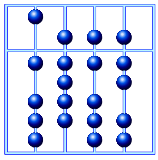


Lehrstuhl für Netzwerkarchitekturen
Fakultät für Informatik
Technische Universität München

Handling the complexity of BGP
via characterization, testing and configuration management

Olaf Maennel

Vollständiger Abdruck der von der Fakultät für Informatik
der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Bernd Brügge, Ph.D.
Prüfer der Dissertation:	1.	Univ.-Prof. Anja Feldmann, Ph.D.
	2.	Univ.-Prof. Timothy G. Griffin, Ph.D., Univ. of Cambridge / UK

Die Dissertation wurde am 29. 6. 2005 bei der Technischen Universität München
eingereicht und durch die Fakultät für Informatik
am 11. 10. 2005 angenommen.

Kurzfassung

In der heutigen Zeit hat das Internet eine überwältigende kommerzielle und soziale Bedeutung eingenommen; dennoch mangelt es an einem Verständnis der grundlegenden Routingprotokolle, wie des Border-Gateway-Protokolls (BGP). Komplexitäten entstehen zum einen dadurch, dass man das Problem "globale Erreichbarkeit" an vielen räumlich weit verteilten Komponenten lösen muss, zum anderen haben sie den Ursprung in der Tatsache, dass die Verkehrslenkungsstrategien (routing policies) eines autonomen Systems (AS) ständigen Veränderungen unterworfen sind, aus Gründen wie Verkehrskapazitätsplanung (traffic engineering) oder um kundenspezifische Wünsche zu erfüllen – ein fehleranfälliges Vorgehen.

In dieser Arbeit behandeln wir diese Probleme in mehrfacher Hinsicht:

Um das Problem der Netzwerkkonfigurationen zu lösen, haben wir ein System entwickelt, mit dem man die AS-weiten Verkehrslenkungsstrategien eines ASes umsetzen kann – im Gegensatz zum herkömmlichen Verfahren, wo die Strategien auf Komponentenbene umgesetzt wird. Damit wird eine Abstraktionsebene geschaffen, die auch viele Vorteile im operationalen Betrieb aufweist. Dies zeigt sich an unseren Erfahrung, die wir beim Einsatz des Systems im Netz der Deutschen Telekom gewonnen haben.

Da allerdings die Ausdrucksmöglichkeiten den Verkehr zu lenken sehr vielfältig sind, führt dies zu komplexen Interaktionen und Dynamiken, die sich auf das gesamte Internet auswirken. Um diese Dynamiken zu verstehen, stellen wir Methoden vor, die Administratoren helfen können, problematische Routingzustände zu identifizieren und zu lokalisieren.

Während die meisten der heutigen Verkehrslenkungsprobleme aus solchen Interaktion entstehen, so gibt es andere, die direkt von der Router-Software/Hardware ausgelöst werden – solche Problem hätte man in einem geeigneten Test-Labor beheben sollen, bevor das Gerät in das Produktionsnetzwerk aufgenommen wurde. Wir beschreiben einen BGP-Lastgenerator, welcher in einer Vielzahl von Gerätetests eingesetzt werden kann. Wir illustrieren seine Fähigkeiten exemplarisch daran, dass wir erklären, wie man komplexe Tests aufsetzt, ohne dass sich der Anwender in Details verliert.

Abstract

Even today, given the widespread usage and critical importance of the Internet, its basic routing protocols such as the Border Gateway Protocol (BGP) are poorly understood. This is in part an artifact of the complex interactions that arise from a distributed system that is administered locally to achieve a global task: reachability. In another part it has its origin in the fact that inter-domain routing policies of autonomous systems (ASes) often undergo constant adjustments for reasons of traffic engineering and/or to address specific customer wishes, an error prone approach.

In this thesis we address these problems in multiple ways:

The problem of policy configurations by developing a system that allows us to manage the overall routing architecture rather than each individual router. With this we raise the abstraction level from individual BGP configuration statements to an AS-wide routing policy.

The richness of policy expressions leads to complex interactions and dynamics that can be observed throughout the Internet. We present a methodology that helps operators to detect problematic routing conditions, and we discuss how to identify that AS which is responsible for an instability.

While we find that some of todays routing issues are stemming from protocol interactions, others are coming from router software/hardware problems that should have been detected in a test-lab before deployment in the operational network. We describe a BGP workload generator that can help in a wide variety of equipment testing. We illustrate the capabilities of the tool by showing how complex tests can be instantiated, without of losing the test engineer in the intricacies of the test setup.

Contents

1 Introduction

The Internet is a complex, highly-configurable, distributed system, which has become part of today's critical communication infrastructure. Companies rely upon this (new) medium for doing their business and it influences the daily lives of many people. While the Internet is technically complex and composed of many Autonomous Systems (ASes), it is the delivery of packets that matters for the end-user. To be able to offer connectivity to the "network of networks", Internet Service Providers (ISPs) interconnect with each other to exchange traffic.

Routing protocols, such as BGP [1] or OSPF [2] / ISIS [3] build the necessary foundation to direct the packets through the networks. The increased competition and widely deployed services, such as VPNs or VoIP, require performance guarantees, which leads to tight Service Level Agreements (SLAs). This means ISPs build and manage their network according to the traffic matrix. But, network traffic is difficult to predict [4]. History has shown that traffic demands can change rapidly and drastically based on at-the-time popular applications. Examples include the rapid rise in HTTP traffic after the introduction of the Mosaic and Netscape browsers, the recent explosion of file-sharing services, and heavy traffic loads generated by worms, viruses and DDos attacks.

ISPs strive to accommodate varying traffic loads and still build networks that are robust to link and router failures. Besides adhere to resilience and performance, they often try to balance the traffic within their networks.

The demand for control over traffic flows does not stop at the doors of an ISP. Large companies build networks on top of the physical infrastructure (e.g., VPNs). Others offer a widely varying range of services to customers – for example, Akamai is using DNS to route traffic [5]. One may ask how many other stub networks will soon deploy sophisticated traffic engineering mechanisms on their own? Or may even end-users try to obtain flexible routing in terms of control over cost and performance of network paths (e.g., overlay networks)?

The above considerations leads to the question who directs traffic flows? How does this impacts the routing system (e.g., [6])? What dynamics arise between inter-, intra-domain routing, overlay networks and the emerging mobility of end-users?

Before one is able to answer such questions, we need an in-depth understanding of the pieces of the puzzle. One piece addressed in this thesis is the inter-domain routing architecture, and its basic routing protocol, the Border Gateway Protocol (BGP).

BGP is the de-facto standard inter-domain routing protocol. Its main propose is to distribute reachability information, while at the same time allowing a flexible control over routing decisions. This means a policy routing protocol has to bridge the gap between the technical realization to guide the packets to their destinations and the different commercial, political, social, etc. interests of the participating networks.

This comes at a price: a distributedly controlled system composed of about $20,000$ competing ASes is hard to debug; and its dynamic behavior is difficult to predict. How can we deploy

mission critical applications in the Internet, when we do not understand why packets do not follow the expected path through the network [7]? How can we make the Internet a robust and fault-tolerant network, when we do not have good metrics to evaluate the quality of routing system [8]? How can we estimate the impact of routing changes, when we do not know how to estimate the inter-domain traffic matrix [9]? (See [10] for more research questions.)

Yet, given the critical importance of the Internet, ISPs, router vendors, and researchers have to work together to find solutions to those problems. To evolve the Internet, it is necessary to understand and fix problems such as long convergence times (e.g., [11]); protocol dynamics (e.g., [12]); which can lead to performance disruptions for a substantial amount of traffic (e.g., [13–15]); and those unforseen interactions between policies (e.g., [16]). Furthermore, ISPs can benefit from systems that instantiate a valid (e.g., [17]) policy inside their network automatically (e.g., [18]), for example to avoid misconfigurations (e.g., [19]); as well as predict the implication of policy changes on the traffic shifts (e.g., [20, 21]). In addition, it is difficult to test protocol and service interactions on the network equipment in a test-lab under field conditions (e.g., [22]).

In this thesis we address those problems in multiple ways: via characterization of BGP dynamics, router testing and proposing a system for configuration management.

1.1 Outline

We start in Chapter 2 with some background information about today's Internet routing architecture. In particular we look at the basics of the BGP protocol, how to configure a router and discuss some related work.

Next, in Chapter 3, we develop a system that allows us to manage the overall routing architecture rather than each individual router. This includes how routing policies of autonomous systems (ASes) are specified. With this we raise the abstraction level from individual BGP configuration statements to a AS-wide routing policy. Our system enables an autonomous system (i) to explicitly specify its inter-domain AS-wide routing policy as first class entities (an extensible collection of individual policies and services such as a peering policy, a filter-martians policy, a signaled black-hole service, etc.); (ii) to specify its routing policy independently of the current state of the network; (iii) to automatically generate the appropriate pieces of the router configurations for all routers in the AS, even routers of different vendors; (iv) to impose a clear separation of tasks that is aligned with the organizational boundaries within an ISP; (v) to automatically generate a documentation of the current active routing policy in RPSL; (vi) to enable customers of the AS to apply changes to the route-sets they announce without any explicit human-to-human interaction. Initial deployment of the system to manage the AS-wide routing policy of Deutsche Telekom affirm the above advantages in an operational setting.

Each AS has its own routing policy, which tries to optimize its own cost, performance, reachability and reliability. Yet, the richness of policy expression in such a well-connected system like the Internet leads to unforeseen effects. This means that the intended and the actual outcome of a policy setting do not necessarily match. The result are complex interactions which lead to dynamics between the ASes that can be observed on multiple vantage points throughout the network. This even leads to severe problematic routing conditions, where the

routing system in itself has no, or more than one stable solution [23]. In Chapter 4 we first review the ingredients that cause BGP dynamics and then study the convergences properties by analyzing raw BGP update traces, in Chapter 4. Such insights in the behavior of BGP help operators to detect problematic routing issues (e.g., divergent, hijacked prefixes).

To further understand the BGP dynamics, we ask the question of the origin of all these updates and if these can be inferred just by observing the control plane of the Internet. We discuss a methodology for identifying the AS that is responsible when a routing change is observed and propagated by BGP in Chapter 5. The origin of such routing instabilities is deduced by examining and correlating BGP updates for many prefixes gathered at many observation points. Although interpreting BGP updates can be difficult and easily misleading, we find that we can pinpoint the origin to either a single AS or a session between two ASes in most cases. For this we developed several heuristics to cope with the limitations of the actual BGP update propagation process and monitoring infrastructure, and apply our methodology and evaluation techniques to actual BGP updates gathered at hundreds of observation points. Furthermore, we performed simulations to evaluate the inference quality achieved by our approach under ideal situations and compared how this correlates with the actual quality and the number of observation points.

While we find that some of todays routing issues are stemming from routing protocol interactions, others are coming from router software/hardware problems that should have been detected in a test-lab before deployment in the operational network. In Chapter 6 we develop a methodology for investigating the relationship between BGP pass-through times and a number of operationally important variables. We explain under what conditions, such as router CPU load, number of BGP peers, etc., this can result in unusually high delays and thus long convergence times.

Measuring BGP pass-through times is an example that can be view from a broader perspective: testing router software implementations, evaluating performance, understanding scalability and data-plane convergence. We believe that we can better answer the needs of test engineers by:

- generating control and data plane traffic that is statistically similar, in both quantitative and timing terms, to observed data
- generating control traffic protocol mix that reflects current operating practice and demand for a given service (e.g., VPNs)
- making multi-protocol tests easier to specify, setup and execute.

In Chapter 7 we describe a tool that generates synthetical BGP update traces for multiple peering sessions that can be targeted towards a few devices under test (DUTs). The tools is user-friendly, highly flexible and supports well-specified test conditions. The main goal is that it is easy for a test engineer to instantiate a complex BGP test. Reasonable defaults are assumed, instead of losing the test engineer in the intricacies of the BGP test setup. Such a tool has to create an initial-test setup that respects user wishes (e.g., number and type of peers) and at the same time is able to reflect some of the variability of the Internet (e.g., number of prefixes, AS path length, usage of communities). This means being able to derive from data characterization a normal setting for a BGP workload generator. With such an approach it becomes easy for a test engineer to set up a test in a lab that reflects, to some degree, the dynamics of the real Internet; more and more complex tests can be constructed, involving multiple services, and this in turn adds confidence in the robustness of the router design and

implementation.

