migrated before r_5. r_6 starts preferring r_8 and sends it to r_4 which selects it over r_1. r_3 does not learn about r_8 and selects r_1 as it is the only path that it knows. Since r_4 is using r_3 to reach r_8 and vice-versa to reach r_1, a forwarding loop is created. The loop will be solved only when r_5 is migrated as both r_3 and r_4 will switch to r_7.

Although cumbersome, following a per-destination ordering is the most flexible way of performing IGP reconfiguration. This allowed us to use it

as a fallback solution when no per-router ordering existed in pure IGP reconfiguration. The fact that it does not always exist in combined reconfiguration further stresses the need for a general solution. In Section 7.7, we discuss how to extend our reconfiguration framework to take into account combined reconfiguration.

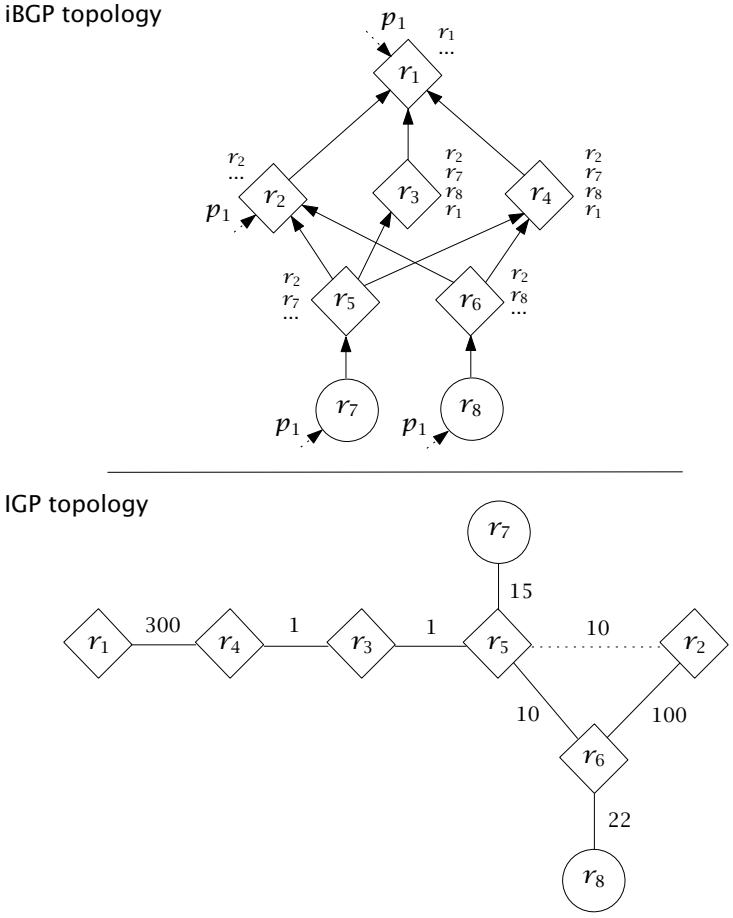


Figure 7.5 The horizontal gadget. Migrating the IGP on a per-destination basis does not guarantee the absence of BGP-induced forwarding loop. In this reconfiguration scenario, r_2 is the considered destination and contradictory reconfiguration constraints exist between r_5 and r_6.

7.4.2 Unavoidable BGP-induced routing oscillations

In addition to forwarding loops, IGP reconfiguration can create permanent BGP routing oscillations even if the BGP configuration is oscillation-free under both the initial and the final IGP configurations. As illustration, consider the IGP reconfiguration scenario depicted in Fig 7.6 where r_A and

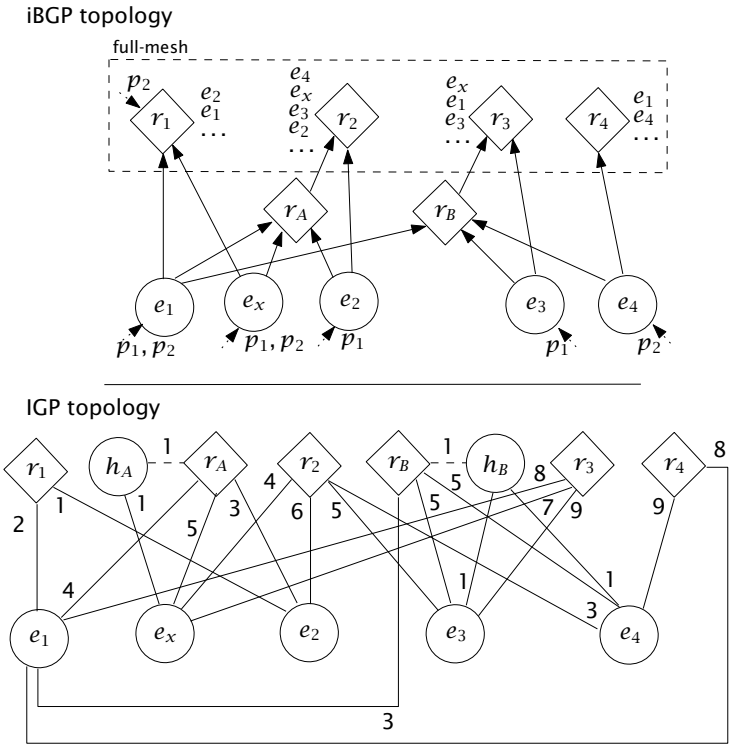


Figure 7.6 Revisited Twice-Bad. Migrating the IGP can create routing oscillations that cannot be avoided at the BGP-level.

r_B are configured as ZBR. This gadget extends the TWICE-BAD gadget described in Chapter 6 (see Fig. 6.5). In particular, as with TWICE-BAD, the gadget contains two dispute wheels, $\Pi = (\vec{\mathcal{U}}, \vec{\mathcal{Q}}, \vec{\mathcal{R}})$ and $\Pi' = (\vec{\mathcal{U}}', \vec{\mathcal{Q}}', \vec{\mathcal{R}}')$. Π relates to prefix p_1, while Π' relates to p_2. Pivot vertices in Π are $\vec{\mathcal{U}} = (r_1\ r_2\ r_3)$, spoke paths are $\vec{\mathcal{Q}} = ((r_1\ e_1)\ (r_2\ e_2)\ (r_3\ e_3))$, and rim paths are $\vec{\mathcal{R}} = ((r_1\ r_2)\ (r_2\ r_3)\ (r_3\ r_1))$. Pivot vertices in Π' are $\vec{\mathcal{U}}' = (r_2\ r_3\ r_4)$, spoke paths are $\vec{\mathcal{Q}}' = ((r_2\ r_A\ e_X)\ (r_3\ r_B\ e_1)\ (r_4\ e_4))$, and rim paths are $\vec{\mathcal{R}}' = ((r_2\ r_4)\ (r_3\ r_2)\ (r_4\ r_3))$.

B is oscillation-free under the initial and the final IGP configuration as no DW is activated. In the initial IGP configuration, r_A steadily selects e_X for both p_1 and p_2. Indeed, r_A is closer to e_X than e_1 or e_2. Since r_2 steadily receives e_X and prefers it over e_2, the spoke path $(r_2\ e_2)$ is not used and Π is prevented from oscillating. Similarly, r_B steadily selects e_4 over e_1 for reaching p_2. This prevents Π' from happening as the spoke path $(r_3\ r_B\ r_1)$ is not used. In the final IGP configuration, similar considerations hold. r_A starts preferring e_2 over e_X which in turn prevents Π' from oscillating as the spoke path $(r_2\ r_A\ e_X)$ is not used. Also, r_B starts preferring e_1 over e_3, also preferred by r_3. This prevents Π from oscillating as the spoke path $(r_3\ e_3)$ is not used.

Consider now what happens during the reconfiguration process. One of the following two cases apply:

1. $r_A < r_B$. r_A starts preferring e_2 for reaching p_1. r_A steadily selects it as best and propagates it to r_2. In this case, nothing prevents Π from permanently oscillating. Such an oscillation is interrupted only when r_B is migrated.

2. $r_B < r_A$. r_B starts preferring e_1 instead of e_3 for prefix p_2. r_B steadily selects it and propagates it to r_3. In this case, nothing prevents Π' from permanently oscillating. Such an oscillation is interrupted only when r_A is migrated.

In any case, a permanent oscillation is created.

7.4.3 Unavoidable BGP-induced dissemination anomalies

In addition to the possibility of creating forwarding loops and routing oscillations, IGP reconfigurations can also create dissemination anomalies in which some BGP routers do not learn any path (see Chapter 5). As illustration, consider the scenario of Fig. 7.7 where r_1 and r_2 become ZBR. This gadget extends the DOUBLE-CROSS gadget described in Chapter 6 (see Fig. 6.6). The reconfiguration scenario is such that both r_1 and r_2 prefer e_2 (resp. e_1) in the initial (resp. final) case.

It is easy to verify that each router has a path towards p_1 in both the initial and in the final IGP configuration. In the initial configuration, r_1 and r_2 steadily select e_2. r_2 further propagates e_2 to rr_1 and rr_2. Since rr_2 learns the path from a client, it will propagate it to rr_3. In the final configuration, r_1 and r_2 steadily select e_1. Similarly, rr_1 and rr_2 learn about it from r_1 and rr_1 propagates it to rr_3.

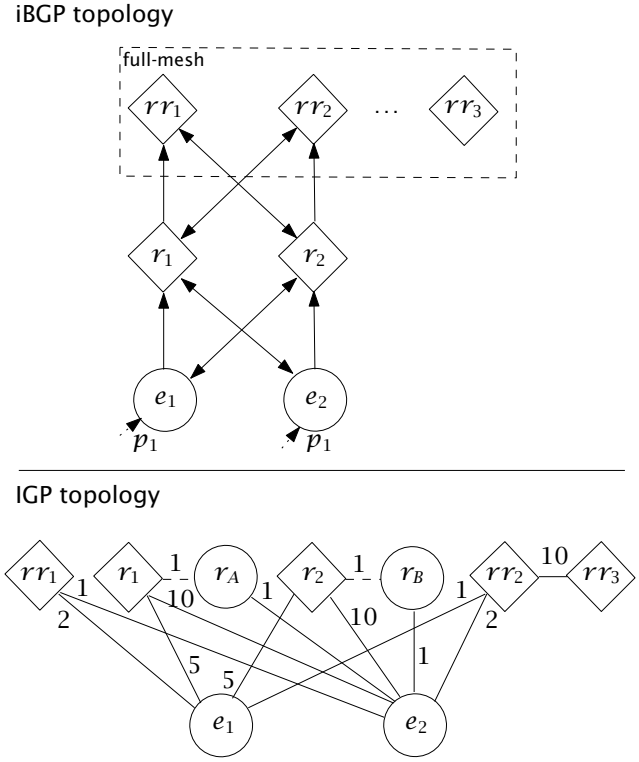


Figure 7.7 Heavy-Cross gadget, an IGP reconfiguration scenario (flat2hierarchical) in which a dissemination-free reconfiguration ordering does not exist.

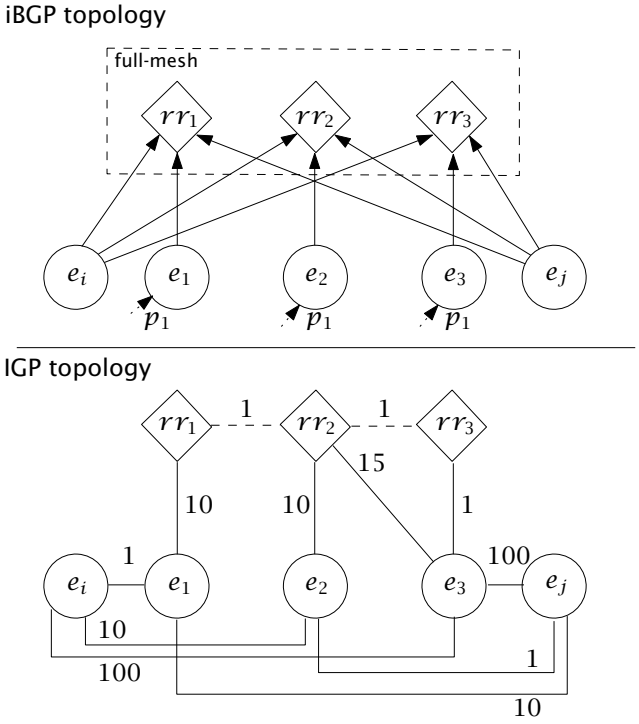


Figure 7.8 Go-With-the-Flow gadget. No IGP reconfiguration ordering can prevent BGP traffic shifts in intermediate configurations.

Consider what happens during the reconfiguration process. We have two possible cases.

1. $r_1 < r_2$. Both r_1 and r_2 prefer the route propagated by their client. They propagate that route to the top-layer. Because of egress point preferences, rr_1 (resp. rr_2) selects route (rr_1 r_2 e_2) (resp. (rr_2 r_1 e_1)), learned over an OVER session. This prevents rr_3 from learning p_1.

2. $r_2 < r_1$. r_2 starts preferring e_1, while r_1 still prefers e_2. As both prefer the path they receive over the OVER session, none of them will reflect that path, causing top layer routers to have no route to p_1.

In any case, a dissemination anomaly arises. Again, we experimentally verified the intended behavior of the gadget by performing the reconfiguration scenario in a virtual environment.

7.4.4 Unavoidable BGP-induced traffic shifts

Finally, some IGP reconfiguration can create unintended BGP traffic shifts, no matter the ordering being followed. Consider the IGP reconfiguration scenario depicted in Fig. 7.8 where rr_1, rr_2, and rr_3 become ZBR.

Under the initial IGP configuration, rr_1, rr_2 and rr_3 steadily select e_3 to reach p_1. Since e_i and e_j do not learn any other path, they will steadily select e_3 as well. Under the final IGP configuration, rr_1, rr_2 and rr_3 steadily select the path propagated by their direct clients. Consequently, e_i and e_j learn all the available paths and steadily select e_1 and e_2, respectively.

Consider now the reconfiguration process. One of the following case applies.

1. $rr_2 < rr_1$. In this case, rr_2 selects e_2 and propagates it. Because of egress point preferences, e_i will steadily select e_2, that is, a route announced by an egress point to which e_i does not send traffic neither in the initial IGP configuration, nor the final.

2. $rr_1 < rr_2$. In this case, rr_1 selects e_1 and propagates it. Because of egress point preferences, e_j will steadily select e_1, that is, a route announced by an egress point to which e_j does not send traffic neither the initial IGP configuration, nor the final.