This thesis concludes with a brief summary in Chapter 8. Our main contribution is a framework for handling the complexity of BGP. This work shows how operators and vendors can benefit from an in-depth understanding of BGP; and how researchers can improve BGP. Furthermore, we provide an outlook, in Chapter 9, how this work may evolve in the future and show how other research areas may benefit from our insights.

1.2 Vorveröffentlichte Teile der Dissertation (Previously Published Work)

Some parts of this thesis have been already published:

Alexander Tudor, Olaf Maennel, Anja Feldmann, and Hongwei Kong.
Towards more Realistic Router Testing.
Agilent Internal Report, Melbourne (Australia), June 2005.

Hagen Böhm, Anja Feldmann, Olaf Maennel, Christian Reiser, and Rüdiger Volk.
AS-Wide Inter-Domain Routing Policies: Design and Realization.
Technical Report, TU München (Germany), June 2005.

Anja Feldmann, Olaf Maennel, Z. Morley Mao, Arthur Berger, and Bruce Maggs.
Locating Internet Routing Instabilities.
In Proceedings of ACM SIGCOMM, Portland (USA), August 2004.

Anja Feldmann, Hongwei Kong, Olaf Maennel, and Alexander Tudor.
Measuring BGP Pass-Through Times.
In Proceedings of Passive & Active Measurement Workshop, Antibes Juan-les-Pins (France), April 2004.

Olaf Maennel, Alexander Tudor, Anja Feldmann, and Sara Bürkle.
Observed properties of BGP convergence
Talk at RIPE 45, Barcelona (Spain), May 2003.

Some parts of this work went also in master theses:

Christian Reiser.
Network-Wide Inter-Domain Routing Policies: Design and Realization.
Diplomarbeit, TU München (Germany), June 2005.

Sara Bürkle.
BGP convergence analysis.
Diplomarbeit, Saarland University, Saarbrücken (Germany), June 2003.

Acknowledgments

Acknowledging every person that contributed directly or indirectly to the realization of this thesis is not possible. I have benefited from too many persons to ever be able to mention them all.

I like to start with the networking architecture group at the Technische Universität München. Foremost I like to thank my thesis advisor, Anja Feldmann. I am extremely grateful for her generosity, confidence, motivation, advice and endless support. I started working with Anja, and most of my colleagues, already during my master studies at the Saarland University in Saarbrücken. Without their invaluable ideas, knowledge and feedback this thesis would simply not exist. I cannot express in words how grateful I am, and I will never be able to repay this debt. Beside from the fun it was working in this group, most of them became good friends.

With regards to friendship I also like to mention Steve Uhlig, who is currently a researcher at UCL in Belgium. He gave me a lot of very inspiring thoughts (not only on this thesis). Furthermore, I like to thank Alexander Tudor from Agilent Labs. Although he was never an official advisor his support kept me going for several years. His enthusiasm and commitment to our router testing project (see Chapters 6 and 7) greatly motivated me and gave me direction. He always found time to discuss problems, and helped me bridge the gap between the work that is on the one hand relevant for research and on the other hand beneficial to practioners.

I am also very thankful to Timothy G. Griffin – it is a great honor that he is part of my thesis committee. I met him at a few conferences and workshops and he always found time to discuss with me. He encouraged me, gave me faith and with that he strengthened my personality as a researcher.

I would like to express special thanks to Bruce Maggs for all that he has done for me. Behind his jolly appearance shines an incredible brilliance. He gave me amazingly insightful comments and always found all the time that is needed to explain and/or discuss a problem. As he was the advisor of Anja, he is for me my “grand-advisor”.

It was also good fortune to meet Zhuoqing Morley Mao, who is an assistant professor at University of Michigan. We had great fun working and chatting together. Morley contributed a lot to this work, in particular to Chapter 5. In the same context I like to thank Arthur Berger and the Akamai staff for sharing their insights.

Let me also express my thanks to Rüdiger Volk and Hagen Böhm from Deutsche Telekom – without their operational insights, the system proposed in Chapter 3 would not have been realizable.

Furthermore, I am grateful to the folks at RIPE, especially Henk Uijterwaal and Matthew Williams, for their continuing support. Without the work of this group, the research community would be missing a significant number of BGP beacons as well as routing observation points. Furthermore, the quality of the provided data is outstanding.

Thanks also to Geoff Huston for taking his time at RIPE 48 and discussing with Alex and me the details of what matters for router testing. Without his help, the choice of variables used in Chapter 7 would be much less relevant for operational people.

I also like to thank other students in Anja's group working on related topics. Among them were Sara Bürkle, Christian Reiser and Wolfgang Mühlbauer. With Sara I started discovering the properties and behavior of the BGP protocol, with Christian I worked on the automated configuration system for the backbone of Deutsche Telekom and with Wolfgang I gained a better understanding of the Internet topology.

I am also thankful to the reviewers of my previously published papers. They provided valuable suggestions regarding the material itself as well as on how to improve its presentation. Furthermore thanks to Nils Kammenhuber, Arne Wichmann and Vinay Aggarwal for their comments on earlier drafts of this thesis. Special thanks belongs to Wolfgang Mühlbauer for his help and support with the final version and publication of this work.

2 Internet Routing Architecture

The Internet is a collection of many independently administrated routing domains. Each routing domain is composed of multiple networks operated under the same authority.

Connectivity within and between such routing domains is accomplished via the Internet Protocol (IP) addresses. Currently there are two types in active use: IP version 4 (IPv4) and IP version 6 (IPv6). IPv4 was initially deployed in 1983 and is still the most commonly used version. IPv4 addresses are 32-bit numbers often expressed as 4 octets in "dotted decimal" notation (e.g., `127.0.0.1`). Deployment of the IPv6 protocol began in 1999. IPv6 addresses are 128-bit numbers and are conventionally expressed using hexadecimal strings (e.g., `2001:db8::dead:beef`). Packet forwarding is based on an initial *prefix* of the IP address that is being used for routing decisions. Prefix-based addressing is in use since the early beginnings of IP [24] in classful routing, and has evolved into supernetting [25] and CIDR [26]. Nowadays a CIDR prefix is written in "slash-notation" providing the network address and a variable length subnet mask (e.g.,`2001:db8::/32`). IP addresses are allocated by IANA [27] from pools of unallocated address space and delegated to the appropriate Regional Internet Registries (AfriNIC, APNIC, ARIN, LACNIC, RIPE NCC) or National Internet Registries. They in turn distribute their address space to the Local Internet Registries and they again to ISPs. See RFC 2050 [28] for more information.

Today, a default-free routing table in the Internet contains roughly $200,000$ prefixes[1], which are announced by independently administrated routing domains. For example a routing domain can be an enterprise network, a campus network or an ISP. Those routing domains are often referred to as *autonomous systems* (ASes), yet note that there is no trivial mapping between a routing domain and an *AS number* (ASN), which is handed out by the route registries. Companies may operate several ASNs or even several companies may appear in the routing system under one single ASN. Nowadays there are roughly about $21,000$ ASes[1], while only about $3,500$[1] sell Internet connectivity to other ASes.

Each AS is composed of a collection of routers that are interconnected, using different link layer technologies including Synchronous Optical Networking links (SONET/SDH), and/or Ethernet. A router consists of many components, a switching fabric, a set of line cards (sometimes equipped with their own CPUs), and a main route processor. Packets enter the router via one of the line cards and leave the router via some other line card as determined by the *Forwarding Information Base* (FIB). The FIB is often situated directly on the line cards of the routers and is constructed from the routing tables computed by the various routing protocols. Inside an AS reachability information is propagated by an Interior Gateway Protocol (IGP) [2], among them are ISIS, OSPF, EIGRP, RIP. Each router selects a shortest path to the destination according to a metric chosen by the network administrator. We distinguish the *core links* that interconnect the routers within the AS and the *edge links* that cross AS

[1] Numbers are rough estimates.

boundaries. The routers where edge links are terminated are called *border routers*. The locations of these border routers are usually called Points Of Presence (PoPs). An AS typically buys Internet connectivity from one or more transit providers. Such providers are often called *upstreams*. Contractual relationships between ASes can be very complex. Beside the afore mentioned *customer-provider* relationship there is also a category referenced to as *peering* relationship[2]. Peers usually share the link cost between them. In addition, the peering link is only used to exchange traffic with the peer and its customers. No transit traffic should flow through the peering links [29–31]. We call an AS that has no upstream provider a *tier-1*. As a consequence all tier-1 provider more or less have to peer with each other[3] and therefore build the core of the Internet, while ISPs that do not provide transit services, and "simple" customers, e.g., multi-homed ASes, are at the periphery. Often the AS graph is depicted with the core AS at the top and the periphery at the bottom.

For the technical realization of such complex policies a *policy routing protocol* is deployed, also referenced to as an Exterior Gateway Protocol (EGP). Today, BGP [32] is the de facto standard inter-domain routing protocol used in the Internet. Its main propose is to exchange reachability information between different ASes while at the same time allowing complex economic relationships. The next section discusses BGP in more details.

2.1 BGP basics

The Border Gateway Protocol (BGP) [1] was designed as a successor to the Exterior Gateway Protocol [33]. It is a variant of the class of distance-vector protocols, where neighboring routers exchange link cost information to destinations. BGP itself is a so-called *path-vector protocol*. A *route advertisement* indicated the reachability of a network. To avoid cycles and to provide a distance-metric the AS number of each AS on the path to the destination is propagated along with the route advertisements.

The signaling of reachable destination prefixes is accomplished over a TCP session between the two BGP speaking routers. BGP is stateful, which means that when a BGP session comes up, first all *best routes* are exchanged. Afterwards, only *incremental updates* are sent whenever the current best route changes. There are two variants of BGP. The *eBGP* variant is used to announce the reachable prefixes on a link between routers that are part of distinct ASes. The *iBGP* variant is used to distribute the best BGP routes inside an AS. As the AS path inside an AS is not modified, it cannot be used to avoid routing loops and thus a full mesh of all BGP speaking routers or route reflection [34] is often configured.

Four kinds of messages can be exchanged between two BGP speakers:

OPEN is used to open up a BGP session.

KEEPALIVEs are used between neighbors to make sure that the connection still persists during periods of inactivity.

UPDATE messages carry network reachability information. An update either advertises a prefix or withdraws a previously announced prefix. Multiple announcements and/or withdraws can be packed into one BGP UPDATE packet.

[2] Note that the word "peer" is often used ambiguously – on one side it just means any *BGP neighbor* (for example as defined in the BGP-RFC 4271 [1]), on the other side common practice may mean the economic relationship where only customer prefixes are propagated.

[3] See [17] for a nice counter-example.

NOTIFICATION is used to tear a BGP session down in case of an error.

If a neighbor advertises a prefix, this can be seen as a commitment from the sending neighbor that it can reach the specified destination. By withdrawing a prefix, the sending neighbor indicates that it can no longer reach the destination (or does not want to carry the traffic towards this destination anymore). Every time the best route changes all neighbors that received an announcement of this route have to be informed about the change.

Routes learned via BGP have associated attributes that are used to determine the best route to a destination when multiple paths exist to a particular destination. These properties are referred to as *BGP attributes*. While BGP allows the network operators to modify attributes or remove some of the attributes, it also distinguishes between *transient* and *non transient* attributes – which means attributes that can be passed on to another AS, or that cannot, respectively. In the following we provide a short summary of how some BGP attributes work, for a detailed discussion see,e.g., [32].

A route advertisement for a particular prefix includes an ordered sequence of ASes that constitute the *AS path*. Whenever a BGP border router propagates a route to a neighbor, the ASN will be prepended to the AS path. As a consequence the AS that *originated* the prefix is at the end of the AS path, while transiting ASes follow from right to left. The information contained in the AS path attribute concerns only the traffic that goes from the local AS towards the prefix. The actual path used in the reverse direction may not to be the same due to local policies enforced along the path between the AS and the destination prefix. Note that the AS number used in BGP does not necessarily corresponds to the routing domains created by the network operators.

Large ISPs may operate serval ASNs, for example to keep traffic local to certain regions (e.g., one in the US, one in Europe, and one in Asia or the Pacific Region); or small companies may simply use the ASN of their upstream provider to avoid the hassle of applying and maintaining their own AS number.

The `NEXT_HOP` attribute tells the router to which IP address it should forward packets. This does not necessarily have to be the neighboring BGP speaker since BGP allows third-party next hops. Within an AS the next hop attribute is not modified, thus it points to the chosen *exit router*[4]. Note that here BGP and IGPs are closely coupled, because the packets need to be able to "find their way" through the local network to reach the BGP exit point. For example if the path towards any exit point becomes unavailable in the IGP, all BGP routes using this exit must be considered unavailalbe and withdrawn. For more detailed information in this area see Teixeira et al. [13–15, 35].

There are other attributes available in BGP, e.g., the *Local Preference* attribute, the *Multiple Exit Discriminator* (MED) and *Community* attributes. The local preference attribute is used within an AS to implement local policies for the best exit point. It is used by some ISPs to influence outgoing traffic (e.g., prefer customers over peers over upstreams). With the Multiple Exit Discriminator, an AS can indicate the best entry point to its neighboring AS in case of multiple connections. The community attributes [36] can be used to "color" routes and to organize them into classes. While all other BGP attributes have a well specified semantic,

[4] Note that the exit router is by default the border router of the remote AS. Yet often *next-hop-self* is configured by the administrator, which indicates that the exit router is the border router of the local AS. This has the advantage that the link between the border routers of both ASes does not have to be carried in IGP.

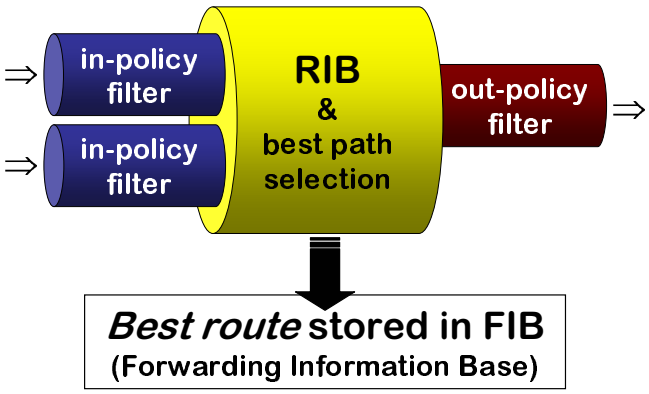


Figure 2.1: Update processing per router.

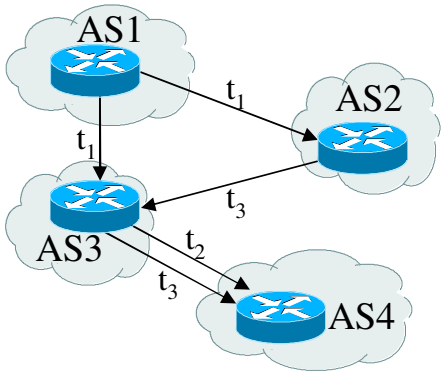


Figure 2.2: Example: update propagation.

the community attribute provides a large code space that can be freely used by a network administrator to define signaling within the domain and/or across domains.

2.1.1 BGP path selection and filtering

A router exchanges routing information about particular network addresses (prefixes) via BGP sessions in the following way (see Figure 2.1 as illustration). First, the *ingress filter* policies of the session over which a route is received decides whether to accept a route or not. If the received route is in accordance with the policy, the router may modify some of the attributes, and then stores it in the BGP routing table. The Routing Information Base (RIB) keeps all routes learned from the BGP neighbors. Then a BGP decision process inspects the attributes to select a preferred route. If the "best" route changes, then the routing table is updated, and the new best route passes through the *egress filter* policies of all sessions. These can again rewrite the BGP attributes or restrict propagation. Finally, if it passes the filter the route is propagated. This process is referred to as route manipulation. Note that scalability is one of the reasons for not propagating alternative routes. The more alternative routes are available the "better" can be the path selection but at the same time all alternatives have to be stored on the router. This consumes memory and the amount of memory stored on each router is limited.

To select the best route among the set of routes for the same prefix in the RIB, a BGP speaker follows a set of selection criteria called the *best path selection algorithm*. Next we discuss 10 of the most prominent rules currently in use. These criteria are applied in the specified order until only one path remains.

1. *Next-hop* reachable? The first rule of the BGP decision process is to make sure that the next-hop is actually reachable. This is usually implemented by a periodic IGP lookup of the next-hops used by BGP. Note that an announcement is typically rejected if the next hop is not reachable[5].
2. Prefer highest `weight`: This rule is not RFC conform but some vendors, including Cisco and Juniper, allow administrators to configure a preference that is local to the router. Sometimes this is referred to as the "sledgehammer" among the best path selection rules. This rule is used rarely.

[5]Note that this is an important detail when we are going to test routers in a test-lab in Chapter 7.

3. Prefer highest `local-pref`: This attribute is an administrative cost specifying the preference among the different routes towards a given destination. Contrary to the `weight`, the `local-pref` is an BGP attribute and is propagated to iBGP neighbors but remains local to the AS. It is set by the border routers upon receiving the BGP route to the value configured by the administrator or to a default of 100.
4. Prefer locally originated routes: Routes that are locally originated by the router or redistributed from IGP by the router are preferred over routes that are learned from other routers.
5. Prefer routes with the shortest `AS path` length: This rule constitutes a distance metric. The best route is the one with the shortest AS path length. The reasoning behind this is the idea that the shortest AS path is also the shortest path to the destination (which does not have to be true in the Internet).
6. Prefer the path with the lowest `origin type`: `IGP` (means that the network layer reachability information was introduced into BGP by the IGP), is lower than `EGP` (the route is learned via an Exterior Gateway Protocol), which is lower than `INCOMPLETE` (the information is learned by some other means, often manual configuration).
7. Prefers the route with the lowest `MED` value: Routes without an MED attribute are considered to have the lowest possible MED value. The MED attribute is used to select a particular egress point in the local domain. This can be used for "cold-potato" routing [37]. Cold potato routing aims at carrying the traffic for as long as possible in the own network before handing it off to the neighbor. The input filter of the neighbor domain might override the MED value, so that the use of this attribute usually relies on a mutual understanding between the two neighboring ASes.
   While one might expect that MED values are only comparable if they are received from the same neighboring AS, some vendors offer to compare MEDs "non deterministically" [38]. Furthermore, MEDs are sometimes used as an internal metric (similar to `local-pref` but respecting the AS path length): As MED comparison is in the decision process after the AS path evaluation it is possible to set MED values at the ingress border routers according to the preference of the network administrator (e.g., to realize the policy: "prefer European peers over Asian peers over US peers, but only if the AS path length is of equal length").
8. Prefers eBGP routes over iBGP: The motivation behind this rule is to hand off traffic to other networks as early as possible. Clearly, handing a packet off at the local router is preferred over carrying it around in the local AS before handing it off.
9. Prefers routes with the lowest IGP cost to the egress point: This rule enables what is commonly termed "hot-potato" routing [13], which means that the local AS tries to get rid of the IP packet as soon as possible. The logic behind "hot-potato" routing is to minimize the resources consumption necessary for forwarding the IP packets that transit through the AS. Therefore the closes exit point is chosen (according to the IGP metric). This rule is useful for transit domains. Most router vendors implement some kind of "scanner processes", running periodically (e.g., once a minute), on their routers to see if the IGP cost has changed and then update the best BGP route accordingly. Note that this "scanner" provides a mechanism for IGP/BGP interactions (see [13–15] for more details).
10. "Tie-breaking" rules: The last rules of the BGP decision process are used when there are still several equivalent routes available. Note that vendors often implement them differently. Some BGP implementations break ties by preferring the routers received

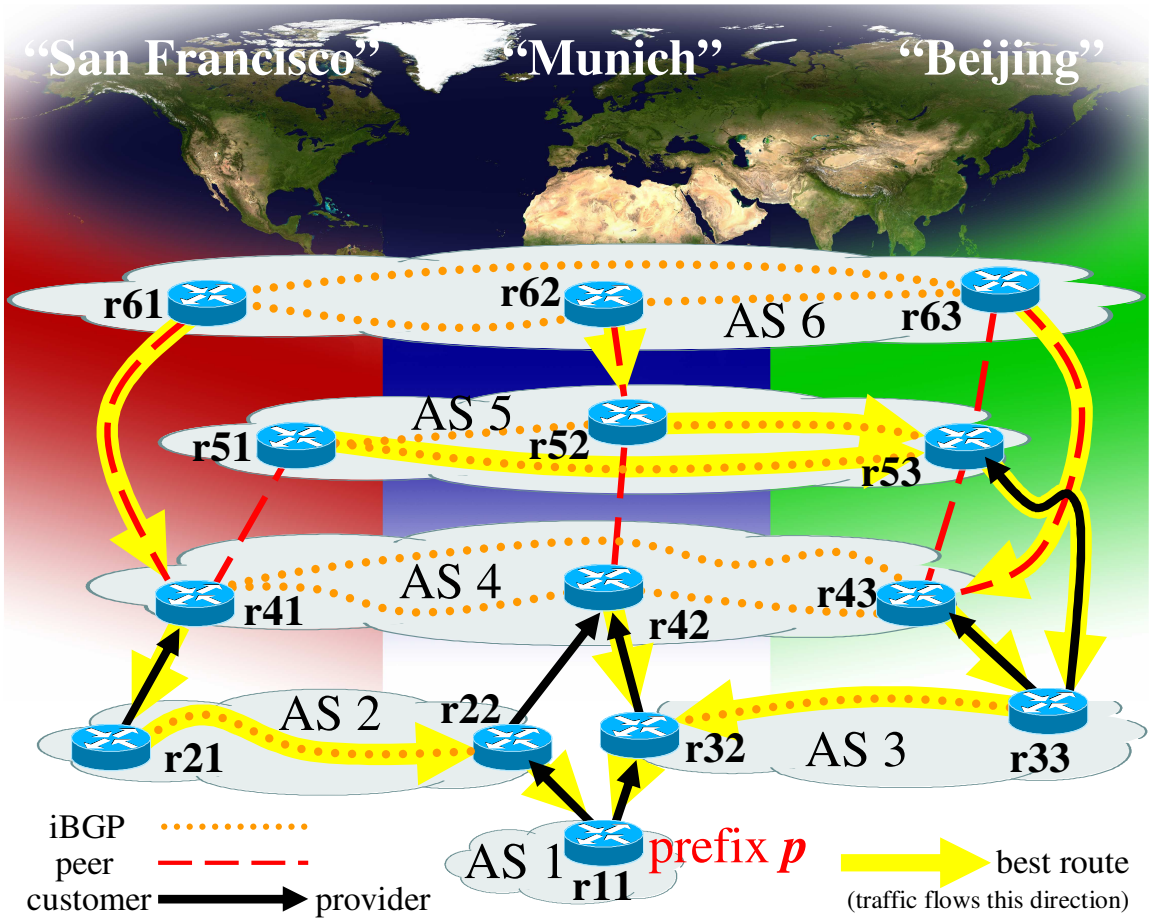


Figure 2.3: Example: Path sections of ASes[6].

from the router with the lowest router-id while other prefer the oldest route. These rules introduces some randomness in the inter-domain routing path selection.

Note that the best path selection process as stated above, summerizes just the typical behavior as implemented by most of the vendors. Some vendors implement slightly different variances and even allow operators to modify the selection process. For example, Cisco introduces the BGP Cost Communities rule [39], which changes the BGP best path selection process at the point of insertion (POI) by selecting the path according to the lowest cost community value. (Note that if the POI is not applied consistently throughout the AS, routing loops can occur.)

The best path selection process is executed on each router that runs BGP. This means that each router in a network picks one route from a possible set of alternatives. Traffic forwarding as well as route propagation is affected by this choice[7]. To improve the scalability inside the AS BGP Route Reflection [34] was designed. The idea is that Route Reflectors (RRs) aggregate route information and thus keep multiple alternative routes inside PoPs or within a certain geographic region. Only the best route, as picked by the best path selection process of the RR, is allowed to be distributed within the AS. A RR is a BGP speaker that reflects iBGP

[6] Images used with permission. (Thanks to `http://visibleearth.nasa.gov/` and `http://www.cisco.com/warp/public/503/2.html`).