In both cases, an unintended traffic shift occurs. Remember that in addition of being detrimental for the traffic *per se*, unintended traffic shifts increase the risk of route dampening [142] as edge routers may have to propagate the path changes outside of the AS via eBGP UPDATES.

7.5 BGP disruptions due to other IGP reconfiguration techniques

So far, we have only considered SITN reconfiguration techniques. However, other reconfiguration techniques exist beyond SITN (see Chapter 3). Most of these techniques consists in progressively changing routers' forwarding tables so as to minimize or to avoid disruptions (e.g., [56, 57, 126, 4, 116]).

In this section, we show that such techniques are also prone to induce BGP anomalies. In particular, we focus on the metric-increment [56] technique as it is a representative technique which does not require modification to current router implementation. Moreover, we restrict our attention to forwarding anomalies. This assumption is without loss of generality as corresponding reconfiguration scenarios can be created considering signaling or dissemination anomalies.

Metric-increment [56] is a technique that avoids transient loops during link reweighting. As an illustration, consider the IGP topology depicted on the left side of Fig. 7.9 where the link (r_4, r_6) has to be shut down for maintenance reasons. To reduce convergence delay, network operators usually prefer to first reroute traffic out of the link by increasing its weight to a pseudo infinite value before actually shutting down the link [131]. However, if a network operator simply modifies the link weight in a single step,

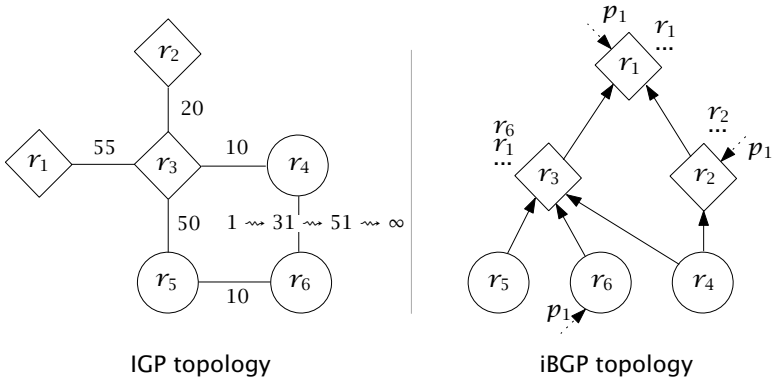


Figure 7.9 Scissors gadget. Applying the metric-increment technique to avoid transient IGP loops cause forwarding loops to BGP destinations.

transient loops for IGP destinations might appear. For instance, depending on the message timing, a transient loop can arise between r_5 and r_6 for packets destined to r_3. Indeed, as soon as r_6 becomes aware of the link weight change, it starts forwarding to r_5 all the packets destined to r_3. If r_5 still relies on the old topological information, it will bounce back these packets as r_6 was on the shortest path from r_5 to r_3 before the link was reweighted.

The metric-increment technique consists in iteratively incrementing the link weight. At each intermediate step, the metric on the link is incremented in such a way that some of the routers that have shortest paths traversing the link will be able to select a better alternative without causing any loops. At the end of the sequence, no shortest path traverses the link and the reweighting process is complete. Interestingly, a loop-free weight increment sequence always exists [56]. In Fig. 7.9, the minimal sequence of weight assignment that prevents transient loops is $\{1 \rightsquigarrow 31 \rightsquigarrow 51 \rightsquigarrow \infty\}$. For example, setting the weight of link (r_4, r_6) to 31 prevents the previously described loop between r_5 and r_6. Indeed, this step forces r_5 to change its next-hop to r_3 *before* r_6 starts forwarding packets to r_5 as the shortest path from r_6 is still $(r_6\ r_4\ r_3)$.

Unfortunately, iteratively incrementing link weights can create loops for BGP destinations. Even worse, this can happen even when both the initial and final configurations are known to be free from anomalies. Consider the iBGP topology on the right side of Fig. 7.9. The iBGP topology is a route reflection hierarchy in which r_1 is the top-layer route reflector, while r_1, r_2 and r_6 are egress points for prefix p_1.

We now describe the impact of the IGP reconfiguration process on BGP prefix p_1. As soon as the link weight is incremented to 31, a BGP-induced forwarding loop is created between r_3 and r_4. Indeed, r_4's best egress point for p_1 is now r_2. In contrast, r_3 does not learn r_2 due to iBGP propagation rules, hence it still uses r_6 as its egress point. Therefore, r_3 will forward packets to r_6 via r_4, while r_4 will send packets to r_2 via r_3, causing a for-

warding loop. This loop disappears when the link weight is incremented from 31 to 51 as r_3 starts preferring r_1 over r_6. Observe that a BGP-induced packet deflection persists in the final state as r_4 will send traffic to r_2 via r_3, while r_3 will deflect traffic to r_1. However, as this situation does not disrupt traffic, operators could be willing to tolerate it during the maintenance of link (r_4, r_6).

7.6 Problem complexity

In this section, we study the computational complexity of one decision problem associated to SCMP: oscillation-free migration. Indeed, a combined migration cannot be seamless if it is not free of BGP oscillations (see Section 7.2). In particular, we consider the following problem:

Problem 7.2 (Avoid Oscillation Problem - AOP)**.** *Given a BGP topology and two IGP topologies, decide if any IGP reconfiguration guarantees no BGP oscillations in all the intermediate configurations.*

Unfortunately, we show that AOP is computationally hard (i.e., $\mathcal{NP}$-hard). Moreover, as our proof can be adapted to dissemination and forwarding issues, deciding if an IGP reconfiguration raises any type of BGP anomalies is also computationally hard.

Our proof is divided in two parts. In the first part, we show that specific IGP reconfigurations can induce the change of the most preferred egress point on some iBGP routers. In the second part, we show that deciding if such changes can lead to BGP oscillations during the reconfiguration is $\mathcal{NP}$-hard.

7.6.1 IGP reconfigurations can cause BGP preference changes

Let $\mathcal{E}$ be the set of egress points of a given iBGP network. Let $\lambda_i^r(e)$ ($\lambda_f^r(e)$) be the position of egress point e in the initial (final) preference list of router r, where the most preferred egress point has position 1.

We now describe an IGP reconfiguration problem in which, at each step, a single BGP router swaps the positions of the two most preferred egress points. Namely, the IGP reconfiguration has three properties:

1. the initial (final) IGP topology is consistent with the initial (final) egress point preferences;
2. at each reconfiguration step, a single router r changes its preferences from λ_i^r to λ_f^r. Any other router $r' \neq r$ is not affected by the reconfiguration step; and
3. for some router r and egress points e_1 and e_2, $\lambda_i^r(e_1) < \lambda_i^r(e_2) \Leftrightarrow \lambda_f^r(e_2) < \lambda_f^r(e_1)$ if e_1 and e_2 are the two most preferred egress points of r, and $\lambda_i^r(e_1) < \lambda_i^r(e_2) \Leftrightarrow \lambda_f^r(e_1) < \lambda_f^r(e_2)$ otherwise. All the other routers have the same egress point preferences in the initial and final configurations.

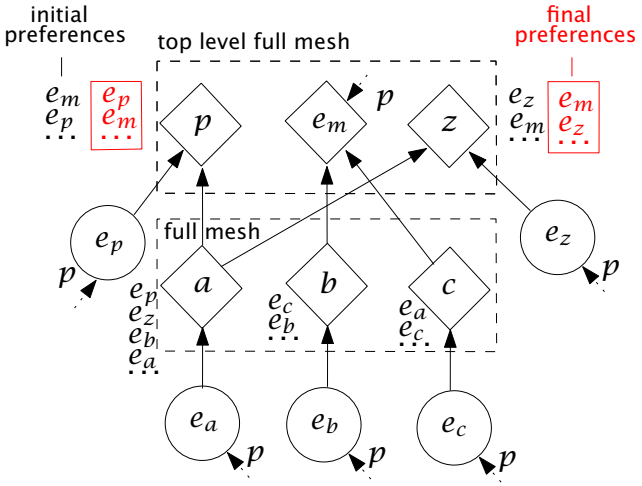


Figure 7.10 Basic structure for our reduction.

We define the initial and final IGP topologies as follows. In both topologies, we have a link (r, e) between any router $r \notin \mathcal{E}$ and any egress point $e \in \mathcal{E}$. The weight of link (r, e) in the initial configuration is $w_i(r, e) = \lambda_i^r(e) + 3|\mathcal{E}|$. In the final configuration, $w_f(r, e) = 1 + 2|\mathcal{E}|$ if $\lambda_f^r(e) = 1$, and $w_f(r, e) = w_i(r, e)$ otherwise. This weight assignment directly ensures Property 3.

Also, such IGP topologies ensure that the shortest path between any router r and any egress point e is $(r\ e)$ in any intermediate configuration (including the initial and the final ones). Indeed, consider any path $P \neq (r\ e)$ between r and e. By definition, P must contain at least two links, hence its weight in any configuration i is $w_i(P) \geq 2 + 4|\mathcal{E}|$. Thus, $w_f(r, e) \leq w_i(r, e) \leq 4|\mathcal{E}| < 2 + 4|\mathcal{E}| \leq w_i(P)$, which also ensures Property 1.

Finally, Property 2 holds since there is a one to one mapping between each edge and one shortest path, hence changing the weight of an edge affects the preferences of a single router.

7.6.2 AOP is $\mathcal{NP}$-hard

To prove that AOP is $\mathcal{NP}$-hard, we now reduce the 3-SAT problem [101] to AOP. Fig. 7.10 and 7.11 depict the reduction from a boolean formula F to a reconfiguration instance $B(F)$. Observe that $B(F)$ can be the result of an IGP reconfiguration, as described in the previous section.

The base BGP topology used in our reduction is represented in Fig. 7.10. Observe that a BAD-GADGET [64] Π' exists among a, b, and c. However, a's preferences are such that Π' is prevented from oscillating whenever a receives a route from e_p or e_z. Thus, Π' cannot oscillate in the initial nor in the final configuration. However, if z is reconfigured and p is not reconfigured yet, then a will not receive the routes to neither e_z nor e_p, and Π' will

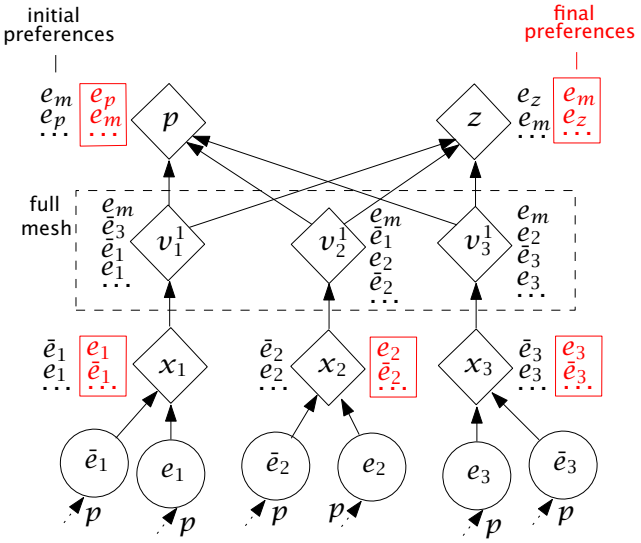


Figure 7.11 Example of the translation of a 3-SAT clause.

oscillate indefinitely. The presence of Π' hence forces any oscillation-free ordering to be such that p is reconfigured before z, which we denote as $p < z$.

The remaining part of $B(F)$ depends on the boolean formula F provided as input in the 3-SAT problem. Refer to Fig. 7.11. For each variable X_i in F, with $i = 1, \ldots, n$, we add one *variable router* x_i and two egress points e_i and $\bar{e}_i$. Egress point preferences are such that each x_i prefers $\bar{e}_i$ in the initial configuration and e_i in the final one. For each clause C_i, we add a *clause gadget* consisting of three *literal routers* v_j^i, with $j = 1, 2, 3$, representing the three literals in the clause. Observe that, since routers p and z can always reach one of their two most preferred egress points, literal routers belonging to different clauses cannot exchange paths. This allows us to consider clause gadgets separately.

For each clause C_i, a BAD-GADGET Π_i might exist among routers v_j^i. Indeed, the following property holds.

Property 7.1. *For each clause C_i, Π_i only exists if the variable routers corresponding to positive literals use their initial preferences, while the variable routers corresponding to negative literals use their final preferences.*

Moreover, since all literal routers prefer e_m over any other egress point, Π_i is prevented from oscillating when p is using its initial configuration or z is using its final configuration.

Intuitively, assigning X_i =TRUE (FALSE, resp.) corresponds to reconfiguring x_i before (after, resp.) p.

We now prove that the reduction is correct.

Theorem 7.1. *F is satisfiable if and only if an oscillation-free ordering exists on $B(F)$.*

Proof. We prove the statement in two steps.

- If F is satisfiable, then let $\mathcal{M}$ be a boolean assignment which satisfies F, and let $\mathcal{T}$ ($\mathcal{F}$, resp.) be the set of the variables that are set to TRUE (FALSE, resp.) in $\mathcal{M}$. Consider the ordering where we first reconfigure the routers corresponding to variables in $\mathcal{T}$ (in arbitrary order), then p, then z, and then the routers corresponding to variables in $\mathcal{F}$ (in arbitrary order). We now show that such an ordering is oscillation-free. Since $p < z$, BAD-GADGET Π' in Fig. 7.10 is prevented from oscillating. Also, for any migration step s, one of the following two cases applies: i) if p is not reconfigured yet or z is already reconfigured, then either p or z selects a path from e_m, preventing all BAD-GADGETS Π_i from oscillating; ii) s is the step in which p is reconfigured and z is still not. Consider any clause C_i and let l be one of the literals that satisfies C_i in $\mathcal{M}$. By construction of the reconfiguration ordering, if $l = X_i$ then router x_i is already migrated at step s. Otherwise, $l = \bar{X}_i$ and router x_i has not yet been migrated. In both cases, no BAD-GADGET Π_i exists at step s, because of Property 7.1. The same argument can be applied to all the clauses, so no oscillation can occur at s. Hence, an oscillation-free ordering exists.
- If F is not satisfiable, assume by contradiction that an oscillation-free ordering exists. The presence of Π' implies $p < z$ in the ordering. Consider any clause C_i and the migration step s immediately after the migration of p. Since neither p nor z select the route from e_m preventing Π_i from oscillating and we assumed that the migration ordering is oscillation-free, we conclude that Π_i does not exist at step s. Therefore, by Property 7.1, there must exist a router x_k such that either i) x_k corresponds to literal X_k in C_i and x_k is already migrated; or ii) x_k corresponds to literal $\bar{X}_k$ in C_i and x_k has not been migrated yet. In the first case, we have $x_k < p$ which maps to X_k = TRUE. Otherwise, we have $p < x_k$ which maps to X_k = FALSE. In both cases, we are able to assign a truth value to X_k that satisfies C_i. Since the same argument can be applied to all the clause gadgets, then we are able to build a boolean assignment that satisfies F, yielding a contradiction.