[7] There are some efforts to allow a router to select multiple paths for load sharing purposes, see e.g., [40].

updates from RR-clients to other clients as well as to other RRs and also distributes iBGP updates from RRs to clients (but not to other RRs). RRs can be organized hierarchical and a full-mesh is only needed on the top-level of the hierarchy. Yet, routing loops cannot occur inside the AS[8]. To avoid single points of failures, redundant RRs can be configured.

The number of alternatives available to a router influences the route selection inside the Internet. To illustrate typical effects of the best path selection algorithm in the context of multiple ASes, consider the example shown in Figure 2.3. This simple illustration shows six ASes. AS1 is multihomed to AS2 and AS3, which both provide transit service for AS1. AS2 and AS3 are customers of AS4. AS3 is also a customer of AS5 (customer-provider relationships are indicated by solid black lines). AS4, AS5 and AS6 build the core of this "Internet-like" example, i.e., they can be considered as tier-1 ASes. They all peer with each other (peering links are indicated by dashed red lines). Keep in mind that ASes often cover the same geographical area while still being competitors. To illustrate this we have chosen three geographically distant location, "San Francisco", "Munich", and "Beijing". In the example of Figure 2.3 a router that is supposed to be located in "San Francisco" is labeled $r*1$, one from "Munich" is labeled $r*2$, and one located in "Beijing" is labeled $r*3$ (where the $*$ stands for the ASN of the AS). Note that AS4, AS5 and AS6 provide service in all three regions and peer in multiple places but not necessarily at every location.

AS1 announces its prefix p to both of its upstream providers. Both have no alternatives routes. Thus they both select the direct path. AS4 learns the route from its customers (AS2 and AS3) over all 4 links. Routers inside AS4 are free to choose their exit point (unless, for example, AS3 decides to use MEDs to signal that it wants to receive incoming IP traffic over the "Munich"-link **and** AS4 respects the MED setting). If MEDs are not used, a "hot-potato"-like behavior is often the default. This means traffic flows directly to the "closest" egress router[9]. In the case of AS4, traffic entering the router in San Francisco (router $r41$) leaves the AS via the "San Francisco"-link to AS2 (link $r41$ to $r21$). Traffic originated in Munich entering router $r42$ leaves the AS via one of the "Munich"-links. Note that this router can chose between two equally good paths – in the BGP sense. In this case the last "tie-braking" rule is used to pick one link, i.e., based on the remote router-id. Next consider how AS5 learns about prefix p. This situation is slightly different, because the path via Beijing has a shorter AS path length (furthermore AS3 is a customer and customers are often preferred via local-pref). Although the router $r52$, which is located in Munich, learns the prefix p over a peering link in Munch (link to router $r42$), and thus could deliver the traffic "locally" – all routers in AS5 may prefer to send the traffic for prefix p to Beijing, to be then handed-over via the customer link to AS3, just to be transfered back to Munich. Finally consider AS6: Traffic from router $r62$ (located in Munich) can be delivered via the peering link to AS5 (to router $r52$), because AS5 picked the customer link over AS3. But then traffic detours via Beijing. If the link in Beijing fails (between $r33$/AS3 and $r53$/AS5), then $r52$ (AS5) would use the peering link to AS4 and therefore withdraws the announcement given to AS6 (because a peer does not offer transit service for another peer). Ironically, in such a case, router $r62$ would chose as next hop either $r43$ or $r41$.

Note that if one login to all routers in AS6 to see which AS path is being used, one would gets different results: router $r61$ uses AS path "`6 4 2 1`", while router $r62$ propagates AS

[8] If appropriately configured, see [41] for detailed information.

[9] Note that the "closest" egress router is determined based on the IGP metric.

path "6 5 3 1" and router $r63$ uses the AS path "6 4 3 1" as best route. Such a simple example, as illustrated in Figure 2.3, shows how complex the interaction in the Internet can be.

In addition, to the complexity induced by path choices, one also needs to consider the dynamics that arise from the propagation of updates. The example in Figure 2.2 (see page 10) shows how "one update" propagates through the network can spawn multiple updates at remote ASes. In Figure 2.2 one new prefix p is announced at AS1. This is called the *triggering event*. A BGP update for prefix p is therefore sent to AS2 and AS3. This update is received by AS3, added to the routing tables, and sent onward to AS4. AS2 also receives an update from AS1 but the propagation through AS3 may take a bit longer (e.g., due to geographical distances) and thus AS3 does not receive this update immediately. Once AS3 receives the update from AS2 it may prefer this path for some reason (e.g., maybe routes from AS3 get a higher local-pref value) and re-updates its routing table and sends a second update for prefix p to AS4. In this rather simple example AS1 added one prefix (triggering event), yet AS4 is sending two updates for the same root cause. This is just one of many examples how updates can be spawned (see as well [12]).

The *Minimum Route-Advertisement Interval* (MRAI) [1] is used to rate limit outgoing BGP updates. The idea is to first receive "all" BGP updates from ingress neighbors (within some specified time), compute the best path and then propagate only one BGP update[10]. The RFC suggests that after one update for one prefix is sent to one peer, there should be a jittered[11] delay of 30 seconds before another update for the same prefix is sent to the same peer. This limits the number of BGP messages that need to be exchanged.

Another way to rate-limit updates is *BGP Route Flap Damping* [43,44]. The goal is to reduce the impact of routing oscillations and therefore the processing load of routers. To achieve this the router collects statistics about announcements and withdrawals of prefixes. Each time a prefix is withdrawn, the router increments the so-called damping penalty for the prefix by a fixed amount (Cisco / Juniper penalty: 1,000). Whenever the router observes an announcement, the router also increments the penalty (Cisco: 500 / Juniper: 1,000/500). Once the penalty exceeds the cutoff threshold the path is no longer used and the prefix is suppressed (Cisco: 2,000 / Juniper: 3,000). This means that after the prefix enters the suppressed state, the prefix is withdrawn – regardless of whether the route is still valid or not. Any flap afterwards incurs additional penalty increments until some maximum is reached (Cisco: 12,000, which corresponds to 60 min). Once the prefix stops flapping, the penalty is decremented over time using an exponential decay until the reuse threshold is reached (Cisco / Juniper: 750). Once the penalty falls below this reuse threshold, the suppressed path is re-advertised to appropriate BGP neighbors. This can lead to route suppression for more than one hour [45,46].

We note that certain vendor-specific implementations differ from the recommendation in the RFC. These cause different propagation patterns. For example Cisco's MRAI implementation differs in at least two major points from the RFC. The first difference is that the timer is implemented on a per-peer basis instead of a per-prefix basis. Scalability reasons do not

[10] Note that this is also used to collect multiple prefixes that can be packed in one BGP update packet.

[11] A jittered timer is a timer that uses randomly varying values. The MRAI, for example, has a typical value of 30 seconds. But if all routers would use a 30 seconds interval timer, this would lead to self-synchronization. This means that all routers send their updates at the same time and pace, every 30 seconds [42]. This is an undesired effect. It can be avoided by varying the timer interval randomly at each router. The jittered timer is implemented in such a way that values between 25 and 31 seconds are normal.

allow an implementation per peer and prefix. As a result almost all outgoing updates will be delayed – not just two consecutive updates (close in time and belonging to one prefix). The second difference is that the MRAI is applied to withdraws as well as announcements. Another example is the MRAI in Junipers routers, called "out-delay" [47], is disabled by default. That means, Juniper is not holding back any BGP update messages. While this speeds up convergence, the risk is that many more updates will be send – which can trigger more route flap damping. The trade-off here is between faster propagation vs. more protocol messages. It is clear that in todays Internet more protocol messages lead to more damping, which does not improve convergence. Even in a fictional Internet without damping, more protocol messages burn more CPU time. We look into this trade-off in more detail in Chapter 6.

2.1.2 Passive BGP data collection architectures

The principle idea behind collecting and archiving BGP data is to provide operators with a view of the network from different locations inside the Internet. This is essential for troubleshooting network problems. Furthermore, researchers can benefit from the data to study protocol behavior. There are two types of data available: BGP table dumps and BGP updates. The former provides a snapshot of the RIB, while the latter contains a time series of the changes to the routes. A table dump can be gathered by "logging" into a router (a real router, or a software BGP speaker) and querying the RIB. Regarding BGP updates, there are three different techniques: First, one dumps an IP packet trace of TCP port 179 on the link between two BGP speakers. Another way is to setup a PC with a software pseudo-BGP speaker, called *BGP collector*, that peers with a router in the operational network. Beside recording the original BGP update packet, a timestamp and both end points are archived. Note that this is considered to be a passive collector, even though it maintains a BGP session with a router in the production network. The collector only receives and archives all messages it receives from its neighbors it does not inject any updates. The third mechanism is by providing access to a router, which can then be queried for a RIB dump or various statistics.

Public BGP archives are available since September 1999. Additional collectors are added frequently. As the growing number of collectors provide different views of the Internet, BGP analysis has become more representative over time. RIPE (Réseaux IP Européens) [48] maintains a BGP collection infrastructure. They offer about 14 different *Remote Route Collectors (RRC)* located at different places in the world. Each box has a number of BGP feeds – in total there are several hundreds BGP feeds[12] available. Another public source is Oregon's Routeviews Project [49]. Beside the public collection points a number of ASes set up their own proprietary BGP collectors. This can be looking glasses, trace collections, or as in the case of Akamai [5] remote collectors. Akamai has collectors in about 500 ASes. Some of these feeds are full-feeds, some are partial feeds, some are even iBGP feeds, and often there are multiple observation points within an AS. All three mentioned archives do not only record BGP traffic, but also snapshots of the BGP tables.

The placement of the collectors in the topology is important. Recall, customer-provider and peering policies, outlined above, have certain implications regarding connectivity. At the top, the connectivity is excellent – many alternative paths of the same AS-path length are available. Closer to the bottom, this diversity is significantly restricted. Furthermore, since

[12] They provide a mix of full-feeds and "peering"-feeds.

customer ASes often have a primary connection to one AS and a backup connection to another AS, the connectivity is further reduced, as the backup path may not be visible to most of the Internet unless a failure close to the customer occurs. Accordingly, a monitoring point at an AS towards the bottom of AS topology may not see any of the updates caused for example by a session reset between two tier-1 ISPs. Such a session reset may not change any of the best routes at this AS. On the other hand a monitoring point at a tier-1 ISP may not see any updates caused by a peering link failure between two of its customers. The redundancy requirement inherent in peering should guarantee this. Therefore Teixeira et al. [35] argue that the currently available public data is not sufficient.

2.2 Router configuration and RPSL

The last section discusses how routes are originated and propagated, and how BGP attributes are modified as they are propagated (which, in turn, affects route selection). This section now presents a very brief overview of how to instantiate this inside a network. The first part of this section looks at low level router configuration mechanisms, while the second part summerizes the Routing Policy Specification Language (RPSL) [50].

Todays routers have a large number of configuration options that control the operation of the main processor, the various interfaces / link technologies, available hardware resources, etc.. For example the configuration determines which routing protocols are enabled, as well as the selection of parameters (e.g., OSPF/ISIS weights) and policies (e.g., ingress and egress filters for each BGP session). This information is configured and stored on a device-by-device basis in a distributed manner, which means on each individual router in the network. The configuration can be altered by applying commands to the router via its operating system, e.g., Cisco IOS [51], JunOS [52,53]. These commands can be specified via a Command Line Interface (CLI) or by uploading a configuration file to the router (e.g., via tftp).

Figure 2.4 shows a shortened example of a configuration of two Cisco routers, while Figure 2.6 (see page 18) shows the interface configuration for a Juniper router. For example, the right column of Figure 2.4 shows the configuration of a Cisco GSR 12008 [54] router. Its hostname is *c12008* ("`hostname c12008`"). Various global settings that apply to the router are configured, in the example DNS lookups are disabled ("`no ip domain-lookup`"), and that classless inter-domain routing is performed ("`ip classless`"), etc.. Other global variables which are not specified in the configuration have default values.