□

Observe that by replacing all the BAD-GADGETS in the reduction with gadgets that trigger a dissemination anomaly or a forwarding loop, we derive similar reductions. This implies that guaranteeing that an IGP migration is free from any kind of BGP anomaly is $\mathcal{NP}$-hard.

Further, observe that BAD-GADGET Π' is used just to force $p < z$. However, it is easy to force $p < z$ by means of an IGP constraint rather than on a BGP constraint (e.g., by adding an IGP destination for which $z < p$ creates an IGP loop). Hence, with a similar proof we can show that avoiding IGP anomalies *and* BGP anomalies during an IGP migration is $\mathcal{NP}$-hard.

Finally, similarly to [143], we conjecture that AOP is PSPACE-hard as checking whether an ordering is seamless cannot be done in polynomial time. Indeed, checking the correctness of a BGP configuration is known to be a $\mathcal{NP}$-hard problem (see Chapter 5).

Conjecture 7.1. *AOP is PSPACE-hard*

7.7 Towards BGP-aware IGP reconfigurations

In this section, we investigate viable approaches to perform reconfigurations that are disruption-free for both IGP and BGP destinations. First, we prove that anomaly-free reconfigurations can be achieved provided that the initial and the final configurations are correct and respect some conditions. Then, we show how to solve the problem in the general case by extending the techniques proposed in Chapter 3 and in Chapter 6.

7.7.1 Configuration guidelines

A first condition enabling graceful reconfigurations for both IGP and BGP consists in ensuring that the egress point preferences in the initial and final configurations are the same.

Theorem 7.2. *If each router has the same egress point preferences in the initial and in the final configurations, no IGP reconfiguration can trigger BGP anomalies.*

Proof. In SITN, reconfiguring a router cause it to directly switch from considering the initial IGP topology to the final one. Hence, at each reconfiguration step, the egress point preferences at each router coincide either with those of the initial or the final configuration which are the same by hypothesis. Since the BGP topology does not change, a BGP anomaly at a reconfiguration step implies that the same anomaly occurs in both the initial and the final configurations, contradicting our assumption on their anomaly-freeness. □

As Theorem 7.2 applies in few practical cases, we now develop less constraining conditions.

Interestingly, the two main sufficient conditions for routing correctness, i.e. the prefer-client condition and the no-spurious-over condition (see Chapter 5), are robust to IGP reconfigurations. Indeed, if the initial and final configuration comply with the sufficient conditions, then no IGP reconfiguration can invalidate them.

The prefer-client condition [66] requires that each route reflector prefer routes from its clients over routes from its iBGP peers or route reflectors. We now show that the prefer-client condition is robust to IGP reconfigurations. In a sense, this means that the prefer-client condition is so strong that it constrains the impact that IGP topology changes have on the BGP decision process.

Theorem 7.3. *If the initial and final configurations satisfy the prefer-client condition, then no IGP reconfiguration can trigger BGP routing anomalies.*

Proof. At each reconfiguration step, each router relies on either the initial or the final IGP weights independently from the configuration of the other routers. As the iBGP configuration does not change, each router has the same set of clients throughout the reconfiguration. Hence, a violation of the prefer-client condition at any intermediate step would result in a violation of the prefer-client condition in either the initial or the final configuration. The statement follows by noting that the prefer-client condition guarantees the absence of BGP routing anomalies. □

The theorem applies to cases in which both the initial and the final configurations enforce the prefer-client condition by conveniently set IGP weights. Also, if the prefer-client condition is enforced at the BGP level (e.g., as proposed in [95, 28]), then IGP and BGP are decoupled enough to guarantee no BGP oscillations during IGP reconfigurations.

The following theorem holds when the initial and the final BGP configurations comply with the no-spurious-over condition. As explained in Chapter 5, the no-spurious-over condition guarantees the absence of dissemination anomalies. This condition requires that only top-layer route reflectors have iBGP peering relationships, while every other pair of routers must have a client-reflector relationship.

Theorem 7.4. *If both the initial and the final configurations comply with the no-spurious-over condition, no IGP reconfiguration can trigger BGP dissemination anomalies.*

Proof. The statement follows by noting that no IGP reconfiguration adds nor removes any iBGP session, hence it cannot invalidate the no-spurious-over condition at any reconfiguration step. □

Unfortunately, sufficient conditions for forwarding correctness are less robust. Intuitively, this is because they impose strong congruence between the IGP and the iBGP topologies, hence changing IGP can lead to temporary violations. However, forwarding issues can be avoided by relying on packet encapsulation (e.g., using MPLS or IP tunnels). Intuitively, packet encapsulation breaks the dependency between IGP and BGP in the forwarding plane. Encapsulation mechanisms like MPLS are commonly deployed in many ISP networks.

Theorem 7.5. *If packet encapsulation is used network-wide, no IGP reconfiguration can trigger BGP forwarding anomalies.*

Proof. If packet encapsulation is deployed, then each packet from any source router r to any BGP destination is guaranteed to reach the egress point e that r selects in BGP. Because of the BGP decision process, e will forward the packet outside the network (provided that eBGP routes are stable), hence the statement. □

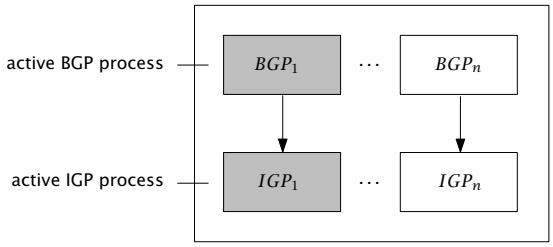


Figure 7.12 Abstract view of a combined *ships-in-the-night* framework. To gracefully accommodate combined reconfiguration scenarios, *ships-in-the-night* techniques should be available at any levels. Also, it should be possible for a protocol A depending on a protocol B to choose which instance of B it wants to depends on.

7.7.2 Extending the BGP Ships-In-The-Night framework

While the previous configuration guidelines provide correctness guarantees, they are not always achievable, nor desirable in some networks. For instance, network operators might have very good reasons to change the preference of the paths (e.g. to achieve traffic engineering objectives).

To solve combined reconfigurations when the previous configuration guidelines do not apply, we propose to extend the BGP reconfiguration framework we described in Chapter 6. In particular, in addition to having multiple isolated IGP and BGP routing processes, we need each BGP process to base its decisions on potentially different IGP processes (see Fig. 7.12). Some adjustments are required to realize that with the current implementation of our framework based on BGP MPLS/VPN techniques. Indeed, the decisions in each BGP namespace are done according to the common global routing table. To force a BGP namespace to consider only the routes of a given IGP process, we should ensure that when a BGP route is advertised in a namespace, the associated next-hop is reachable only via that IGP process. As illustration, consider the flat2hierarchical reconfiguration scenario described in Fig. 7.13 in which our reconfiguration framework is used. The IGP reconfiguration scenario is such that $B1$ and $B2$ become ZBR. Moreover, the IGP configurations are such that $e1$ and $e2$ are reachable in the flat IGP only, while $e1'$ and $e2'$ are reachable in the hierarchical IGP only. From the BGP point-of-view, $B1$ and $B2$ act as route-reflectors. A prefix p is learned on $E1$ and $E2$. $E1$ and $E2$ are configured to set $e1$ (resp. $e2$) as BGP next-hop when they propagate a route in the initial namespace. Similarly, $E1$ and $E2$ set $e1'$ (resp. $e2'$) as BGP next-hop when they propagate a route in the final namespace. Therefore, in the initial VRF, BGP decisions will be made according the flat IGP while in the final VRF, BGP decisions will be made according to the hierarchical IGP. This configuration effectively decouples BGP from the IGP as migrating the IGP will have no impact on the overlaying BGP.

While the previous technique decouples BGP from the IGP, it still requires an ordering to be found for the IGP destinations. Moreover, it also

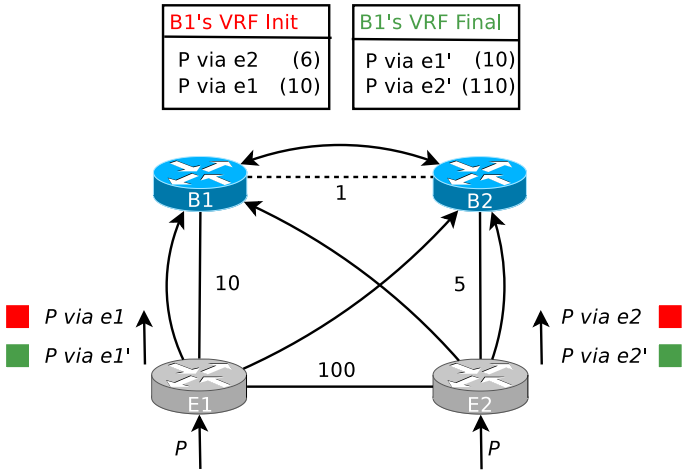


Figure 7.13 Extension of the reconfiguration framework to support for combined reconfigurations. To make two different BGP namespaces (i.e. VRFs) depend on two different IGPs, BGP next-hops can be rewritten according to the namespace such that each next-hop is only reachable via a particular IGP. In this flat2hierarchical reconfiguration, $e1$ and $e2$ are only reachable via the initial IGP and will be set as next-hops for the routes announced in the initial namespace. Similarly, $e1'$ and $e2'$ are only reachable via the final IGP and will be set as next-hops for the routes announced in the final namespace.

requires to duplicate next-hop announcements. Another way to implement a combined reconfiguration framework is to use the *Multi-topology routing* (MTR) mechanisms that are available on today's routers [201, 186]. MTR allows for the creation of multiple logical topologies running on the same underlying physical infrastructure. Each topology can run a different set of routing protocols (e.g. IGP and BGP) along with different configurations. Each BGP router performs a best-path calculation individually for each class-specific topology. To support for combined reconfiguration scenarios, we would define two logical topologies, running respectively the initial and the final IGP configurations along with the BGP configuration. As with the current framework, the migration would be performed by switching the topology responsible for forwarding on a per-router basis.

7.8 Conclusions

In this chapter, we highlighted the importance of considering the dependency between network protocols even for problems that seem to be restricted to a single protocol. In particular, we showed that state of the art IGP reconfiguration techniques should be revisited in the presence of BGP. Indeed, such techniques can create any type of *unavoidable* BGP routing and forwarding anomalies even when a few changes are made.

From the previous chapters, we know that achieving disruption-free reconfiguration of IGP and BGP is highly complex. Not surprisingly, the combination of the two problems is by no means easier (we conjecture the problem is PSPACE-hard). Therefore, we advocate the need to rely on additional configuration guidelines and reconfiguration tools. To this end, we have shown how to extend our reconfiguration framework (initially described in Chapter 6) to support combined reconfigurations scenarios.

As future works, we think that understanding the different types of protocol interactions, especially during a reconfiguration process, is an interesting open problem. We also plan to extend our study to other protocols like multicast protocols.

Part V

Practical network reconfigurations

Chapter 8

Provisioning validated network configurations with NCGuard

8.1 Introduction

Since the 1970s, IP networks have evolved into highly complex distributed systems. This complexity results from at least two factors. First, IP networks can be extremely large. Indeed, some networks are now composed of more than 10,000 devices [180]. Second, IP networks can be highly heterogenous. This heterogeneity materializes itself both in terms of the number of different vendors providing devices (up to ten in large networks [183]) and in terms of the different roles played by these devices (more than 70 in some networks [180]).

To realize the operators' high-level objectives, these complex networks have to be configured. Configuring an IP network consists in adjusting the behavior of each device in a consistent manner so as to satisfy end-to-end requirements. Unfortunately, configuring every single device is complex, time consuming and error-prone. Indeed, IP networks rely on a myriad of different distributed protocols (e.g., routing and signaling) and mechanisms (e.g., access-list, QoS, SNMP, etc.). Each of these protocols and mechanisms comes with numerous and often, vendor-dependent, tunable parameters. To tune these parameters, each vendor provides a different low-level configuration language containing thousands of commands. Having to rely on these languages, network operators cannot completely automate their work and have often no other choice but to configure every device manually. Actually, the methodology used by most network operators to configure their networks is very simple [175]. Since vast part of the configuration is usually redundant, network operators often use a wiki or a set of text files to store small configuration templates for the most common management tasks [21]. When a configuration must be changed, the network operators update the configuration by cutting and pasting from the template, after having tweaked a few parameters like IP addresses. In

some cases, if the same operation has to be performed frequently, a small script is written and stored on the wiki.

Manual network management can cause a lot of potentially disruptive errors. Several surveys [166, 184] have shown that human errors are responsible for the vast majority (up to 80%) of network downtime and devices outages. Misconfigurations resulting from manual configuration are usually referred to as "fat fingers" as pressing a single wrong key on the keyboard is sufficient to create a global outage. For instance, in 2002, the misplacement of a *single bracket* in a complex route filter caused service disruption for approximately 20 percent of the US customers of UUNET [197]. Actually, numerous other misconfigurations had negative consequences on end-to-end Internet connectivity [208, 206, 207, 174, 210, 176, 197, 150]. Other misconfigurations even caused entire networks (among which two hospitals) to melt down [171, 195, 135]. In fact, network misconfigurations are so ubiquitous that, in 2002, 1% of the BGP table size was suspected to suffer from misconfigurations [87]. Several researches on the reasons behind downtimes in IP backbones consistently cite misconfiguration as the major culprit [79, 97, 89]. In addition to being disruptive, misconfiguration are also expensive. As an illustration, large (resp. medium) businesses lose an average of 3.60% (resp. 1.00%) in annual revenue due to network downtime [119, 120]. Worse yet, the cost of network downtime can escalate up to millions of dollars [170, 166] in the case of mission critical business like brokerage institutions.

In contrast to network management, software management has vastly improved in the last fifty years. Indeed, several methods to improve software quality have been proposed and *are being used* [138]. Most of these methods share common steps. First, the requirements of each application is analyzed and documented in details. Second, the software is divided in modules which are then implemented, usually in a high-level language. Third, the functional behavior of each module is carefully tested, or even validated for the key ones.