In addition to global settings, the configuration includes details about interface specific parameters (such as IP addresses). Note that the GSR in our example is equipped with four Gigabit SX and 8 Fast Ethernet interfaces, but we show only a stripped-down configuration with two interfaces (`FastEthernet1/2` and `GigabitEthernet5/0`).

The "`router bgp 65001`" entry identifies the AS number (65001) and may include a list of commands that enable/disable certain features. This includes the "`neighbor`" command which is used to configure BGP sessions. The example in Figure 2.4 shows one eBGP[13] session configured to a router with the IP address `10.5.1.2` (see "`neighbor 10.5.1.2`

[13] The example shows an eBGP configuration because the remote-as number (65000) is different from the routers ASN (65001). Note that if both numbers match, then an iBGP session is configured.

```
version 12.2
!
hostname c7507
!
ip subnet-zero
ip tftp source-interface GigabitEthernet4/0/0
no ip domain-lookup
!
!
interface FastEthernet1/1/0
 description c7507 FE1/1/0 to c12008 FE1/2
 ip address 10.5.1.2 255.255.255.0
 no ip mroute-cache
 full-duplex
!
interface GigabitEthernet4/0/0
 description c7507 GE4/0/0 to rt 1/1
 ip address 10.2.1.1 255.255.255.0
 no ip mroute-cache
 no negotiation auto
!
!
router bgp 65000
 no synchronization
 bgp log-neighbor-changes
 neighbor 10.5.1.1 remote-as 65001
 no auto-summary
!
ip classless
ip route 10.4.2.0 255.255.255.0 10.5.1.1
!
!
!
snmp-server community public RO
snmp-server enable traps tty
!
!
!
line con 0
 exec-timeout 0 0
line aux 0
line vty 0 4
 password ocsic
 login
!
end
```

```
version 12.0
!
hostname c12008
!
ip subnet-zero
ip tftp source-interface GigabitEthernet5/0
no ip domain-lookup
!
!
interface FastEthernet1/2
 description c12008 FE1/2 to c7507 FE1/1/0
 ip address 10.5.1.1 255.255.255.0
 no ip directed-broadcast
 duplex full
!
interface GigabitEthernet5/0
 description c12008 GE5/0 to rt 1/C
 ip address 10.2.3.1 255.255.255.0
 no ip directed-broadcast
 no negotiation auto
!
!
router bgp 65001
 no synchronization
 bgp log-neighbor-changes
 neighbor 10.5.1.2 remote-as 65000
 no auto-summary
!
ip classless
ip route 10.3.2.0 255.255.255.0 10.5.1.2
!
!
!
snmp-server community public RO
snmp-server enable traps rf
!
!
!
line con 0
line aux 0
line vty 0 4
 exec-timeout 0 0
 password ocsic
 login
!
end
```

Figure 2.4: Sample router configuration for router `c7507` and `c12008`.

```
BGP router identifier 10.3.0.1, local AS number 65000
BGP table version is 1, main routing table version 1

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.5.1.1        4 65001  336571  494703  5213436    0    0 24w6d      163201
```

Figure 2.5: Output of `show ip bgp summary` command.

`remote-as 65000`"). Of course the BGP session setup has to match the configuration on the remote router. To illustrate this Figure 2.4 shows in the left column the simplified configuration of a Cisco 7507. The BGP configuration part reads "`router bgp 65000`" indicating the ASN of the other router that has to match the remote-as command. Then the line "`neighbor 10.5.1.1 remote-as 65001`" sets up the other end of the BGP configuration. The CLI can be used for debugging and status reports. For example Figure 2.5 shows the output of the command "`show ip bgp summary`" which was executed on the c7507 router.

In this thesis we are in particular interested in the eBGP parts of the router configuration. Route manipulations are specified via route-maps (Cisco terminology) and can filter a prefix or change each of the attributes of the routing information. The route-maps may specify destination network checks via access lists, AS path checks via regular expressions matching the AS path string, communities via community lists, etc.. They can be applied to incoming routing updates and to outgoing routing updates. Figure 3.9, and 3.10 on page 47 provide more detailed examples of policy configurations on Cisco routers. Each entry consists of a filter (e.g.,`prefix-list martians`) and a set of attribute manipulations (e.g.,`set community 1:1 additive`). If a router receives a route over a BGP session with an associated import route-map it processes the route-map entries one by one until one of the filters matches the current route. As long as the route-map statement does not contain the keyword "continue" the attribute manipulations are applied to the route and the route is either accepted via the keyword `permit` or denied via the keyword `deny`. If the route-map entry contains the `continue` keyword (see Figure 3.9) with value `300` attribute manipulation are stored in a todo list and processing continues with the route-map entry 300. Juniper offers a similar mechanism by configuring `policy-options` and accessing further policy-statements via the keyword `next policy` (not shown in Figure 2.6).

```
version 6.2R3.10;
system {
    host-name TUM-OM1;
    login {
        user olafm {
            full-name "O. Maennel";
            uid 2000;
            class super-user;
            authentication {
                encrypted-password "...";
            }
       }
    }
    services {
      ssh;
      telnet;
    }
    syslog {
        user * {
            any emergency;
        }
        file messages {
            any notice;
            authorization info;
        }
    }
}
interfaces {
    fxp0 {
        unit 0 {
            family inet {
                address 10.1.1.1/24;
            }
        }
    }
}
```

Figure 2.6: Simple Juniper configuration

Nearly each vendor has developed one or more configuration languages for its routers. As this is not very desirable there are efforts from the IETF to standardize routing policies in a vendor independent configuration language. RPSL has been designed as such a vendor-independent language that is able to capture AS and prefix information. RPSL can also denote BGP

attribute modifications. The IRRToolSet [55] is one software implementation that is able to process RPSL. For example RtConfig [56] translates RPSL expressions to vendor-specific router code. In the near future such tools should not be necessary anymore, vendors should be able to build their low-level router configuration based on RPSL.

In RPSL information is organized in objects, which are intended to communicate policy information among ISPs. This is done via the Internet Routing Registries [57], such as the RADB [58]. The IRRs provide an easy way for consistent and up-to-date configuration of filters. Furthermore, such a database facilitates troubleshooting failures, because prefixes are registered along with contract information. For example if a customer reports that a certain prefix p is not reachable and the prefix is not registered, then it takes a lot longer to figure out who to call to resolve the problem.

Yet there is a lot more potential in the IRR [59]. Unfortunately, most of a policy is stored as text-comments in a database. While this still helps humans in debugging routing problems, it becomes problematic for automated tools to validate routing policies, e.g., [60, 61]. In the following we summerize some of the features of RPSL [50, 62, 63] as far as needed in this thesis.

RPSL is structured in objects, which can be maintained by network administrators. To ensure the authenticity of updates to the database, every object contains a maintained-by attribute pointing towards a *maintainer object*. This contains a reference to an administrative and/or technical contact as well as a specification of an authentication method to ensure that the updates are coming from an authorized maintainer. Updating the database can be done via e-mail and the authentication might be a clear text password or a PGP signed message [63]. The verification is performed automatically by the database.

In RPSL there are 12 different classes of records, that either describe portion of a policy, or describe who is administering this policy. Objects are instantiated classes, which consist of several attribute/value pairs. An attribute may appear once (single valued) or multiple times (multi valued), depending on the specification. Values have a defined type and can be checked during processing, e.g., when checking into a database. One class, which realizes one of the core functionalities of RPSL is the *route object*. It contains a prefix and the origin AS of this prefix. This allows to lookup all prefixes associated with an AS. With this it is possible to generated prefix filters based on neighboring ASes, without human interactions between the two ASes. Such filters can minimizes the error of address space that is not "owned" by an AS.

Prefixes can be grouped using the *route set objects*. By convention route set names must start with "rs-" (e.g., valid route set names might be RS-FOO or RS-BAR). Objects referencing other objects can be created. Beside route sets, which contains routes, there are AS sets ("as-"), which contains ASes, peering sets ("prng-"), which pack several BGP sessions into one object, filter sets ("fltr-"), which contain prefix, AS path or community filters, etc..

The delimiter ":" is used to construct a hierarchical namespace. Each level of the hierarchy may have a new maintainer, which is responsible for all objects lesser levels. Each level-name either consists of a route set name or a AS number. At least one route set name must be present in the hierarchy to indicate the type (e.g., AS1:AS-FOO, AS1:AS2:RS-CUSTOMERS,...).

Each AS has one unique object within the IRR [57], which is called the *aut-num* object. It is supposed to describe the routing policy of an AS. This is realized via two attributes: *import*

and *export*. Both can appear multiple times. To determine whether an update is accepted or distributed, the applicable statements are searched in the order they are specified for a matching rule. An simple example is shown in Figure 2.7 and states that AS1 has one or more (direct) connections to AS2 and accepts the address space `131.159.0.0/16`. No further restrictions apply.

```
aut-num: AS1
import: from AS2 action pref = 1; accept { 131.159.0.0/16 };
```

Figure 2.7: Example of the aut-num object.

```
import: from <peering> [action <action>] accept <filter>;
```

Figure 2.8: Definition of the aut-num import attribute.

Figure 2.8 shows the generalized format of such import statements. Note that the export statement has a similar structure. Here `<peering>` specifies BGP sessions for which the statement (called factor) applies. This can apply to a neighboring AS as whole, or only to a subset of the sessions (specified via AS SET (AS-) or a peering set (PRNG-)). `<filter>` specifies which prefixes can be accepted by an AS. Beside prefix filters this includes AS path restrictions as well as community filters. It can be stated explicitly or via a filter set (FLTR-) reference. The `<action>` is optional and specifies in which way the update is manipulated if it passes the filter. Actions can change any attribute of the update except the prefix. Common actions are addition and/or removal of communities or changes to the local preference, etc..

RPSL is designed in such a way that it should be possible to specify any policy using multiple import statements. Yet, this is inconvenient, if one considers the following policy example: three customers are connected to an ISP (AS1). Prefix filters should be applied based on the address space owned by each customers respectively. Furthermore, an AS path prepending service should be offered to all three customers, which is triggered by a community (e.g., "do not prepend the AS path" = no community present; "prepend AS1 one time" = community $1:1$ is set; "prepend AS1 two times" = community $1:2$ is present). To specify such a policy, nine import statements are needed – three for each customer to filter the routes and to enable AS path prepending. To ease the configuration structured policies have been introduced, which enable on to handle different tasks independently. So it is possible to compose several factors into a term. Figure 2.9 shows a generalized term that handles the AS path prepending[14]. Such a generalized term can be combined via the "refine" statement with other terms, e.g., to the above mentioned route-filter, see Figure 2.10. Such a term is processed in the following way: each term (refine-element) is searched for the first match. If any of the terms has no matching factor, the update will be denied. Otherwise, all actions of the matching factors are collected and applied to the update.

Unfortunately there are some limitations within RPSL. One is that community wild-cards (such as regular expressions) are not supported (by in the current version of RPSL). Furthermore, not all router configuration statements can be mapped to RPSL expressions. For example:

- the maximal number of prefixes from neighbor to accept
- an MD5-based BGP authentication

[14]Note that AS-ANY is a representation for all neighbors; ANY is a filter accepting anything

```
{
    from AS-ANY action aspath.prepend(AS1, AS1);
        community.contains(1:2);
    from AS-ANY action aspath.prepend(AS1);
        community.contains(1:1);
    from AS-ANY accept ANY;
}
```

Figure 2.9: Generalized RPSL term for AS path prepending.

```
import: {
            from AS-ANY action aspath.prepend(AS1, AS1);
                community.contains(1:2);
            from AS-ANY action aspath.prepend(AS1);
                community.contains(1:1);
            from AS-ANY accept ANY;
        }
        refine
        {
            from AS2 accept AS2;
            from AS3 accept AS3;
            from AS4 accept AS4;
        }
```

Figure 2.10: Example of structured policy in RPSL.

- e-bgp-multihop
- default routes

However, RPSLs features have proven to be extremely useful in terms of debugging, documentation, and legibility. Hopefully router vendors will soon allow operators to configure their routers via RPSL. Already some tools exists (e.g., IRRToolSet) that allows operators to check, manipulate and transformate RPSL.

2.3 State of the art and related work

This section discuss related work in the context of (i) configuration management, (ii) BGP dynamics, and (iii) router testing.

2.3.1 Configuration management

Work towards systems that help ISPs manage and structure their policies exists in a manifold diversity. Tools that are able to manage routers or switches in an automated or semi-automated fashion range from free software scripts, like RANCID [64], to commercial products from companies such as AlterPoint, Cariden, CPlane, Gold Wire Technology, Intelliden, Network-Clarity, OpNet, Rendition Networks, Tripwire, Voyence, Wandl, etc.. This set also includes products from router vendors themselves, e.g., Cisco IP Solution Center [65]. Such tools can help with keeping track of router updates, maintain consistency across similar devices, and document critical changes automatically. The goals of such utilities are to help

administrators to reduce manual configuration and provisioning tasks, and this is claimed to lead to a minimization of human errors, which in turn reduces potential downtime. Furthermore, they provide more security through tight access controls and configuration audits. Some even include mechanisms to optimize packet traffic flows. While most products have the capability to manage multiple devices in the network at the same time, most of them focus on direct device management, e.g., the routers. They allow, e.g., setting up MPLS, VPNs, or downloading previously stored configurations. Note that most of these products are targeted towards enterprise solutions, and only some of them are developed for the needs of Internet service providers. This is due to the fact that policies of ISPs are often complex and do not fit any pre-defined specifications. Therefore most ISP rely on engineers for maintaining and realizing their routing policy. To the best of our knowledge, this task is often accomplished by using a combination of small utilities, including homegrown scripts, templates, and human experience.

Recent work by Caldwell et al. [18] shows the problems inhering automated AS-wide policy configuration and demonstrate possible directions. Gottlieb et al. [66] describe an automated provisioning system for BGP. How to migrate, store and analyze router configuration is presented in [18, 67]. Other work, such as [68], shows the complexity of creating automated configuration tools. For example in [68], the authors illustrate an automated firewall configuration tool.

A good overview about how network operators are using BGP to realize various policies in their network (such as traffic engineering) can be found in Uhlig et al. [20]. Yet the implications of certain routing policies on the control and data plane can be difficult to predict [69, 70]. To gain a better understanding about the control plane interactions within the Internet, some projects monitor the routing system, e.g., [59], and try to validate the routing policies in use, e.g., [17, 60, 71–73], while others try to understand their interactions via simulation, e.g., [74].

Griffin et al. have shown that policy interactions between ASes can cause BGP divergence at an Internet-wide level [7,75,76]. Even assuming the unrealistic case that all ISPs register their policies in the IRRs, the complexity for validating the Internet routing system convergence properties is NP-hard [16, 23, 41]. This indicates that the research community needs a solid understanding about BGP dynamics occurring in the Internet.

2.3.2 BGP dynamics

In his work on developing a signal propagation model for BGP updates, T. Griffin [12] observed that “In practice, BGP updates are perplexing and interpretation is very difficult”. And indeed, BGP has been studied extensively within the last few years, e.g., [77–83], but is still an active research area. In general it is difficult to infer root causes of observed BGP updates given the opaque routing policies and the limited visibility into the topology [35].

Unexplainable large convergence times have been reported [84] and it is unclear whether they are attributable to protocol interactions [13–15], implementation idiosyncrasies [85], hardware limitations or other factors [19].

Researchers address these problems by characterizing the routing table [86], analyzing unusual events [87], and/or study unexpected behavior [88]. For example, Geoff Huston looks at

general statistics of the routing tables and is even able to derive valuable insights into Internet evolution [89] (i.e., the "dot-com-boom" showed significant different characteristics of how people used address space allocations and the way they announced more specific prefixes for doing traffic engineering).

Typically such studies about BGP dynamics analyze either particular destination prefixes to understand the trend of BGP update growth over time [45,90], or particular time periods, e.g., during worm outbreak, to understand how BGP behaved while Internet is under stress [91].

Caesar et al. [92] as well as Chang et al. [93] use three dimensions to infer the origins of routing instabilities: *time, views,* and *prefixes*. Yet Caesar et al. first distinguish quiescent and turbulent periods while Heidemann et al. perform the per view and prefix steps in one clustering step. Lad et al. [94] use an idealized model which assumes shortest path routing. Finally Wu et al. [95] designs a system that identifies a few dozen significant routing disruptions within millions of BGP update messages.

Still a lot of questions cannot be answered by just observing update propagation. Therefore some researchers actively inject faults into the network (e.g., BGP Beacons [96]). For example Labovitz et al. [97–100] studied reachability, packet loss rate, and delay to a set of web hosts while the stub networks experienced routing changes. Follow-up studies by Mao et al. [101], Bush et al. [8], and Bürkle [102] investigated similar questions with more beacons and more observation points.

To understand BGP dynamics, researchers need a solid knowledge about the Internet topology (which includes ISPs policies, AS relationships, etc.). Some projects start with large Internet mapping efforts, for example by using traceroute to discover the network topology [103, 104]. Others try to infer the routing policies without access to the router configurations, e.g., [105,106]. Since Gao's [31] AS relationship inference methodology has been published, researchers try to get a rule of thumb who is a customer of which provider and who peers with whom [107,108]. But commercial relationships can be complex, and the decisions chosen by BGP do not have to match the optimal available paths in the Internet [109, 110].

2.3.3 Router testing

Besides those dynamics that can be studied from active and passive BGP research (which uses the Internet itself as test environment), there are also efforts from vendors, ISPs, the IETF and researchers to investigate protocol behavior before the deployment in the operational network. Those studies are conducted in test-beds that consist of a small number of routers and test-traffic generators, e.g., Router Testers from Agilent [111], IXIA [112], Spirent Communications [113] or Arsin [114].[15]

Furthermore, the work of the IETF, in particular the Benchmarking Methodology Working Group (bmwg) [22], is relevant for testing research. They recommend metrics and test setups for test-beds. This includes for example a terminology to standardize how to measure eBGP convergence in the control plane of a single BGP router [115, 116].

[15] Note that those products do not generate IP or routing traffic that is necessarily consistent with the traffic in the Internet. For example test-traffic often does not reflect temporal variability as captured by self-similarity nor does it capture the full range of IP addresses.

Studies at the router level explain the behavior of protocols at the network scale. Related work such as [101, 117, 118] examines the details of the implementation variants of different vendors. Other work looks at how routers react to large routing tables loads [119]. We address related issues in Chapter 6.

3 AS-Wide Inter-Domain Routing Policies

A network's *routing policy* is embodied in the configuration of the routing protocols of each individual router and link in the network. These configurations are the basis on which the Internet routing protocols create the "network-wide intelligence" that transforms individual network components into an operational IP network [120]. Initially the goal of a routing policy was to just ensure connectivity. But nowadays the requirements and therefore the complexity has grown substantially and include maintaining contractual or business relationships between different domains, ensuring resilience requirements even under overload, achieving stability, guaranteeing efficient internal operations, and supporting growing customer demands to control their traffic flows.

Sophisticated customers of Internet transit services demand more control over how their routes are propagated past their immediate transit providers (with the goal to control the paths of the traffic attracted by the routes [121]). Community-signaled services address this problem, and more recently they have become an essential part of transit providers service offerings. The NOPEER community [122] gives a simple and useful example that has additional appeal by virtue of defining Internet-wide semantics. An overview of more community-based hacks for traffic engineering can be found in [123].

Even though the demands on the routing policy have grown, the way how a routing policy is expressed and realized has not changed. It is still embedded in the low level router configuration commands of the individual network components of the network. This means that an AS-wide entity, the routing policy, is not managed AS-wide but rather component-by-component. Therefore artifacts within the routing policy can lead to routing stability problems, e.g., [7, 124]. Even worse, component-by-component realization often involves manual configuration which is error-prone and expensive [18, 19, 66, 67, 72]. In the worst case a simple omission on one router can invalidate the overall policy[1].

Already the problem of formulating the AS-wide routing policy of an ISP is non-trivial. The IRR, which facilitates universal access to routing policies and is part of an architecture for coordinating Internet routing policies [128], has not yet fulfilled its potential [59]. Either the ISP does not even try to formulate its policy in the proposed language, RPSL, or the policy is not consistent with the routing policy actually realized. Yet, the growing complexity of policies as well as the growing customer demands in addition to the intrinsic complexities of BGP itself accentuate the need for a system that allows an abstract formulation of the AS-wide routing policy of an ISP as well as the automatic configuration of the network components.

[1] Discussions at operator forums, such as NANOG [125], RIPE [126], APRICOT [127], shows that running a large ISP is not only done by optimizing selfish interests but also involves participating in a community composed of competitors. Mailing lists document nicely the daily struggle of operators to troubleshoot invalid policies and their effects.

In this chapter we propose a system for formulating and managing the routing policy as a first class entity. Using the routing policy as its basis, our system generates the necessary configuration *configlets* for all routers in a specific network to realize the routing policy as well as a documentation of the routing policy. In effect we raise the management level for the routing policy from the management of individual network components to managing the routing policy itself. A version of our system is in production use at a Deutsche Telekom (`AS 3320`), an ISP with more than thousand routers.

The remainder of this chapter is organized as follows: in Section 3.1 we explore the concept of a routing policy and then explain, in Section 3.2, our system design in more detail. Section 3.3 describes the data models. Section 3.4 discusses how to construct a database-driven provisioning system, the *configurator*, based on these data models. This chapter concludes with Section 3.5 which shows some generated *configlets* for the examples presented in Section 3.3 and discusses our experiences with the system. Section 3.6 summerizes our contributions.

3.1 Network-wide routing policy

Before we can present our system for defining and realizing an *AS-wide routing policy* we need a better understanding of the underlying concepts of a routing policy.

A routing policy should capture the essences of the many rules according to which ASes interact with each other. The multitude of such rules include rules for realizing best common practice (such as martians filter, prefix aggregation), rules for internal routing (including filtering of internal address space, traffic engineering), rules for maintaining business relationships (such as a peering policy), and rules for offering value-added services to customers (such as community-signaled services). As a result, a BGP session is governed by some combination of these rules, which together determine which routes are accepted on ingress or propagated on egress. The building blocks for an AS-wide routing policy are policies and services. Each policy or service corresponds to a rule like the ones discussed above. The difference is that a service is an optional value-added feature, that can be booked for any neighbor. Policies are applicable AS-wide (within the AS). One aspect to consider is that rules change over time, that new rules are added while others are removed, and that even policies may become services or vice versa.

Almost none of the above rules refer to specific BGP neighbors or specific network components. Indeed, the rules can be formulated independently of the specifics of any network. Just the selection of BGP sessions, to which a rule has to be applied, depends on certain parameters of the BGP session, the BGP neighbor, or the services that are booked for the BGP session or the BGP neighbor. In fact, certain rules cannot be specified on a session-by-session basis. Rather they require that some actions are taken for all routes entering a network and for all routes leaving the network. But depending on the session over which they enter the AS and the session over which they leave the AS the actions have to differ. For example, a peering policy [129] may mark all routes on ingress with a community that reflects the peering type of the BGP session and on egress it will filter the routes based on the community tags and the peering type of the session. Here the peering type, a parameter of the session, determines the actions on the routes. Other rules, such as a community-signaled black-hole service [130], require different actions to be taken for different sets of sessions. For those sessions that have

subscribed to the service, the rule checks if an appropriate black-hole community is set for a route. If it is, traffic to that prefix is redirected to a discard interface. If a session has not subscribed to the service, the rule does not allow the BGP neighbor to send routes tagged with the black-hole community. These are rejected.

Relying on the above example we identify the following strategies for defining an AS-wide routing policy:

- It consists of a collection of *policies* and *services*.
- Policies apply AS-wide.
- Services are bookable for each session or each neighbor.
- Policies and services perform *route manipulations* for *subsets* of BGP sessions. (A route manipulation consists of applying a filter to the incoming or outgoing routes and manipulating their attributes.)
- Subsets of BGP sessions are identified via *conditions* on BGP session parameters or their services.
- A policy/service may perform route manipulations on *multiple* subsets of BGP sessions.