In this chapter, we describe NCGuard (the *Network Configuration safeGuard*), a novel *software-based* configuration framework which leverages several software-engineering techniques. NCGuard is structured around three building blocks: (i) a high-level model, (ii) a validation engine and, (iii) a generation engine. The *high-level model* defines the network-wide configuration while abstracting away low-level details. The *validation engine* verifies that the high-level model complies with the operators' objectives. The *generation engine* translates the model into actual configurations (possibly written in different languages) and provisions them automatically on the appropriate devices. In addition to these three building blocks, NCGuard also provides a programmatic interface to enable automation.

NCGuard provides network operators with numerous benefits including: manageability, consistency and scalability.

NCGuard improves network manageability. Thanks to NCGuard high-level model, network operators do not have to bother with low-level details

anymore. Multiple vendors are implicitly taken into account by the generation engine. NCGuard also simplifies network maintainability and orchestration where human interaction is minimized. Indeed, NCGuard enables released-based network upgrades as well as automated rollout (resp. rollback) of new (resp. old) configurations. Moreover, NCGuard takes care of the management of network-wide parameters such as IP addresses.

NCGuard enforces network consistency. NCGuard enforces configuration standardization. Standardization reduces the overall complexity of the system [180] and simplifies network troubleshooting. The NCGuard model also acts as a clear and up-to-date network documentation which most networks often simply lack. Moreover, NCGuard validation engine limits the apparition of mistakes in the model and helps ensuring that the high-level intent of the operator is always respected.

NCGuard enables network scalability. Deploying a new device in NCGuard boils down to instantiating the appropriate part of the model. Moreover, thanks to NCGuard's programmatic interface, multiple devices can be added in a few lines. Similarly, the NCGuard model is extensible and can easily accommodate new devices.

The rest of the chapter is structured as follows. Section 8.2 provides an overview of NCGuard design. Section 8.3 describes the high-level representation. Section 8.4 describes how NCGuard is able to validate the high-level representation. Section 8.5 describes NCGuard generation engine. Section 8.6 describes a NCGuard extension to support network reconfiguration. Finally, Section 8.7 reviews related work and Section 8.8 ends the chapter.

8.2 NCGuard design

NCGuard design is described in Figure 8.1. In addition to the network model, NCGuard relies on two engines: a *validation engine* and a *generation engine*. The validation engine verifies that the network model is compliant with a set of rules representing the configuration objectives of the network operator. Both the model and the rules are provided by the network operator. If the model does not comply with the rules, the validation engine produces meaningful errors (e.g., a counter-example) and warnings. If the model does comply with the rules, the generation engine generates the configuration of each device in the appropriate language by using vendor templates. A vendor template defines the translation steps required to transform the high-level representation into an actual configuration file.

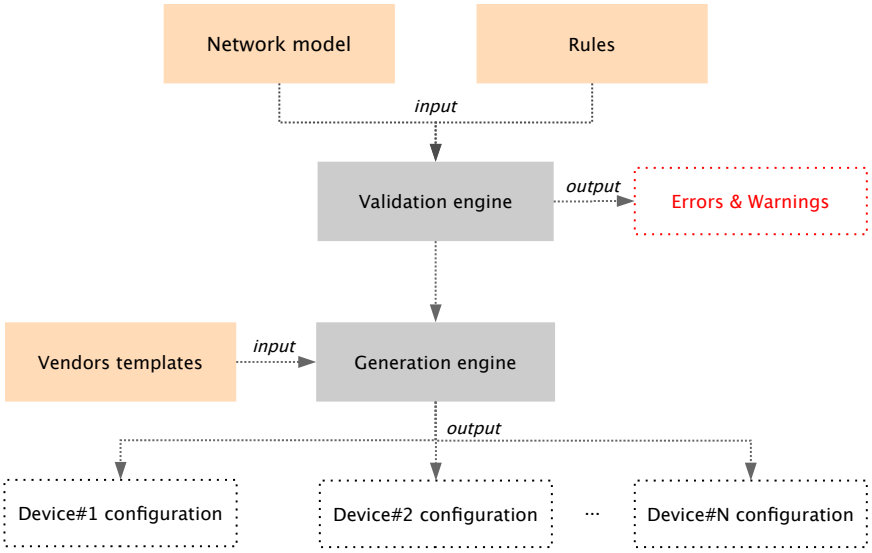


Figure 8.1 NCGuard Design. With NCGuard, network management follows a top-down approach. First, the *validation engine* validates the network model against the configuration rules provided by the operator. Once the network model is validated, the *generation engine* translates it into actual device configurations by using different vendors templates.

8.3 NCGuard high-level representation

Instead of dealing with a set of per-device configurations, NCGuard uses a single high-level XML representation of the network configuration. We chose XML as it is a highly flexible and self-describing metalanguage which can easily accommodate hierarchical structures such as network configurations. Relying on a high-level representation brings several benefits. First, it completely removes the need for redundancy by factorizing identical parts of a configuration in a single entity and by only detailing the differences in the specific parts. For instance, instead of configuring each router with a logging server, that value is represented once in the high-level representation and then automatically replicated on all the devices. Actually, all errors due to symmetry incoherence can be avoided by following this simple principle. Second, relying on a high-level representation allows NCGuard to be totally vendor-independent and to represent heterogenous networks under a common entity. Third, it allows to leverage numerous XML tools and libraries. Detailed explanations of the XML representation are available in [209].

To produce the high-level XML representation, network operators can either write it manually or generate it by using NCGuard's programmatic interface. NCGuard's programmatic interface is implemented in Python using object-oriented programming. Each concept of the network configuration is represented by a class (e.g., `Network`, `Router`, `BGPConfiguration`, `IGPConfiguration`, etc.). In addition to these classes, helper classes are also available, for instance, to automatically manage the allocation of IP addresses. Based on this object-oriented representation, a `Translator` object generates the XML representation that NCGuard uses as input. As an example, Listing 8.1 illustrates how to represent the Internet2 IGP (see Fig. 1.2) and BGP configuration using NCGuard's programmatic interface.

8.4 NCGuard validation engine

While relying on a high-level model reduces the likelihood of making some types of mistakes (e.g., duplication mistakes), some of them can still happen. Even worse, these mistakes could be mirrored in every single device as the model gets translated into low-level configurations. Therefore, the model should be validated prior to the generation phase so as to ensure that it complies with the configuration objectives of the operator. In this section, we first characterize the objectives (or intents) pursued by network operators when configuring a network, then we describe how to represent them in NCGuard by using *validation rules*.

8.4.1 Characterizing configuration objectives

To identify the configuration objectives pursued by operators, we studied the network-wide configuration of two research networks, the Belgian Re-

```
1   ################## Abilene ######################
2   abilene = Network("Abilene", 11537)
3
4   ################## Prefix Defintion ######################
5   abilene.addInternalPrefix([IPv4Network('100.0.0.0/8')])
6   abilene.setInternalLoopbackPrefix(IPv4Network('100.0.0.0/16'))
7   abilene.setSecondaryInternalLoopbackPrefix(IPv4Network('50.0.0.0/16'))
8   abilene.setInternalLinksPrefix(IPv4Network('100.1.0.0/16'))
9   abilene.setExternalLinksPrefix(IPv4Network('100.10.0.0/16'))
10
11  ################## Routers Definition ######################
12  SEAT = Router('SEAT')
13  LOSA = Router('LOSA')
14  SALT = Router('SALT')
15  KANS = Router('KANS')
16  HOUS = Router('HOUS')
17  CHIC = Router('CHIC')
18  ATLA = Router('ATLA')
19  NEWY = Router('NEWY')
20  WASH = Router('WASH')
21  abilene.addRouter(SEAT, LOSA, SALT, KANS, HOUS, CHIC, ATLA, NEWY, WASH)
22
23  ################## Intra-Links Definition ######################
24  abilene.addIntraLink(SEAT, LOSA, attributes={'ospf_weight':1342})
25  abilene.addIntraLink(SEAT, SALT, attributes={'ospf_weight':913})
26  abilene.addIntraLink(LOSA, SALT, attributes={'ospf_weight':1303})
27  abilene.addIntraLink(LOSA, HOUS, attributes={'ospf_weight':1705})
28  abilene.addIntraLink(SALT, KANS, attributes={'ospf_weight':1330})
29  abilene.addIntraLink(KANS, CHIC, attributes={'ospf_weight':690})
30  abilene.addIntraLink(KANS, HOUS, attributes={'ospf_weight':818})
31  abilene.addIntraLink(CHIC, NEWY, attributes={'ospf_weight':1000})
32  abilene.addIntraLink(CHIC, WASH, attributes={'ospf_weight':905})
33  abilene.addIntraLink(NEWY, WASH, attributes={'ospf_weight':277})
34  abilene.addIntraLink(WASH, ATLA, attributes={'ospf_weight':700})
35  abilene.addIntraLink(CHIC, ATLA, attributes={'ospf_weight':1045})
36  abilene.addIntraLink(ATLA, HOUS, attributes={'ospf_weight':1385})
37
38  ################ IGP Configuration ##################
39  abilene.igp.enableOSPF()
40  abilene.igp.setOSPFOptions(['AllInternalLinks', 'Flat'])
41
42  ############### BGP Configuration ##################
43  abilene.bgp.enableBGP()
44  abilene.bgp.createIBGPFullMesh()
```

Listing 8.1 By using NCGuard's programmatic interface, only 45 lines of code are needed to describe the Internet2 IGP and BGP configuration.

search Network and the Abilene network. We also analyzed several vendors recommendations such as [35, 36, 78, 60] as well as the configuration problems found by tools such as rcc [44], minerals [81] and others [87, 103, 45, 10].

Typically, configuration objectives can be hierarchically decomposed into high-level objectives and low-level objectives. High-level objectives correspond to general decisions concerning the network organization. Examples of high-level objectives include: ensuring the distribution of interdomain routes to all BGP routers, ensuring the distribution of intradomain routes to all routers or, ensuring sub-second convergence upon any IGP link failures. In contrast, low-level objectives are more specific and implement high-level objectives. Usually, the same high-level objective can be realized trough different combinations of low-level objectives. For instance, to ensure the distribution of interdomain routes, three different realizations exist: (i) use an iBGP full-mesh, (ii) use route reflectors and, (iii) use confederations. Each of these objectives further depends on others. For example, an iBGP full-mesh requires the IGP to advertise reachability for the endpoints of all iBGP sessions. This IGP objective is usually realized by using OSPF or ISIS. If OSPF is used it further requires the OSPF areas to be connected.

Most configuration objectives, in particular the low-level ones, can be classified into three different *objective patterns*: (*non-*)*presence*, *uniqueness* and *symmetry*. The last two patterns were already mentioned in [45].

The *presence* pattern represents objectives in which some configuration command must be present on a set of devices. For instance, [35] recommends to define an identifier on all routers to avoid letting them select a different one after a reboot. Another example is the `passive` keyword which should appear on the interfaces that connect an OSPF router to a different network to avoid creating OSPF adjacencies with a neighboring peer [35]. The *non-presence* pattern is the dual of the presence pattern. For instance, it can check that stub areas in OSPF do not contain virtual links [36].

The *uniqueness* pattern represents objectives where several network elements must have a unique value for a given configuration parameter. One example is the IP addresses assigned to different physical interfaces that must be different [35]. Another example is that, for security reasons, all eBGP sessions must use a different MD5 password.

The *symmetry* pattern represents objectives where two configurations must contain related or identical parameters. We distinguish two types of symmetry patterns. The first type is the *simple equality* pattern in which some configuration parameters in two different network elements must be equal. For example, two routers attached to the same physical link must use the same layer-2 encapsulation. As another example, the same OSPF timers must be configured on two adjacent routers. A third example could be that the same OSPF weights must be used on both sides of a link (to enforce symmetrical routing). The second type of symmetry pattern is the *cross equality* pattern in which the equivalence has to be ensured across

multiple parameters. For instance, in order to establish an iBGP session between routers *A* and *B*, router *A* has to configure *B* as its neighbor, and *B* has to configure *A* as its neighbor [36, 35].

Finally, the *custom* pattern represents more complex configuration objectives that do not belong to any of the previous patterns. Usually, these configuration objectives relate to the entire network. One example of such objective could be that the network should remain connected after any single link failure or after the simultaneous failure of any pair of routers. Another example could be that at least two disjoint equal-cost-paths must exist between any routers.

8.4.2 Modeling and verifying configuration objectives

In NCGuard, a configuration objective is expressed as a *rule*. A rule verifies that a given property is satisfied by the high-level representation. As with the high-level representation, we chose to represent the rules with XML elements. Since rules are hierarchically organized, the XML elements representing the rules are organized as a tree, where the root corresponds to a very high-level rule such as "the network is correct" (see Fig. 8.2). Each rule can define one or more dependencies (i.e., branches in the tree). Depending on the rule, all the dependencies must be verified (*and* dependencies) in order for the high-level rule to be verified or only one of them (*or* dependencies).

```
<rules rootid="CORRECT_NETWORK">
  <rule id="CORRECT_NETWORK">
    <description>This is the main rule.</description>
    <dependencies type="and">
      <depends>CORRECT_CONFIGURATION</depends>
      <depends>CORRECT_IP_CONNECTIVITY</depends>
      <depends>CORRECT_VLAN</depends>
      <depends>CORRECT_OSPF</depends>
      <depends>CORRECT_BGP</depends>
    </dependencies>
  </rule>
  ...
</rules>
```

Listing 8.2 NCGuard hierarchically organizes validation rules

NCGuard uses three techniques to verify rules. First, rules can be verified by constraining the syntactical structure of the high-level XML representation. For instance, forcing each XML node representing a router to have exactly one child node containing its identifier. Typically, such constraints are expressed by associating the high-level representation with a grammar which precisely defines the syntax of the high-level XML representation. In NCGuard, this grammar is implemented by using a XML Schema [43]. Rules implemented with this technique are called *structural rules*. Second, rules can be checked by executing queries on the network representation. These rules are called *query rules*. In NCGuard, the queries are implemented with XQuery, an XML query language [148, 127]. Third,

	Structural	Query	Language	*Total*
Uniqueness	14	6	-	20
Symmetry	10	-	-	10
Presence	82	15	-	97
Custom	-	6	3	9
Total	*106*	*27*	*3*	*136*

Table 8.1 More than 100 rules have been defined to check Abilene's configuration objectives. Most of them are implemented with structural rules, as syntactic constraints expressed on the high-level representation.

rules can be checked by using a programming language (Java in our prototype). These rules are called *language rules*. Depending on the rule, one verification technique is typically more appropriate. For instance, presence and uniqueness rules are easily implemented using a structural or a query rule. Conversely, custom rules usually require a query rule or a language rule as they require more expressiveness. Finally, symmetry rules are implicitly verified since redundant informations are factorized in the high-level representation. For instance, the symmetry of the MTU definition is ensured by defining it directly on the link and not on the interfaces connected to the link. As an illustration, we developed a high-level representation of the Abilene network by reverse-engineering their router configurations. Based on these configurations, we also inferred 136 configuration objectives. We represented most of these configuration objectives by using structural and query rules. Table 8.1 details the distribution of the techniques used to validate the configuration rules.