Accordingly, any system that supports the definition of an AS-wide routing policy needs primitives for defining policies and services as well as specifying sets of session and route manipulations. There is no need for these primitives to be network-specific. Indeed, in principle it should be possible to use a policy or a service defined by one ISP within another one.

3.2 System design

In this chapter we propose a simple and efficient way for defining AS-wide routing policies and realizing them. In effect we are bridging the gap between the concepts underlying AS-wide routing policies and how routing policies have to be configured on a session-by-session basis. In order to realize a routing policy, we need additional information besides a policy specification: a description of the network and a library with BGP operations to be used as building blocks. All three entities are realized as databases. These databases are the input to a *configurator* that generates eBGP configuration *configlets* for all routers (all BGP sessions) in the network. In addition, the *configurator* generates a policy description in RPSL, which, if desired, can be published at the IRR and may in the future be useful for generating *configlets*. An overview of the system is shown in Figure 3.1.

In this section we briefly sketch the abstractions we have chosen for the inputs to our tool, the *configurator*, before discussing why these are sufficient for satisfying the concepts of an AS-wide routing policy. Then we explain how the *configurator* generates the desired output, *configlets* for all eBGP sessions. We end this section with a discussion of the advantages that our system offers.

3.2.1 Concepts underlying the *configurator* input

The decomposition of the inputs to the *configurator* into three logical entities, routing policy specification, network specification, and library with BGP operations, imposes a clear sep-

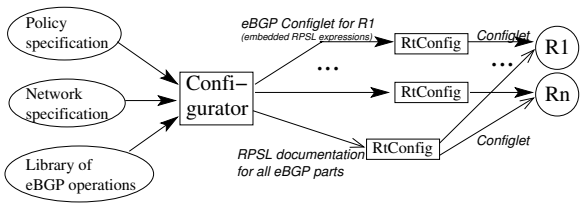


Figure 3.1: Overview of proposed system.

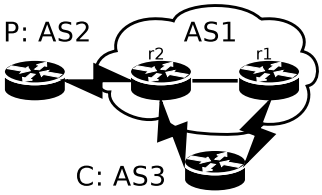


Figure 3.2: Example network

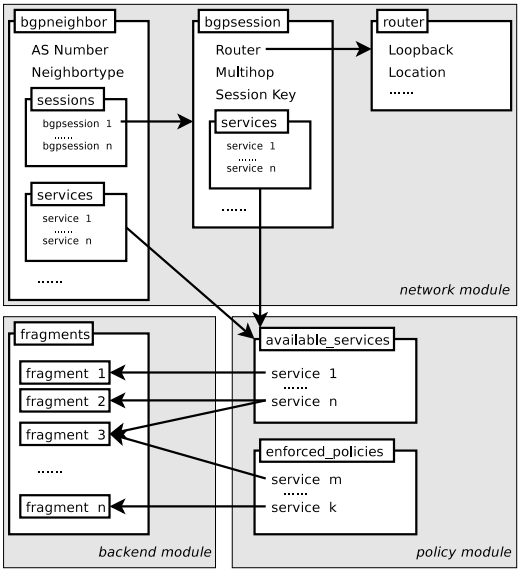


Figure 3.3: Main database objects and their relationships.

aration of tasks and is aligned with the organizational boundaries within an ISP. The policy designer should focus his/her attention on the set of policies and services (e.g., based on what criteria tagging should be done, where to filter or manipulate attributes). The network administrator, who adds a new BGP session, should only have to worry about the parameters of the specific session and which services are to be booked (e.g., add routers to the network, setup customers, active services in the network as booked by customers). He/she does not have to worry about ensuring that the configuration of this session respects all policies. The person realizing and testing some BGP operation has no need to know which BGP sessions use this specific operation (e.g., testing that the configuration works on the routers that are in use by the ISP before the configuration gets uploaded to routers in production use; this includes evaulating new features).

The overall AS-wide routing policy is stored in the database of the *policy module* and consists of a set of policies and services. The main difference between a service and a policy is that services are bookable and therefore require interactions with the *network module*, while policies apply AS-wide. Accordingly, we refer to services as *available services* and to policies as *enforced policies*. As they rely on the same concepts they are both specified as *service*. Each service is expressed as one or more steps of first selecting a set of BGP sessions based on some *conditions* on the *network module* database and then applying one or more BGP operations from the *back-end module* database to these sessions. Each of these combinations is referred to as a *sessionset*. Note, that this process results in two selection steps. The first step, the condition in the *sessionset*, selects a set of BGP sessions. The second step, the BGP operation, selects a set of routes (via a filter) for that session. Both levels of selection are crucial.

The network specification is stored in a database within the *network module*. It contains a description of the network including routers, BGP neighbors, and BGP sessions with their parameters and list of booked services. We include BGP neighbors (another AS), as most re-

lationships are realized via multiple BGP sessions between the ASes, that either use the same or similar services. To simplify the specification of the network inheritance of parameters is supported. The BGP session inherits (unless overwritten) its router-specific elements from the router on which it is configured and its other elements from the BGP neighbor. Services are parameters of BGP neighbors and sessions. The only difference is that parameters are singular, while services are not. A network operator might want to use different service parameter values for different subsets of routes in order to, e.g., tune traffic flows through its network. We support this via the concept of *cases*.

The *back-end module* database contains a library of *fragments*. Each *fragment* pairs a name with a specific BGP operation. A BGP operation is specific to a single BGP session. A BGP operation consists of selecting a group of routes, which are to be accepted or denied, and performing specific BGP attribute manipulations on the selected routes. Route selection is done via *filters* based on lists of prefixes, lists of communities, regular expressions over AS paths, etc.. Attribute manipulations involve some *actions* on route attributes, e.g., setting the local preference. These are very low-level operations, whose syntax depends on the configuration language supported by the router on which the session is realized. Since, for the purpose of specifying a routing policy, this syntax is irrelevant, the functionality is encapsulated in a library object which can be referred to by a name. *Fragments* are the building blocks for our assembly system, the services and policies.

Besides encapsulating vendor-specific realizations each *fragment* also includes a non-router vendor-specific one to enable us to generate a realization of the routing policy via an intermediate language, a necessity if the number of router vendors that have to be supported increases. RPSL, an object-oriented language designed for publishing routing policies, is the prime candidate for such an intermediate language. RPSL supports most of the features provided by the vendor-specific languages, has excellent support for filter handling, delegation of responsibility, and the ability to operate on sets of routes, sets of BGP sessions, and sets of ASes. As filter handling and delegation are hard problems that are well handled by RPSL, we allow the *fragment* writers to include RPSL filter expressions inside the vendor-specific parts of the *fragments*. These are resolved with the help of RtConfig [56], which is part of the IRRToolSet [55]. RtConfig converts RPSL expressions to vendor-specific router configurations. Since RtConfig can interact directly with the IRR [57], it is possible to retrieve information about neighbors and their sets of routes. This allows us to delegate the responsibility of specifying, e.g., a customer routeset, to the customer itself. This has the benefit that, if the customer changes its routeset, he can handle the change him-/herself [63]. No human interaction with the ISP is necessary to update the filters, a rather convenient feature.

An overview of the main objects in the three databases is shown in Figure 3.3. This figure also includes the various relationships between the modules:

- The *fragments* of the *back-end module* need information about the specific router and the specific BGP session in order to be converted into a *configlet*. We solve this by writing *fragments* as templates. They can use variables to refer to network-specific elements and a functional interface to access other data sources.
- As soon as a service is specified as an *available service*, it is made bookable as a BGP session parameter. This enables the network operator to book the service for any session. Note, that each service specification defines an appropriate language for specifying the now feasible network configuration. *Enforced services* are not available

as BGP session parameters!
- Conditions within the *policy module* are based on the session parameters from the *network module*. This is again solved via variables for referring to network-specific elements.
- Services are defined by applying *fragments* to subsets of BGP sessions. Such references are possible by simply using the name of the *fragment*.

3.2.2 Enabling abstract routing policy specification

Next we discuss how our system is able to address the concepts identified for defining a routing policy.
"Policies and services perform route manipulations": For us a route manipulation corresponds to selecting a group of routes, which are to be accepted or denied, and performing specific BGP attribute manipulations on the selected routes. Accordingly, each route manipulation is a BGP operation. The policy designer can use *fragments* to specify what kind of route manipulations to perform as long as the operation of the *fragment* is well-specified and documented.

"Subsets of BGP sessions are identified via conditions on BGP session parameters": While policies can be specified in an abstract manner, they eventually have to be combined with the network in which the routing policy is realized. At that point the policies have to be applied to specific sessions. The choice of sessions depends on the session parameters and the booked services. This network-specific information is accessible to the policy designer via variables referencing the network elements. Identification or rather selection of BGP sessions is therefore possible via *conditions* on *any* of the BGP session parameters.

"Subsets of BGP sessions are identified via conditions on their services": Once an *service* is specified as *available service*, it is accessible as BGP session parameter. There is no fundamental difference in identifying or selecting sessions based on booked services or based on other session parameters. The condition can include both.

"Policies and services perform route manipulations on subsets of BGP sessions": Given that *fragments* are organized in the *back-end module* database and network elements are organized in the *network module* database, policies or services can be defined by multiple selections and one join operation. Identifying BGP sessions via conditions is a "select operation" on the network database, determining the appropriate route manipulations is a "select operation" on the *back-end module* database. Combining them is a "join". This is expressible via *sessionset*.

"A policy/service may perform route manipulations on multiple subsets of BGP sessions": This corresponds to multiple select and join operations, meaning multiple *sessionsets*.

"Policies apply AS-wide; services are bookable for each session or each neighbor": Available services are bookable by the network administrator via the *network module*, *enforced policies* are not selectable. Their route manipulations just depend on the *conditions* that they use for selecting BGP sessions.

This completes our discussion about how our proposed system allows us to address the concepts for defining routing policies. Lets consider how to realize the *configurator*.

3.2.3 Design alternatives for the *configurator*

The next problem to tackle is how to realize a routing policy. Realizing a routing policy corresponds to generating appropriate vendor-specific *configlets* for all routers in the network using the data of the three modules: *policy,* network*, and* back-end as input. This is the task of the *configurator*. We choose to generate *configlets* with embedded RPSL filter statements as well as pure RPSL. The latter currently serves documentational purposes. But as the capabilities of RPSL, RtConfig, and the vendor-specific router configuration languages evolve, and our experience with the system grows, this might become the basis for automatically generating the *configlets*.

The motivation for using an hybrid approach is that it lets us benefit from the advantages of generating both, while avoiding some of their disadvantages. The benefits of generating the *configlets* directly are precise control over how policies are realized as well as the chance to optimize the generated code. In addition, the tool can ensure that any filter IDs are used consistently across different instantiations as well as across different routers (very useful for debugging). This eases the transition from a manual configuration or template-based process to our system in the sense that it is often possible to write policies that resemble the existing configurations. Once all configurations are based on the *configlets* generated by the system, it is possible to add new features by enhancing the routing policy. If there is a problem, it is clearly within the *configurator* or any of its input databases. There are no dependencies on other tools or developers. Indeed, since each *fragment* has a very precise and simple functionality, they are easy to write and easy to test.

Currently our tool supports two router vendors: Cisco and Juniper. If the number of router vendors increases, an intermediate language such as RPSL is not only useful but a necessity. Still at this point RPSL does not suffice. In addition, RtConfig is a complex tool grown over time that is fairly intolerant to mistakes in its input data, which is aggravated by a somewhat sparse documentation. Another issue with RPSL and RtConfig is that they do not yet support all features. Also adding support for new vendor features is not trivial and usually comes with some time delay. These problems are circumvented by our hybrid approach.

The *configurator* proceeds by processing the services one by one. For each service, listed under available service or enforced policy, it considers all *sessionsets* within their service specifications. For each *sessionset* the sessions for which the condition evaluates to true are selected. For each of these relevant sessions and all *fragments* within the *sessionset* it adds the *fragment* together with the necessary parameters to a fragment_list at the session. Once all services have been processed, the *configurator* generates the *configlets* from the fragment_lists, one session at a time, by merging the appropriate vendor-specific realizations of the *fragment* functionality into one *configlet*. The generation of the RPSL documentation proceeds in a different order to take advantage of RPSL's capability of grouping sessions.