In the following, we provide four examples of NCGuard rules implemented with different techniques.

Rule#1: All routers have a unique router id This combined rule (presence and uniqueness) is expressed by two structural constraints inside the XML schema (see Listing 8.3).

Rule#2: A loopback interface is present on all nodes This presence rule is expressed by a query rule. As described earlier, NCGuard implements query rules by using XQuery expressions. To simplify the addition of query rules, we abstracted them in a XML element composed of a *scope*, a *descendant*, and a *condition* sub-elements (see Listing 8.4). The scope element identifies the set of XML nodes in the high-level representation on which the rule applies. The descendant element identifies (if needed) specific children nodes of the scope node. Finally, the condition element states a property that must be respected by at least one descendant node for each scope node. Consequently, the rule depicted in Listing 8.4 gets translated by NCGuard into a XQuery which checks that for all routers

```
<xs:complexType name="router">
    <xs:sequence>
        ...
        <xs:element name="rid" type="InetAddressIPv4" minOccurs="1" maxOccurs
            ="1"/>
        ...
    </xs:sequence>
</xs:complexType>

<xs:key name="nodeIdKey">
    <xs:selector xpath="topology/routers/router"/>
    <xs:field xpath="@id"/>
</xs:key>
```

Listing 8.3 Presence and uniqueness rules are implemented with syntactical constraints expressed on the high-level representation by using a XML schema. In this excerpt, the first constraint mandates a router element to have a rid, while the second constraint mandates the rid to be unique.

(scope), there is at least one interface (descendant) for which the id is a loopback id (condition).

```
<rule id="LOOPBACK_INTERFACE_ON_EACH_NODE" type="presence">
  <presence>
    <scope>ALL_ROUTERS</scope>
    <descendant>interfaces/interface</descendant>
    <condition>substring(@id,1,2)='lo'</condition>
  </presence>
</rule>
```

Listing 8.4 Presence rule: Loopback interface on each node

Notice that the distinction between scope and descendant, although not technically needed, allows to reuse scope definitions in different rules, which happens frequently. Examples of frequently used scopes include: the set of all routers, the set of all border routers, the set of all loopback interfaces, etc.

Rule#3: Loopback addresses are advertised in OSPF This presence rule (see Listing 8.5) is also verified by a query rule. Notice that this rule is perfectly equivalent to the previous one except for the scope definition.

```
<rule id="LOOPBACK_ADVERTISED_OSPF" type="presence">
  <presence>
    <scope>OSPF_NODE</scope>
    <descendant>interfaces/interface</descendant>
    <condition>substring(@id,1,2)='lo'</condition>
  </presence>
</rule>
```

Listing 8.5 Loopback addresses must be announced in OSPF

Rule#4: OSPF areas must be directly connected to the backbone area This rule is an example of a custom rule as it belongs to none of the patterns identified above. However, it can be implemented with a tailored XQuery query. To report useful errors and warnings, NCGuard queries should always identify the elements that do not respect the rule. In Listing 8.6, the query identifies the OSPF areas without an area border router connected to the backbone area.

```
<rule id="AREAS_CONNECTED_TO_BACKBONE_AREA" type="custom">
  <custom>
    <xquery><![CDATA[
for $area in /domain/ospf/areas/area[@id!="0.0.0.0"]
let $backbone_nodes :=
/domain/ospf/areas/area[@id="0.0.0.0"]/nodes/node
where not(exists($backbone_nodes[@id=$area/nodes/node/@id]))
return <result><area id="{$area/@id}"/></result>]]>
    </xquery>...
```

Listing 8.6 OSPF areas must be directly connected to the backbone area

To actually validate the representation, the validation engine proceeds in two steps. First, it checks if the XML representation is syntactically correct. If it is the case, the validation engine recursively verifies the rules located in the tree, starting at the root. Whenever a rule is not verified, an error containing a counter-example is printed (see Listing 8.7 for an example). Indeed, as noted by [39], meaningful error messages are important to help network operators troubleshoot the model. Regarding its implementation, the validation engine was written in about 1500 lines of Java, leveraging the saxon XML libraries.

```
Node NY does not have the 'next-hop-self' option enabled.
Therefore, routes announced on eBGP session 'NY-GEANT'
might be unreachable inside the AS.
Activate 'next-hop-self' or put interface 'ge-4/0/0.102' inside OSPF (as passive).
```

Listing 8.7 Excerpt of NCGuard output. To help debugging, meaningful errors are important for network operators. To that extent, NCGuard prints counter-examples whenever it is possible.

8.5 NCGuard generation engine

Just as a high-level language must be compiled to machine code, a high-level model must be compiled or translated into low-level configurations, possibly written in different languages. The generation engine performs this translation in two steps.

The first generation step consists in transforming the high-level representation into an intermediate representation (also XML-based) whose goal is to facilitate the overall translation process. The translation process is

simplified because the intermediate representation is closer to the configurations of real routers. For example, while the MTU is defined directly on the link in the high-level representation, it is specified on each interface in the intermediate representation. The intermediate representation also contains characteristics of each target router (vendor, OS, memory, types of interfaces, etc.). These characteristics are crucial as some configuration parameters are only valid on some specific platforms.

The second generation step consists in translating the intermediate representations into actual vendor configurations by applying XSLT templates [29]. One XSLT template is used for each configuration language to be supported. Currently, NCGuard generates both Cisco IOS and Juniper JunOS configurations. In addition to configuration languages, tailored XSLT templates could be provided to generate appropriate input for simulators such as C-BGP [110] or traffic engineering tools [10, 45]. A snapshot of the high-level representation is depicted in Listing 8.8 along with the translated Juniper configuration in Listing 8.9.

```
<interface id="so-0/0/0">
  <unit number="0">
    <ip mask="31" type="ipv4">64.57.28.11</ip>
  </unit>...
```

Listing 8.8 Snapshot of the high-level representation

```
interfaces {
    so-0/0/0 {
        unit 0 {
            family inet {
                address 64.57.28.11/31;
...
```

Listing 8.9 Juniper configuration corresponding to Listing 8.8

Regarding its implementation, NCGuard generation engine was written in about 800 lines of Java. The XSLT style sheets used to generate the BGP and OSPF JunOS configurations contain about 1200 lines.

8.6 Automating network reconfiguration

In this section, we show how to extend NCGuard's programmatic interface (see Section 8.3) to support and pilot an entire network reconfiguration process.

In NCGuard programmatic interface, a reconfiguration process is encoded as a directed acyclic graph $G = (V, E)$, or reconfiguration graph. The reconfiguration graph is either computed automatically, for instance, by using the ordering techniques described in the previous chapters or is

manually fixed by the network operator. Each node C_i in V represents a high-level network representation, where C_{init} and C_{fin} represent the initial and the final representation, respectively. Each edge $e = (C_i, C_{i+1})$ in E represents two consecutive high-level representations of the reconfiguration process. Observe that consecutive representations can differ completely or differ by only one line on one router. Each edge in the graph is further associated with a list configuration statements and a predicate. When applied to the network, the list of configuration statements transform C_i into C_{i+1}. A predicate is a condition that must be satisfied before executing the reconfiguration operation. Currently, we have identified four predicates which can be combined to form more complex conditions:

#1 WaitForProtocolConvergence This predicate assesses that a protocol instance has converged. Assessing the convergence of a protocol can be done by monitoring the state of the routers either by using "show commands" or by directly collecting the routing updates. Among others, assessing the convergence is important to avoid the problems related to transient states. This predicate can also be used to monitor the stability of a protocol instance. Notice that the implementation of the predicate must be particularized depending on the protocol.

#2 WaitForProtocolCompleteness This predicate assesses that a protocol instance contains required routing information. The required routing information should be provided beforehand by the network operator. For instance, this predicate can verify that a IGP has propagated a route for all the internal prefixes. This predicate is important to avoid loss of visibility during the reconfiguration. With respect to the *WaitForProtocolConvergence* predicate, this predicate checks the presence of routing information, not its stability.

#3 WaitForTimerExpiration This simple predicate assesses that a given amount of time has elapsed before executing the associated transition. It is useful as in most reconfiguration scenarios each intermediate step takes a bounded time to finish (e.g. a few minutes).

#4 WaitForManualInput This predicates waits for a manual input from the network operator. It is useful when the operator wants to keep full control of some critical reconfiguration steps (e.g., the launching time).

An actual migration corresponds to a path $C_{init} \dots C_k \dots C_{fin}$ in the migration graph. If multiple paths exist between C_{init} and C_{fin}, one path is chosen before the migration. Several criteria can be used to discriminate among several paths. For instance, an operator might prefer the path with the minimal amount of intermediate states or a path in which the reconfiguration steps are kept as local as possible.

As illustration, we consider a reconfiguration scenario performed in the Internet2 network (see Fig.1.2) where an iBGP full-mesh is replaced with a three-level route-reflection hierarchy. As reconfiguration strategy, we use

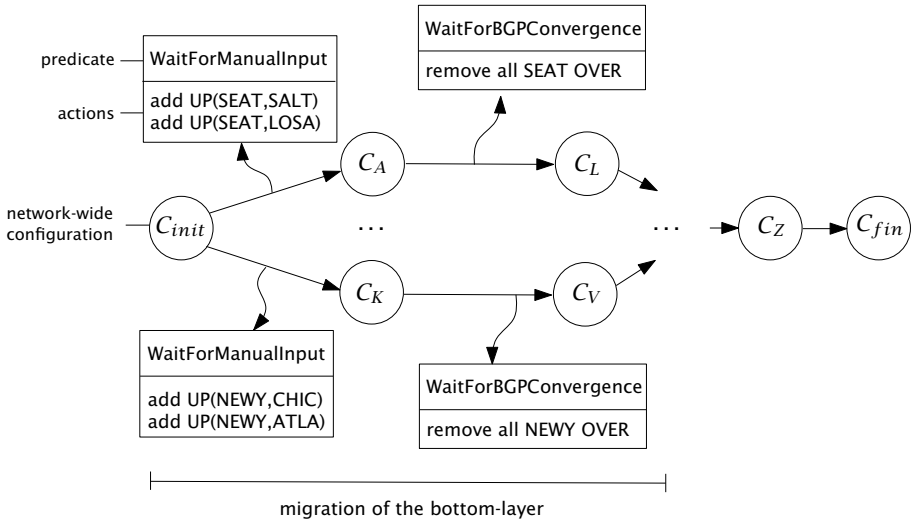


Figure 8.2 Snapshot of a migration graph.

the best current practices [156, 198, 68] which consist in migrating routers in a bottom-up manner. Each router is migrated in two steps. First, the final UP sessions are established along with the initial OVER sessions. Once the duplicated routes have been propagated on the UP sessions, the initial OVER sessions are removed. Regarding the final route-reflection topology, it also follows the best current practices by closely following the physical topology. In particular, routers KANS and HOUS form the top-layer, routers SALT, LOSA, CHIC and ATLA, the intermediate one, while routers SEAT, NEWY and WASH form the lowest one. For redundancy purposes, each router is connected to the two route-reflectors located immediately above it.

A snapshot of the corresponding reconfiguration graph is depicted in Fig. 8.2. Also, Listing 8.10 shows how NCGuard can represent the entire migration scenario in less than 30 lines (lines 69–98). In this example, NCGuard generates first all the intermediate configurations and associates each of them with an intermediate node in the reconfiguration graph (lines 72–77). The nodes are then linked together and each transition is associated with a predicate (lines 79–92). A timer based predicate of five seconds is used between each transition. In addition to pilot the reconfiguration, NCGuard is also able to gather statistics during the migration, e.g., by logging the output of “show commands”. In this particular example, a “show ip bgp all” is performed on all routers after each migration step. The actual migration process is finally launched (line 95). During this process, NCGuard uses the Command Line Interface (CLI) to interact with the network equipments. Since configuration changes often concern only specific parts of the configuration, only the changes are pushed on the devices. On average, the complete Internet2 migration process took less than 10 minutes to complete (15 repetitions). Such a time efficiency enables operators

to migrate an entire network within a single maintenance window (which typically lasts for a few hours only [199]).

Finally, a network operator will probably want to test its migration process beforehand, even when NCGuard is used. One way to do that is to perform the migration in a virtual environment. Thanks to NCGuard's programmatic interface, generating and provisioning virtual environments is really easy. At the time of writing, NCGuard supports the generation of dynamips [158] labs, which allow to virtualize Cisco routers. To further help network operators, NCGuard provides the ability to inject tailored (IGP or BGP) routes or routes feed collected from the existing network or from Routeviews [190]. The ability to generate a virtual lab out of a high-level representation is also useful for researchers. Indeed, it enabled us to validate the intended behavior of *all the gadgets* described in this thesis.

```
1  ################# Migration ######################
2  # We create the Lab
3  ucl_lab = Lab("UCLab")
4
5  # Add servers to the lab (multiple server support)
6  ucl_lab.add_server(Server(name="nostromo"))
7
8  # Generate the initial network-wide configurations
9  translate([abilene], lab=ucl_lab)
10
11 ################ Migration graph definition ########################
12 network_states = []
13
14 # Generate the network-wide configuration of each intermediate state
15 for rid in compute_migration_order(final_ibgp, randomize=True):
16     print "Generating configuration for router %s" % (rid)
17     migrate_router(abilene.routers[rid], final_ibgp, lab=ucl_lab)
18     translate([abilene], lab=ucl_lab)
19     network_states.append(MigrationState(name="ITER#"+str(i))
20
21 # Link the state together and associate the transitions with a predicate
22 migration_process = MigrationProcess(name="ABILENE_FM2RR", network=abilene)
23
24 for (head_state, tail_state) in zip(network_states, network_states[1:]):
25     migration_process.add_transition(
26         MigrationTransition(head_state, tail_state, TimerPredicate(5),
27         options = {'show_commands' :
28                 {
29                     'internal_routers' : {
30                         'commands': ['show ip bgp all'],
31                         'routers' : abilene.routers.values()
32                     }
33                 }
34                })
35
36 # Start the migration process !
37 migration_process.start()
```

Listing 8.10 Thanks to NCGuard's programmatic interface, it is possible to describe the complete full-mesh to route-reflection scenario of the Internet2 network. On average, less than 10 minutes are required to complete the migration scenario.

8.7 Related Work

Numerous works have been proposed to improve network configuration management. This section briefly surveys the most relevant of them with respect to NCGuard.

The closest work with respect to NCGuard is probably Metaconfiguration [90]. Metaconfiguration also uses a XML-based representation of the network and validates it trough a set of rules. However, Metaconfiguration validation is not as detailed and as extensible as NCGuard's validation techniques. For instance, it does not allow rules to be hierarchically organized.

PRESTO [39] is a configuration management system which is built upon a database and a series of composable active templates. An active template is basically a text-file, written in the low-level configuration language, which supports for loops, conditional logic and access to the database. Other works follow the same template-based philosophy whether they are research initiatives (e.g. [61]), open-sources tools (e.g. [175]) or proprietary tools (e.g. [189, 181, 180]). Compared to NCGuard, these tools still require the network operators to know every single configuration language. Moreover, they do not support the validation of the model. Another network management system which abstracts the network as a relational database is DECOR [26]. However, DECOR goes one step further by incorporating in the database other network management aspects, such as dynamic properties and network operations (e.g., reconfiguration). With respect to DECOR, NCGuard better supports network reconfiguration as it can encompass the reconfiguration techniques developed in the thesis (see Section 8.6). PACMAN [27] is another management system which enables automation and abstraction while not requiring operators and network designers to adopt a completely new network management paradigm. While PACMAN improves network management by automating the low-level actions, it does not increase the level of abstraction at which it is performed, as such it cannot reap the same level of benefits as the ones provided by NCGuard and its top-down approach.

Recently, Autonetkit [96] simplified the configuration of complex, large-scale emulated networks by relying on a high-level view of the network written in Python. Based on a high-level view, Autonetkit automatically provisions a Netkit lab [106] with the appropriate Quagga router configurations. While sharing a lot of common features with NCGuard, including the ability to generate virtual labs automatically, Autonetkit has fewer verification capabilities and does not support live reconfigurations.

Several works have aimed specifically at improving the configuration of BGP and its associated routing policies. BGP configuration is indeed known to be one of the most complex and changing part of the global configuration [85]. The Routing Policy Specification Language [1] was designed to allow network operators to document their routing policies in the `whois` databases. Some network operators, notably in Europe, rely on these RPSL databases to automatically configure their route filters by using the IRR

toolset. Tools proposed by Maennel *et al.* [182] and Gottlieb *et al.* [61] also automate the configuration of BGP sessions. Nettle [146] abstracted away the problem of configuring BGP. Nettle is a domain-specific embedded language based on Haskell for configuring BGP networks. Nettle separates the specification of the configuration from its actual representation in the router. Also, Nettle allows to prevent anomalies such as oscillations from happening.

Instead of producing correct configurations, other approaches have tried to detect configuration errors. Two main approaches exist. The first approach is to apply data mining techniques (e.g.,[21, 38, 81]) to the configuration files in order to detect statistical deviations that are likely to be errors. While these techniques require no input from the operators, they generate a lot of false positives as some configurations tend to be highly specialized. Operators are often reluctant to analyze all the warnings reported by these tools. Moreover, machine learning techniques do not work on heterogenous networks in which different languages are used. The second approach to detect misconfigurations is to use static analysis [44] which checks that configurations comply with some predefined rules. NCGuard also relies on static analysis, but it is used to validate the model, not the actual configurations.

Most router vendors now provides XML-based APIs and the Internet Engineering Task Force (IETF) is developing NETCONF [40] to provide a standardized interface to router configuration. Along with NETCONF, the IETF standardized YANG [15], a XML-based data modeling language used to model configuration and state data manipulated by NETCONF. Being XML-based, YANG configurations could also benefit from NCGuard validation techniques. Moreover, NCGuard can be easily extended to produce YANG configurations and use NETCONF as provisioning mechanism.

8.8 Conclusions

Nowadays, most IP networks are still configured manually, which is both error-prone and costly. In this chapter, we argued for a radically different approach inspired by software engineering techniques. To support our approach, we developed NCGuard, the Network Configuration SafeGuard. The philosophy behind NCGuard can be summarized in three key points. First, NCGuard encourages the network operator to formally specify the objectives of his network. These objectives are defined as a set of rules that must be met by the configuration. Second, NCGuard abstracts away low-level configuration language by proposing a high-level XML-based representation to the operator. Third, NCGuard validates the high-level representation and generates the router configurations in their respective configuration languages. In addition to these three points, NCGuard also provides a programmatic interface which enables it to support automated provisioning but also disruption-free reconfiguration by combining it with the reconfiguration techniques we developed in this thesis.

While being an important first step, our work is far from being finished. First, our model must be extended to support other routing protocols and mechanisms. Also, it could be made more modular by decomposing the model into different abstraction layers. Second, the coverage of NCGuard validation should be improved. We see two ways of achieving this objective: (i) release a library of validation rules so that it could be extended by other people (provided that they use a compatible high-level representation) and (ii) combine NCGuard rule-bases validation with other verification techniques, for instance, model checking. Third, regarding the generation of the configuration, NCGuard could leverage the ability of new routers to support for standardized languages like YANG [15].

Regarding the reconfiguration process, an important open problem is how to deal with failures or unplanned events that happen during the reconfiguration. Indeed, in case of such event, it might be better to abort the reconfiguration process and undo some of the changes so as to recover an operational state. The problem of backtracking the reconfiguration process is known to be complex [153] and is still largely unsolved. In our opinion, different strategies exist depending on the sensitivity of the network operator to failures. For instance, the operator could pinpoint special nodes in the reconfiguration graph that can act as milestones. If an error is detected, NCGuard can automatically backtrack the network to the previously encountered milestone. Another strategy, more automated, could be to precompute backtrack edges linking a node to a previous one known to be correct according to different types of failures. We believe that developing techniques and tools to automatically compute milestones and backtrack edges is a promising area of research.

Chapter 9

Conclusions and open problems

Network-wide reconfigurations, although beneficial, are often avoided or delayed as they can introduce numerous anomalies. In view of our experimental findings, this is not surprising as we repeatedly showed that *poorly executed reconfigurations can create long and service-affecting outages*. Indeed, in contrast to network failures where the disruption is usually limited to a short time period [89] (e.g., a few minutes), reconfiguration anomalies can last for a substantial time period (e.g., a few hours in the case of a large network reconfiguration).

In this thesis, we developed techniques and tools to enable anomaly-free routing reconfiguration for both intradomain and interdomain routing protocols. To avoid routing and forwarding anomalies, our approach was to order the configuration changes so that every intermediate step maintains network correctness. We started this thesis by asking ourselves two questions: *Does an anomaly-free reconfiguration ordering always exist?* and *Is it easy to compute?* Unfortunately, the answer to both of these questions turned out to be negative. Indeed, we systematically discovered reconfiguration cases in which no ordering exists. Moreover, we proved that deciding if an anomaly-free reconfiguration ordering exists is computationally hard ($\mathcal{NP}$-hard). Despite the inherent complexity of the reconfiguration problem, we managed to enable disruption-free reconfigurations (or at least reduce the anomalies) in most reconfiguration scenarios. We now review our contributions.

In Chapter 2, we provided a general characterization of the forwarding anomalies that could happen in an IGP reconfiguration scenario. Thanks to that characterization, a network operator knows precisely what types of forwarding anomalies he/she can face when performing a given IGP reconfiguration. Our main result was to show that forwarding loops can *only* happen in pure link-state IGP reconfiguration scenarios, while traffic shifts can happen in any IGP reconfiguration scenario. In Chapter 3, we studied how to avoid forwarding loops in link-state IGP reconfigurations. To compute a per-router ordering (when it exists), we developed a correct and complete algorithm and a correct but not complete heuristic. Even though the algorithm is theoretically inefficient, it was able to find an anomaly-

free ordering for all networks, but one. Regarding the heuristic, its time efficiency allowed us to make the ordering resilient against single-link failures. In Chapter 4, we developed an effective heuristic which reduces the amount of traffic shifts created during distance-vector to link-state reconfigurations.

After our study of IGP reconfigurations, our next target was to enable anomaly-free BGP reconfigurations. However, when we started our work, we realized that the known correctness properties related to iBGP configuration (i.e., signaling and forwarding correctness) were not complete. Indeed, the impact of iBGP route propagation was overlooked. We explored that problem in Chapter 5. Counter-intuitively, we showed that *adding* iBGP sessions can *reduce* the overall amount of BGP routes globally known in the network. The reduction can be such that some routers can end up learning no route to reach some destinations. Unfortunately, checking for dissemination correctness is computationally-hard, however we provided two sufficient conditions to enforce it as well as design guidelines.

In Chapter 6, we studied BGP reconfigurations. With respect to IGP reconfigurations, avoiding BGP routing and forwarding anomalies is harder for at least three reasons. First, local changes can unpredictably change routing decisions at remote routers. These remote changes must therefore be considered when computing the ordering. Second, the number of routes carried by the IGP and BGP usually differs by at least two orders of magnitude. Third, network operators do not control where BGP destinations are originated (i.e., a BGP destination can be originated from any subset of routers) while IGP destinations are usually originated from well-known routers (usually one). This additional complexity spurred us to develop additional tools. In particular, we developed a reconfiguration framework in which the initial and the final network-wide BGP configurations coexist. With that framework, reconfiguring a router consists in switching the BGP configuration which is responsible for packet forwarding. We proved that this switching was safe. Our framework leverages existing technologies and can work on today's routers.

In Chapter 7, we considered the impact of IGP reconfiguration on BGP. We showed that IGP configuration can create any routing and forwarding anomaly for interdomain destinations even if the reconfiguration is anomaly-free at the IGP level. The interactions between the protocols must therefore be taken into account during the reconfiguration. Unfortunately, these interactions complicate even more the reconfiguration problem. To solve combined IGP and BGP reconfigurations, we extended the reconfiguration framework originally designed for BGP reconfigurations by allowing multiple IGP and BGP processes to coexist.

Finally, in Chapter 8, we presented a top-down approach to network management along with a working implementation named NCGuard (Network Configuration safeGuard). Our approach encourages the network architect to first specify formally the objectives of his/her network. These objectives are defined as a set of rules that must be met by the configuration. Then, the network architect writes a high-level representation of

his/her network. NCGuard validates the high-level representation against the rules defined by the architect and generates the routers configurations in their respective languages. In addition to drastically reduce the likelihood of misconfigurations, software-based network management allowed us to completely automate the reconfiguration process including the live provisioning of device configurations.

Open problems

Research on network reconfiguration is still in its infancy and, as pointed out in each chapter, there is plenty of room for further research activities. Regarding IGP reconfigurations, the algorithms of Chapter 3 can be extended to limit the creation of traffic shifts during the reconfiguration. While we argued that the reconfiguration process can be performed when the network is lightly loaded, such periods might not exist (e.g., in the case of a worldwide content-provider). Also, the ability of migrating several routers almost simultaneously can be leveraged to speed up the reconfiguration process or to use it as a fallback approach when no per-router ordering exists. Regarding DV to LS reconfigurations (see Chapter 4), a further work would be to integrate the support of graph separator in the heuristic. Leveraging graph separator would allow to speed up the reconfiguration process without sacrificing correctness.

Regarding BGP reconfigurations, we only assumed the initial and the final configurations to be free from anomalies. However, in many networks, common design guidelines are followed. These guidelines could therefore be taken into account to develop smart “in-place” reconfiguration techniques that do not require to rely on the framework. Regarding the reconfiguration framework, we have chosen to deliberately rely on existing technologies so as to support incremental deployment. However, relaxing that constraint would allow the framework to scale better, e.g., by sharing common entries in the RIB and in the FIB. We also believe that studying the interactions between routing protocols is an interesting open problem raised by the thesis.

Finally, another interesting area of research would be to develop a distributed reconfiguration protocol. With such a protocol, routers would be aware that a reconfiguration will take place and could react accordingly (e.g., by cleverly ordering the updates in the control- and in the forwarding-planes).

Concluding remarks

Overall, this thesis has shown that large and network-wide reconfigurations can be tackled without having to reboot the network or to suffer from losses. By using safe reconfiguration techniques, network operators are now in a position to move at anytime to whatever is the best configuration for them according to their needs.

While we believe our work is an important leap towards more flexible network management, it is just a first step. In particular, multiple networks aspects beyond routing rely on configuration and could thus benefit from tailored reconfiguration techniques, e.g., data-link layer (switches), security policies (firewalls) or load-balancers.

Finally, we envision network reconfiguration to be a building block of future network management paradigms, fostering the development and the deployment of new technologies, protocols and devices without disrupting existing services.

Bibliography

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

[2] B. Albrightson, J.J. Garcia-Luna-Aceves, and J. Boyle. EIGRP - A Fast Routing Protocol Based On Distance Vectors. In *Proc. Networld/Interop 94*, 1994.

[3] M. A. Alim and T. G. Griffin. On the interaction of multiple routing algorithms. In *Proc. CoNEXT*, 2011.

[4] R. Alimi, Y. Wang, and Y. R. Yang. Shadow configuration as a network management primitive. In *Proc. SIGCOMM*, 2008.

[5] L. Andersson, I. Minei, and B. Thomas. LDP Specification. RFC 5036 (Draft Standard), October 2007.

[6] A. Atlas and A. Zinin. Basic Specification for IP Fast Reroute: Loop-Free Alternates. RFC 5286 (Proposed Standard), September 2008.

[7] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209 (Proposed Standard), December 2001. Updated by RFCs 3936, 4420, 4874.

[8] H. Ballani, P. Francis, T. Cao, and J. Wang. Making routers last longer with ViAggre. In *Proc. NSDI*, 2009.

[9] S. Balon and G. Leduc. Combined Intra- and Inter-domain Traffic Engineering using Hot-Potato Aware Link Weights Optimization. In *Proc. SIGMETRICS*, 2008.

[10] S. Balon, J. Lepropre, O. Delcourt, F. Skivée, and G. Leduc. Traffic Engineering an Operational Network with the TOTEM Toolbox. *IEEE Transactions on Network and Service Management*, 4(1):51–61, June 2007.

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

[12] R. Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[13] T. Benson, A. Akella, and D. Maltz. Phoenix: A System for Automatically Reconfiguring Networks. In *Poster Session, USENIX Symposium on Networked Systems Design and Implementation*, NSDI, 2009.

[14] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. In *USENIX NSDI*, Boston, MA, 2009.

[15] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020 (Proposed Standard), October 2010.

[16] R. Bolla, R. Bruschi, A. Cianfrani, and M. Listanti. Enabling backbone networks to sleep. *IEEE Network: The Magazine of Global Internetworking*, 25(2):26–31, March 2011.

[17] M. Buob, M. Meulle, and S. Uhlig. Checking for optimal egress points in iBGP routing. In *Proc. DRCN*, 2007.

[18] M. Buob, S. Uhlig, and M. Meulle. Designing optimal iBGP route-reflection topologies. In *Proc. Networking*, 2008.

[19] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proc. NSDI*, Berkeley, CA, USA, 2005.

[20] M. Caesar and J. Rexford. BGP routing policies in ISP networks. *Network, IEEE*, 19(6):5–11, 2005.

[21] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford. The cutting EDGE of IP router configuration. *SIGCOMM Comput. Commun. Rev.*, 34(1):21–26, 2004.

[22] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990.

[23] S. Chan-Olmsted and M. Jamison. Rivalry through alliances:: Competitive Strategy in the Global Telecommunications Market. *European Management Journal*, 19(3):317 – 331, 2001.

[24] E. Chen. Route Refresh Capability for BGP-4. RFC 2918 (Proposed Standard), September 2000.

[25] R. Chen, A. Shaikh, J. Wang, and P. Francis. Address-based Route Reflection. In *Proc. CoNEXT*, 2011.

[26] X. Chen, Y. Mao, Z. M. Mao, and J. Van der Merwe. Decor: Declarative network management and operation. *SIGCOMM Comput. Commun. Rev.*, 40(1):61–66, January 2010.

[27] X. Chen, Z. M. Mao, and J. Van der Merwe. PACMAN: a platform for automated and controlled network operations and configuration management. In *Proc. CONEXT*, 2009.

[28] L. Cittadini, G. Di Battista, and S. Vissicchio. Doing don'ts: Modifying BGP attributes within an autonomous system. In *Proc. NOMS*, 2010.

[29] J. Clark. XSL transformations (XSLT) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[30] T. Cormen, C. Stein, R. Rivest, and C. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[31] S. Das, O. Egecioglu, and A. El Abbadi. Anonimos: An LP Based Approach for Anonymizing Weighted Social Network Graphs. *Trans. on Know. and Data Eng.*, pages 1–14, 2010.

[32] B. Decraene, JL. Le Roux, and I. Minei. LDP Extension for Inter-Area Label Switched Paths (LSPs). RFC 5283, 2008.

[33] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998.

[34] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[35] K. Dooley and I. Brown. *Cisco IOS Cookbook*. O'Reilly Media, Inc., 2006.

[36] J. Doyle. *CCIE Professional Development, Routing TCP/IP, Volume 1*. Cisco Press, 1998.

[37] J. Doyle and J. DeHaven Carroll. *CCIE Professional Development, Routing TCP/IP, Volume 2*. Cisco Press, 2011.

[38] K. El-Arini and K. Killourhy. Bayesian detection of router configuration anomalies. In *MineNet '05: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 221–222, New York, NY, USA, 2005. ACM.

[39] W. Enck, P. McDaniel, S. Sen, P. Sebos, S. Spoerel, A. Greenberg, S. Rao, and W. Aiello. Configuration management at massive scale: System design and experience. In *USENIX Annual Technical Conference*, pages 73–86. USENIX, 2007.

[40] R. Enns. NETCONF Configuration Protocol. RFC 4741, December 2006.

[41] M. Fabbi and D. Curtis. Introducing a Second Network Vendor Saves Money and Solidifies Operations. Gartner Technical White Paper, May 2009.

[42] M. Fabbi and D. Curtis. Debunking the Myth of the Single-Vendor Network. Gartner Technical White Paper, 2010.

[43] D. Fallside and P. Walmsley. XML Schema Part 0: Primer Second Edition. W3C recommendation, W3C, October 2004.

[44] N. Feamster and H. Balakrishnan. Detecting BGP Configuration Faults with Static Analysis. In *Proc. NSDI*, 2005.

[45] A. Feldmann and J. Rexford. IP Network Configuration for Intradomain Traffic Engineering, Sept. 2001.

[46] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601 (Proposed Standard), August 2006.

[47] C. Filsfils, P. Francois, M. Shand, B. Decraene, J. Uttaro, N. Leymann, and M. Horneffer. LFA applicability in SP networks. Internet Draft, May 2011.

[48] C. Filsfils, P. Mohapatra, J. Bettink, P. Dharwadkar, P. De Vriendt, Y. Tsier, V. Van Den Schrieck, O. Bonaventure, and P. Francois. BGP Prefix Independent Convergence (PIC). Technical report, Cisco, 2011.

[49] A. Flavel and M. Roughan. Stable and flexible iBGP. In *Proc. SIGCOMM*, 2009.

[50] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *IEEE Comm. Mag.*, pages 118–124, 2002.

[51] B. Fortz and M. Thorup. Optimizing ospf/is-is weights in a changing world. *Selected Areas in Communications, IEEE Journal on*, 20(4):756 –767, May 2002.

[52] P. Francois. *Improving the Convergence of IP Routing Protocols*. PhD thesis, Université catholique de Louvain, October 2007.

[53] P. Francois and O. Bonaventure. Avoiding transient loops during the convergence of link-state routing protocols. *Trans. on Netw.*, 15(6):1280–1932, 2007.

[54] P. Francois, B. Decraene, C. Pelsser, K. Patel, and C. Filsfils. Graceful BGP session shutdown. Internet-Draft, December 2011.

[55] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving subsecond IGP convergence in large IP networks. *Comput. Commun. Rev.*, 35(3):33–44, 2005.

[56] P. Francois, M. Shand, and O. Bonaventure. Disruption-free topology reconfiguration in OSPF Networks. In *Proc. INFOCOM*, 2007.

[57] J. Fu, P. Sjodin, and G. Karlsson. Loop-Free Updates of Forwarding Tables. *Trans. on Netw. and Serv. Man.*, 5(1):22–35, 2008.

[58] L. Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE Global Internet*, November 2000.

[59] J.J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *Networking, IEEE/ACM Transactions on*, 1(1):130 –141, feb 1993.

[60] A. Garrett. *JUNOS Cookbook*. O'Reilly Media, Inc., 1 edition, 4 2006.

[61] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang. Automated provisioning of BGP customers. *IEEE Network*, Nov/Dec 2003.

[62] T. G. Griffin, F. B. Shepherd, and G. T. Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Trans. Netw.*, 10:232–243, April 2002.

[63] T. G. Griffin and J. L. Sobrinho. Metarouting. In *Proc. SIGCOMM*, 2005.

[64] T. G. Griffin and G. T. Wilfong. An analysis of BGP convergence properties. In *Proc. SIGCOMM*, 1999.

[65] T. G. Griffin and G. T. Wilfong. Analysis of the MED Oscillation Problem in BGP. In *Proc. ICNP*, 2002.

[66] T. G. Griffin and G. T. Wilfong. On the correctness of IBGP configuration. In *Proc. SIGCOMM*, 2002.

[67] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543 (Proposed Standard), March 1999. Obsoleted by RFCs 3261, 3262, 3263, 3264, 3265.

[68] G. Herrero and J. van der Ven. *Network Mergers and Migrations: Junos Design and Implementation.* Wiley, 2010.

[69] Hewlett-Packard. Migration from Cisco IGRP and EIGRP to industry-standard OSPF. Technical White Paper, 2012.

[70] G. Iannaccone, C. Chuah, S. Bhattacharyya, and C. Diot. Feasibility of IP restoration in a tier-1 backbone. *IEEE Network*, 18:13–19, 2004.

[71] Donald B. Johnson. Finding All the Elementary Circuits of a Directed Graph. *SIAM J. Comput.*, 4(1):77–84, 1975.

[72] Juniper Networks. Configuring OSPF Routing Policy, 2010.

[73] Juniper Networks. Migrating EIGRP Networks to OSPF. White Paper, 2010.

[74] M. Kaminski, P. Medvedev, and M. Milanic. Shortest Paths Between Shortest Paths. *Theor. Comp. Sc.*, 412(39):5205 – 5210, 2011.

[75] E. Keller, J. Rexford, and J. Van Der Merwe. Seamless BGP migration with router grafting. In *Proc. NSDI*, 2010.

[76] E. Keller, M. Yu, M. Caesar, and J. Rexford. Virtually eliminating router bugs. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 13–24, New York, NY, USA, 2009. ACM.

[77] R. Keralapura, C-N. Chuah, and Y. Fan. Optimal Strategy for Graceful Network Upgrade. In *Proc. INM*, 2006.

[78] M. Kolon and J. Doyle, editors. *Juniper Networks(r) Routers: The Complete Reference*. Osborne/McGraw-Hill, February 2002.

[79] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of Internet stability and backbone failures. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 278 –285, 1999.

[80] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet Inter-Domain Traffic. In *Proc. SIGCOMM*, 2010.

[81] F. Le, S. Lee, T. Wong, H. Kim, and D. Newcomb. Minerals: using data mining to detect router misconfigurations. In *MineNet '06*, pages 293–298, 2006.

[82] F. Le, G. Xie, D. Pei, J. Wang, and H. Zhang. Shedding light on the glue logic of the internet routing architecture. In *Proc. SIGCOMM*, 2008.

[83] F. Le, G. Xie, and H. Zhang. Understanding route redistribution. In *ICNP*, pages 81–92. IEEE, 2007.

[84] F. Le, G. Xie, and H. Zhang. Theory and new primitives for safely connecting routing protocol instances. In *Proc. SIGCOMM*, 2010.

[85] S. Lee, T. Wong, and H. Kim. To automate or not to automate: On the complexity of network configuration. In *IEEE ICC 2008*, Beijing, China, May 2008.

[86] N. Leymann, B. Decraene, C. Filsfils, M. Konstantynowicz, and D. Steinberg. Seamless MPLS Architecture. Internet draft, 2012.

[87] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proc. SIGCOMM*, 2002.

[88] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjálmtỳsson, and A. Greenberg. Routing design in operational networks: A look from the inside. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 27–40. ACM, 2004.

[89] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, Y. Ganjali, and C. Diot. Characterization of failures in an operational IP backbone network. *Trans. on Netw.*, 16:749–762, 2008.

[90] M. Matuska. Metaconfiguration of the computer network. Technical Report 27/2004, CESNET, 2004.

[91] S. Mirtorabi, P. Psenak, A. Lindem, and A. Oswal. OSPF Multi-Area Adjacency. RFC 5185, 2008.

[92] J. Moy. OSPF Version 2. RFC 2328 (Standard), April 1998.

[93] J. Moy, P. Pillay-Esnault, and A. Lindem. Graceful OSPF Restart. RFC 3623 (Proposed Standard), November 2003.

[94] W. Muhlbauer, S. Uhlig, B. Fu, M. Meulle, and O. Maennel. In Search for an Appropriate Granularity to Model Routing Policies. In *Proceedings of ACM SIGCOMM*, 2007.

[95] R. Musunuri and J.A. Cobb. A complete solution for iBGP stability. In *Proc. ICC*, 2004.

[96] H. Nguyen, M. Roughan, S. Knight, N. Falkner, O. Maennel, and R. Bush. How to Build Complex, Large-Scale Emulated Networks. In *TRIDENTCOM*, pages 3–18, 2010.

[97] D. Oppenheimer, A. Ganapathi, and D. A. Patterson. Why do internet services fail, and what can be done about it? In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 1–1, Berkeley, CA, USA, 2003. USENIX Association.

[98] I. Oprescu, M. Meulle, S. Uhlig, C. Pelsser, O. Maennel, and P. Owezarski. oBGP: an Overlay for a Scalable iBGP Control Plane. In *Proc. IFIP Networking*, 2011.

[99] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), February 1990.

[100] P. Pan, G. Swallow, and A. Atlas. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090 (Proposed Standard), May 2005.

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

[102] J. H. Park, P. Cheng, S. Amante, D. Kim, D. McPherson, and L. Zhang. Quantifying i-BGP Convergence inside Large ISPs. Technical report, UCLA, 2011.

[103] H. Peine and R. Schwarz. A multi-view tool for checking the security semantics of router configurations. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, page 56, Washington, DC, USA, 2003. IEEE Computer Society.

[104] C. Pelsser, T. Takeda, E. Oki, and K. Shiomoto. Improving route diversity through the design of iBGP topologies. In *Proc. ICC*, 2008.

[105] C. Pelsser, S. Uhlig, T. Takeda, B. Quoitin, and K. Shiomoto. Providing scalable NH-diverse iBGP route redistribution to achieve sub-second switch-over time. *Comput. Netw.*, 54(14):2492–2505, 2010.

[106] M. Pizzonia and M. Rimondini. Netkit: easy emulation of complex networks on inexpensive hardware. In *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, TridentCom '08, pages 7:1–7:10, 2008.

[107] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[108] T. Przygienda, N. Shen, and N. Sheth. M-ISIS: Multi Topology (MT) Routing in Intermediate System to Intermediate Systems (IS-ISs). RFC 5120 (Proposed Standard), February 2008.

[109] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Multi-Topology (MT) Routing in OSPF. RFC 4915 (Proposed Standard), June 2007.

[110] B. Quoitin. *BGP-based interdomain traffic engineering*. PhD thesis, Université catholique de Louvain, August 2006.

[111] B. Quoitin and S. Uhlig. Modeling the Routing of an Autonomous System with C-BGP. *IEEE Network*, 19(6), November 2005.

[112] R. Rastogi, Y. Breitbart, M. Garofalakis, and A. Kumar. Optimal configuration of OSPF aggregates. *IEEE/ACM Trans. Netw.*, 11(2):181–194, April 2003.

[113] R. Raszuk, R. Fernando, K. Patel, D. McPherson, and K. Kumaki. Distribution of diverse BGP paths. Internet Draft, 2011.

[114] A. Rawat and M. A. Shayman. Preventing Persistent Oscillations and Loops in IBGP Configuration with Route Reflection. *Comput. Netw.*, 50:3642–3665, December 2006.

[115] S. Raza, Z. Yuanbo, and C.-N. Chuah. Graceful Network State Migrations. *Trans. on Netw.*, 19(4):1097 –1110, 2011.

[116] S. Raza, Y. Zhu, and C.-N. Chuah. Graceful Network Operations. In *Proc. INFOCOM*, 2009.

[117] M. Reitblatt, N. Foster, J. Rexford, and D. Walker. Consistent Updates for Software-Defined Networks: Change You Can Believe In! In *Proc. HotNets-X*, 2011.

[118] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[119] Infonetics Research. The Costs of Enterprise Downtime, North America. White paper, 2004.

[120] Infonetics Research. The Costs of Downtime: North American Medium Businesses. White paper, 2006.

[121] M. Roughan, W. Willinger, O. Maennel, D. Perouli, and R. Bush. 10 Lessons from 10 Years of Measuring and Modeling the Internet's Autonomous Systems. *Selected Areas in Communications, IEEE Journal on*, 29(9):1810 –1821, october 2011.

[122] J.-L. Le Roux, J.-P. Vasseur, and J. Boyle. Requirements for Inter-Area MPLS Traffic Engineering. RFC 4105 (Informational), June 2005.

[123] A. Shaikh, R. Dube, and A. Varma. Avoiding instability during graceful shutdown of multiple OSPF routers. *Trans. on Netw.*, 14:532–542, June 2006.

[124] M. Shand and S. Bryant. IP Fast Reroute Framework. RFC 5714 (Informational), January 2010.

[125] M. Shand and L. Ginsberg. Restart Signaling for IS-IS. RFC 5306, 2008.

[126] L. Shi, J. Fu, and X. Fu. Loop-Free Forwarding Table Updates with Minimal Link Overflow. In *Proc. ICC*, 2009.

[127] J. Siméon, J. Robie, D. Florescu, D. Chamberlin, M. Fernández, and S. Boag. XQuery 1.0: An XML query language. W3C recommendation, W3C, January 2007.

[128] J. L. Sobrinho and T. Quelhas. A Theory for the Connectivity Discovered by Routing Protocols. *Networking, IEEE/ACM Transactions on*, PP(99):1, 2011.

[129] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with rocketfuel. In *Proc. SIGCOMM*, 2002.

[130] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan. Traffic matrix reloaded: Impact of routing changes. In *Proc. PAM*, 2005.

[131] R. Teixeira and J. Rexford. Managing Routing Disruptions in Internet Service Provider Networks. *IEEE Comm. Mag.*, 44(3):160 – 165, 2006.

[132] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of hot-potato routing in ip networks. In *Proc. SIGMETRICS*, 2004.

[133] T. M. Thomas. *OSPF Network Design Solutions, Second Edition*. Cisco Press, 2003.

[134] S. Uhlig and S. Tandel. Quantifying the BGP routes diversity inside a tier-1 network. In *Proc. Networking*, 2006.

[135] B. Ujcich, K. Wang, B. Parker, and D. Schmiedt. Thoughts on the Internet architecture from a modern enterprise network outage. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 494 –497, april 2012.

[136] V. Valancius and N. Feamster. Multiplexing BGP sessions with BGP-Mux. In *Proc. CoNEXT*, 2007.

[137] V. Van den Schrieck, P. Francois, and O. Bonaventure. BGP Add-Paths : The Scaling/Performance Tradeoffs. *IEEE Jour. on Sel. Areas in Comm.*, 28(8):1299 – 1307, October 2010.

[138] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, March 2009.

[139] L. Vanbever, G. Pardoen, and O. Bonaventure. Towards validated network configurations with NCGuard. In *Proc. INM*, pages 1–6, 2008.

[140] L. Vanbever, S. Vissicchio, C. Pelsser, P. Francois, and O. Bonaventure. Lossless Migrations of Link-State IGPs. *IEEE/ACM Transactions on Networking*, 2012. (To appear).

[141] J. Vasseur, M. Pickavet, and P. Demeester. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[142] C. Villamizar, R. Chandra, and R. Govindan. BGP Route Flap Damping. RFC 2439 (Proposed Standard), November 1998.

[143] S. Vissicchio. Governing Routing in the Evolving Internet. PhD. Thesis, 2012. `http://www.dia.uniroma3.it/~compunet/www/docs/vissicchio-thesis-text.pdf`.

[144] S. Vissicchio, L. Cittadini, M. Pizzonia, L. Vergantini, V. Mezzapesa, and M. L. Papagni. Beyond the Best: Real-Time Non-Invasive Collection of BGP Messages. In *Proc. INM/WREN 2010*, 2010.

[145] S. Vissicchio, L. Cittadini, L. Vanbever, and O. Bonaventure. iBGP Deceptions: More Sessions, Fewer Routes. In *Proc. INFOCOM*, 2012.

[146] A. Voellmy and P. Hudak. Nettle: A Language for Configuring Routing Networks. In Walid Taha, editor, *Domain-Specific Languages*, volume 5658 of *Lecture Notes in Computer Science*, pages 211–235. Springer Berlin / Heidelberg, 2009.

[147] M. Vutukuru, P. Valiant, S. Kopparty, and H. Balakrishnan. How to Construct a Correct and Scalable iBGP Configuration. In *Proc. INFOCOM*, 2006.

[148] P. Walmsley. *XQuery*. O'Reilly Media, Inc., 2007.

[149] D. Walton, A. Retana, E. Chen, and J. Scudder. Advertisement of multiple paths in BGP. Internet Draft, July 2011.

[150] T. Wan and P. C. van Oorschot. Analysis of BGP prefix origins during Google's may 2005 outage. In *Proc. IPDPS*. IEEE Computer Society, 2006.

[151] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *Proc. SIGCOMM*, 2008.

[152] D. Wilcox, K. Chang, and V. Grover. Valuation of mergers and acquisitions in the telecommunications industry: a study on diversification and firm size. *Information & Management*, 38(7):459 – 471, 2001.

[153] Y. Yemini, A.V. Konstantinou, and D. Florissi. NESTOR: an architecture for network self-management and organization. *Selected Areas in Communications, IEEE Journal on*, 18(5):758 –766, may 2000.

[154] J. Yu. Scalable Routing Design Principles. RFC 2791 (Informational), July 2000.

[155] A. Zaheer, L. Johnson, M. Abe, and S. Faraz. *Troubleshooting IP Routing Protocols*. Cisco Press, 1 edition, 5 2002.

[156] R. Zhang and M. Bartell. *BGP Design and Implementation*. Cisco Press, 2003.

Webography

[157] BGP Routing Table Analysis Reports. http://bgp.potaroo.net/.

[158] Cisco 7200 Simulator. http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator.

[159] Configuring Virtual Routing and Forwarding. Official Cisco Documentation, Cisco Systems, Inc. www.cisco.com.

[160] Configuring VRF-lite. Official Cisco Documentation, Cisco Systems, Inc. www.cisco.com.

[161] Gurobi. http://www.gurobi.com/.

[162] IP Routing Protocol-Independent Commands. Cisco IOS IP Command Reference, Volume 2 of 3: Routing Protocols, 2006. www.cisco.com.

[163] OSPF -vs- ISIS. http://www.merit.edu/mail.archives/nanog/2005-06/msg00406.html. NANOG thread, 2005.

[164] World Internet Usage and Population Statistics. http://www.internetworldstats.com/stats.htm.

[165] Results of the GEANT OSPF to ISIS Migration. GEANT IPv6 Task Force Meeting, 2003. http://www.geant.net/eumedconnect/upload/pdf/GEANT-OSPF-to-ISIS-Migration.pdf.

[166] As the Value of Enterprise Networks Escalates, So Does the Need for Configuration Management. The Yankee Group. White paper Enterprise Computing Networking, 2004. http://www.cs.princeton.edu/courses/archive/spring12/cos461/papers/Yankee04.pdf.

[167] AT&T Ends Bid To Add Network Capacity Through T-Mobile USA Purchase, 2011. http://www.att.com/gen/press-room?pid=22146&cdvn=news&newsarticleid=33560&mapcode=corporate%7Cwireless-networks-general.

[168] GEANT Backbone Topology, 2011. http://www.geant.net.

[169] Seamless Network-Wide IGP Migrations, 2011. http://inl.info.ucl.ac.be/softwares/seamless-network-migration.

[170] Smart Management for Robust Carrier Network Health and Reduced TCO! NANOG54 Panel, 2012. http://www.nanog.org/meetings/nanog54/abstracts.php?pt=MTkwMyZuYW5vZzU0&nm=nanog54.

[171] A. Barnard. Internal glitches shut down Boston hospital for four days. Boston Globe, 2002.

[172] N. Bargisen and M. Lyngbol. Integrating Networks. NANOG41 Presentation, 2007. http://www.nanog.org/meetings/nanog41/presentations/integrating-network-TDC.pdf.

[173] M. Brown, C. Hepner, and A. Popescu. Internet captivity and depeering. NANOG45 Presentation, 2009. http://www.nanog.org/meetings/nanog45/presentations/Tuesday/Brown_Internet_Peering_N45.pdf.

[174] M. A. Brown. Renesys: Pakistan hijacks YouTube, 2008. http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml.

[175] B. Chapman. Automating Network Configuration. NANOG49 Presentation, 2010. http://www.nanog.org/meetings/nanog49/presentations/Sunday/Automating_Configuration_n49.pdf.

[176] J. Cowie. Renesys: China's 18-Minute Mystery, 2010. http://www.renesys.com/blog/2010/11/chinas-18-minute-mystery.shtml.

[177] D. Deitrich. Bogons and Bogon Filtering. NANOG33 presentation, Feb 2005. http://www.nanog.org/meetings/nanog33/presentations/deitrich.pdf.

[178] Equinix Direct. http://www.equinix.com/solutions/regional-solutions/americas/equinix-direct/.

[179] V. Gill and J. Mitchell. AOL Backbone OSPF-ISIS Migration. NANOG29 Presentation, 2003. http://www.nanog.org/meetings/nanog29/presentations/gill.pdf.

[180] V. Gill and D. Schmidt. Automated configuration management. NANOG55 Presentation, 2012. http://www.nanog.org/meetings/nanog55/presentations/Tuesday/Gill_Schmidt.pdf.

[181] V. Gill and M. Shields. Programatic networks - autogen. NANOG44 Presentation, 2008. http://www.nanog.org/meetings/nanog44/presentations/Monday/Gill_programatic_N44.pdf.

[182] O. Maennel, A. Feldmann, C. Reiser, R. Volk, and H. Böhm. AS-Wide Inter-Domain Routing Policies: Design and Realization. NANOG34 Presentation, 2005. http://www.nanog.org/meetings/nanog34/presentations/feldman.pdf.

[183] J. Moran. Smart Network Management. AOL's Take. NANOG54 Presentation, 2012. http://www.nanog.org/meetings/nanog54/presentations/Tuesday/Moran.pdf.

[184] Juniper Network. What's Behind Network Downtime? White paper, http://www.juniper.net/solutions/literature/white_papers/200249.pdf, 2008.

[185] Juniper Networks. Default Route Preference Values. http://www.juniper.net/techpubs/software/junos/junos94/swconfig-routing/default-route-preference-values.html.

[186] Juniper Networks. Multi-topology routing. White paper, 2010. http://www.juniper.net/us/en/local/pdf/whitepapers/2000308-en.pdf.

[187] NetworkWorld. AT&T says its needs T-Mobile spectrum. http://www.networkworld.com/news/2011/060911-att-says-its-needs-t-mobile.html.

[188] NetworkWorld. Level 3's Global Crossing buyout could save carriers, though challenges remain. http://www.networkworld.com/news/2011/041111-level3-global-crossing.html.

[189] NetYCE. Whitepaper: Enabling Network Services Provisioning. http://www.netyce.com/images/downloads/whitepaper%20yce%20-%20v1_final.pdf.

[190] University of Oregon. Route Views Project. http://www.routeviews.org/.

[191] I. Pepelnjak. Changing the Routing Protocol in Your Network, 2007. http://stack.nil.com/ipcorner/ChangingRoutingProtocol/.

[192] J. Qiu. SimBGP: Python Event-driven BGP simulator. http://www.bgpvista.com/simbgp.php.

[193] R. Raszuk. To Add-Paths or not to Add-Paths. NANOG48 Presentation, 2010. http://www.nanog.org/meetings/nanog48/presentations/Tuesday/Raszuk_To_AddPaths_N48.pdf.

[194] RIPE Routing Information Service (RIS). http://www.ripe.net/ris.

[195] S. Chapman. Computer outage leaves hospital in chaos. Computer World UK, 2008. http://www.computerworlduk.com/news/it-business/12162/computer-outage-leaves-hospital-in-chaos/.

[196] F. Shamim and K. Raza. Which routing protocol ? Comparison between OSPF ISIS. NANOG49 Presentation. http://www.nanog.org/meetings/nanog49/presentations/Sunday/Shamim_Which_Routing_N49.pdf.

[197] W. F. Slater. The Internet Outage and Attacks of October 2002. Whitepaper. Chicago Chapter of the Internet Society, 2002. http://www.isoc-chicago.org/internetoutage.pdf.

[198] P. Smith. BGP Techniques for Service Providers. NANOG50 Presentation, 2010. http://www.nanog.org/meetings/nanog50/presentations/Sunday/NANOG50.Talk33.NANOG50-BGP-Techniques.pdf.

[199] Sprint. Standard Maintenance Windows. https://www.sprint.net/index.php?p=support_maint_window.

[200] Cisco Systems. What Is Administrative Distance? http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094195.shtml.

[201] Cisco Systems. Multi-Topology Routing Configuration Guide, Cisco IOS Release 15.1S, 2011. http://www.cisco.com/en/US/docs/ios-xml/ios/mtr/configuration/15-1s/mtr-15-1s-book.pdf.

[202] P. Templin. Small Network Operator - Lessons Learned. NANOG45 Presentation, 2009. http://www.nanog.org/meetings/nanog45/presentations/Tuesday/Templin_lessonslearned_N45.pdf.

[203] The Internet2 Observatory Data Collections. http://www.internet2.edu/observatory/archive/data-collections.html.

[204] NANOG thread. IPv6: IS-IS or OSPFv3. http://mailman.nanog.org/pipermail/nanog/2008-December/006194.html, 2008.

[205] NANOG thread. OSPF vs IS-IS, 2011. http://mailman.nanog.org/pipermail/nanog/2011-August/039016.html.

[206] T. Underwood. Renesys: Internet-Wide Catastrophe-Last Year, 2005. http://www.renesys.com/blog/2005/12/internetwide_nearcatastrophela.shtml.

[207] T. Underwood. Renesys: Con-Ed Steals the 'Net, 2006. http://www.renesys.com/blog/2006/01/coned_steals_the_net.shtml.

[208] I. van Beijnum. Large ATT outage, 2002. http://www.bgpexpert.com/archive2002q4.php.

[209] L. Vanbever and G. Pardoen. NCGuard, 2008. http://inl.info.ucl.ac.be/softwares/ncguard-network-configuration-safeguard.

[210] E. Zmijewski. Renesys: Reckless Driving on the Internet, 2009. http://www.renesys.com/blog/2009/02/the-flap-heard-around-the-worl.shtml.