The functionality within our system goes beyond RPSL's concept of a policy. For RPSL a policy is a BGP operation on a set of routes. For us a routing policy is collection of policies and services that are each specified as first class entities. As a result if one adds a session to the network in our system one has an explicit choice of which services to enable while all policies are automatically applied. With RPSL one has to explicitly add the session to all sets for all policies that need to be applied to this session. Still, our system is not a superset of RPSL, rather it can be realized via RPSL.

3.2.4 Advantages of the approach

Lets consider the properties that our proposed system offers for defining an AS-wide routing policy.

Abstraction: Our system allows us to express a routing policy as a set of enforced policies and available services using high-level language primitives rather than forcing one to use low level BGP filters and attribute manipulations. The specifications are stored in a database which eases their maintenance.

Separability: It is possible to independently express how each policy is realized.

Extensibility: Extensibility goes two ways: First it is possible to add new policies or services and/or to change existing ones. Changing existing or introducing new policies is possible in stages. Second it is possible to take advantage of new router features without having to change the routing policy. One just has to rewrite the appropriate *fragments*.

Customizability: Services rely on *fragments* that can access network-specific parameters and filters. As such the services are easily customizable. In addition, our decision to enable the use of RPSL filters allows us to use an existing database.

Modularity: Our system encourages the policy designer to realize each of the many rules of a routing policy as a service. In addition, the services are in principle independent of each other. Of course if two policies manipulate the same BGP attribute, possible conflicts in the realizations have to be addressed and resolved by the overall system.

Debuggability: By adding comments to the *configlets* the origin of each line in the generated *configlet* it is easily traceable. Each *fragment* can be debugged individually and the system is able to recognize dependencies and force the designer to disambiguate them. How optimized the generated *configlets* are depends on how optimized the *fragments* are. To improve readability and consistency across routers some redundancy can easily be justified as it improves debugability.

Testability: It is possible to automatically generate *configlets* for all common service combinations. These can then be used as test cases in labs or for manual inspection by the routing policy designer.

3.3 Data model

Our proposed system relies on three databases, the *network module*, which captures the state of the network, the *policy module* for specifying the routing policy, and the *back-end module* for providing a library of base elements for formulating services. We choose to represent each database as a set of XML documents. Among the advantages of choosing XML are: the set of tags is flexible and extensible; XML is designed for representing data and is capable of capturing structure even in semi-structured data; a wide range of editors and libraries are available to ease the generation, maintenance, verification, and processing of the modules.

To illustrate the capabilities of our system we use the example shown in Figure 3.2 on page 28. AS 1 uses our system and operates two routers: $r1$ and $r2$. It has two BGP neighbors: P and C. While C (AS 2) is a customer with two BGP sessions ($c1$ to router $r1$ and $c2$ to $r2$), P (AS 3) is a peer with one BGP session (p to $r2$). The routing policy consists of the policies: peering, martians filter, and community filter and the services: black-hole and traffic engineering via MED. The black-hole service is booked for all of C sessions and restricted to an authorized routeset. In addition, C has the ability to use MED based traffic engineering on session $c2$.

3.3.1 Network Module

The task of the *network module* is the description of the components of the network that are necessary for realizing a routing policy. On first sight one would say that these are routers and BGP sessions. We choose to explicitly represent BGP neighbors as their BGP sessions often share parameters and use the same policies. This allows BGP sessions to inherit properties from BGP neighbors. In addition, there are some AS-wide parameters, such as the AS number. Figure 3.4 shows examples of how to describe network elements and the relavant XML tags of the network of discussed above. (The * after an element indicates that it can be used multiple times.)

In general, each `<router>` consists of a set of interfaces, is of a certain hardware type, runs a specific OS version, is at a specific location, meaning in a country and geographic region, etc.. Yet, not all of these elements are relevant for expressing policies. So for example, location matters only if policies tag routes with communities marking the location where the route entered the network. The OS version is needed by the *configurator* to ensure that *configlets* that have been tested for the specific OS versions are generated.

Each `<bgpneighbor>` captures a relationship with a specific BGP neighbor, which is realized by a number of associated BGP sessions. Each BGP neighbor description includes their AS number and some set of associated basic BGP configuration options, e.g., the maximum number of prefixes that it is allowed to announce. In terms of routing policy it is mandatory/desirable to specify filters to be able to apply incoming and/or outgoing filters. Furthermore, the routing policy has to include a service that ensures that the business relationship with the BGP neighbor is respected by enforcing the peering status: upstream, peer, customer, etc.. We propose to enforce this via a policy rather then offering it as a service. Accordingly, the neighbor type is not a service parameter but an subelement of BGP neighbor. This way it can be used in the condition of the peering policy. Observe, that there is no need to specify on which routers the BGP sessions are realized.

Each `<bgpsession>` has to contain sufficient details to realize the session. This includes information about the endpoints, such as the IP addresses, port numbers, and AS numbers. As one does not want to duplicate information from the router or the BGP neighbor we rely on inheritance. This suffices to access the local endpoint information via a cross-reference to the router. Furthermore, the BGP neighbor contains a cross-reference to all its BGP sessions. Accordingly, access to the remote endpoint information is possible. Cross-relationships are realized by using the value of the `<name>` tag. This enables a BGP session to inherit properties from the router as well as the BGP neighbor, including the service selection. To disable inheritance or to overwrite inherited values the subelements of `<router>` and `<bgpneighbor>` are valid within `<bgpsession>`.

```
<router>
<name> r1 </name>
<loopback>1.0.0.1</loopback>
<location>
  <city> Munich </city>
  <country> DE </country>
  <region> Europe </region>
</location>
<system>
  <hw> GSR </hw>
  <sw> IOS 42.0(12)ST3 </sw>
</system>
</router>
```

(a) Router *r*1

Router name	`<name></name>`
Router IP address (e.g., loopback IP)	`<loopback></loopback>`
Router location	`<location>`
e.g., Region	`<region></region>`
e.g., Country	`<country></country>`
	`</location>`
Router information	`<system>`
e.g., Hardware type	`<hw></hw>`
e.g., Software version	`<sw></sw>`
	`</system>`

(b) Subelements of `<router>`

```
<bgpneighbor>
  <name> Neighbor P </name>
  <neighborAS> 2 </neighborAS>
  <neighbortype>
     peer
  </neighbortype>
  <session> p </session>
  <filter_import>
     AS2:RS-I
  </filter_import>
  <filter_export>
     AS2:RS-E
  </filter_export>
</bgpneighbor>
```

(c) BGP neighbor *P*

Neighbor name	`<name></name>`
Neighbor type (e.g., peer)	`<neighbortype></neighbortype>`
AS number of the neighbor	`<neighborAS></neighborAS>`
Import filter handle	`<filter_import></filter_import>`
Export filter handle	`<filter_export></filter_export>`
BGP session list	`<session></session>*`
Subscribed services	`<services></services>`

(d) Basic subelements of `<bgpneighbor>`

```
<bgpsession>
  <name> p </name>
  <myrouter> r2 </myrouter>
  <remoteIPaddr>
     2.0.0.1
  </remoteIPaddr>
</bgpsession>
```

(e) BGP session *p*

Session name	`<name></name>`
Router name (within this AS)	`<myrouter></myrouter>`
AS number of remote endpoint	`<neighborAS></neighborAS>`
IP address of remote endpoint	`<remoteIPaddr></remoteIPaddr>`
Port number of remote endpoint	`<remoteport></remoteport>`
Subscribed services	`<services></services>`

(f) Basic subelements of `<bgpsession>`

```
<bgpsession>
 <name> c2 </name>
 ...
 <services>
  <egress_med>
   <case>
    <filter> SET-2000 </filter>
    <med_value> 2000 </med_value>
   </case>
   <case>
    <filter> SET-3500 </filter>
    <med_value> 3500 </med_value>
   </case>
   <default>
    <med_value> 100 </med_value>
   </default>
  </egress_med>
 </services>
</bgpsession>
```

(g) BGP session *c*2 for neighbor *C*

Service name	`<`*servicename*`>`
Case differentiation	`<case>`
Filter specification	`<filter></filter>`
Parameter name	`<`*parametername*`></`*parametername*`>*`
	`</case>`
	`<default>`
Parameter name	`<`*parametername*`></`*parametername*`>*`
	`</default>`
	`</`*servicename*`>`

(h) Subelements of `<service>`

Figure 3.4: *Network module* XML elements

Network-wide policies	<enforced_policies>
Service name	<name> </name>*
	</enforced_policies>
Available routing services	<available_services>
Service name	<name> </name>*
	</available_services>
Service definition	<service>
Name	<name> </name>
Parameter list	<parameter> </parameter>*
Session selection criteria	<sessionset>
Ingress or egress	<direction> </direction>
Session selection criteria	<condition> </condition>
Group of manipulations	<task>
Manipulations to perform	<fragment> </fragment>*
	</task>
Alternative manipulation group	<default>
Manipulations to perform	<fragment> </fragment>*
	</default>
	</sessionset>*
	</service>*

Figure 3.5: *Policy module* XML base elements.

While services are defined within the *policy module* they are booked via the *network module*. Accordingly, each <bgpneighbor> and <bgpsession> contains a <services> section. The list of available services together with their parameters are extracted (indicated by the italic font in Figure 3.4 (h)) from the list of services listed under <available_services>.

Still this is not quite sufficient to enable an appropriate specification of the services in the *network module*. For example an ISP might want to use the MED value 100 as its default. But for some subset of its routes it prefers the value 2000 and for yet another subset the value 3500. This example highlights that a case differentiation is needed within the service specification. Each case has the ability to select routes according to some filter and specify a different parameter value. For the user of the system it can be useful to specify a default case which is applicable if none of the other filters match. A default case is just a case entry without filter. Such a service definition can lead to conflicts, e.g., if route A matches the filters in the first and the second case entry. Which parameter value is appropriate? We decided to resolve such conflicts according to the sequence in which the cases are specified. In this case the parameter value from the first case entry is chosen for route A. We stress that inheritance also applies to services: if a BGP neighbor uses a consistent service set with consistent parameters for all BGP sessions it is sufficient to specify them once (at the BGP neighbor). Otherwise they can be redefined, removed, or/and changed individually at each session.

3.3.2 Policies

The *policy module* is at the core of the routing policy description. It allows a policy designer to define an AS-wide routing policy using the concepts identified in Section 3.1. It enables him to distinguish services that are selectable on a per session basis from policies, that have to be applied AS-wide. Accordingly, the top level tags, see Figure 3.5, are <service> for specification, <enforced_policy> and <available_service> to distinguish between services and policies.

```
<enforced_policies>
   <name>  peering  </name>
</enforced_policies>

<availabe_services>
   <name>  blackhole  </name>
</available_services>
```

```
<service>
   <name>  blackhole  </name>
   <parameter>  <blackhole_set/>  </parameter>
   <sessionset>
      <direction>  ingress  </direction>
      <condition>  IF($service.blackhole)  </condition>
      <task>
         <fragment>  ingress_blackhole_community  </fragment>
         <fragment>  ingress_blackhole_accept  </fragment>
      </task>
      <default>
         <fragment>  ingress_blackhole_community_deny  </fragment>
      </default>
   </sessionset>
</service>
```

```
<service>
   <name>  peering  </name>
   <parameter>  $session.neighbortype  </parameter>
   <sessionset>
      <direction>  ingress  </direction>
      <task>
         <fragment>  ingress_mark_neighbortype  </fragment>
      </task>
   </sessionset>
   <sessionset>
      <direction>  egress  </direction>
      <condition>  EQUAL($session.neighbortype, "peer")  </condition>
      <task>
         <fragment>  egress_allow_customer  </fragment>
         <fragment>  egress_allow_self  </fragment>
      </task>
   </sessionset>
   <sessionset>
      <direction>  egress  </direction>
      <condition>  EQUAL($session.neighbortype, "upstream")  </condition>
      <task>
         <fragment>  egress_allow_customer  </fragment>
         <fragment>  egress_allow_self  </fragment>
      </task>
   </sessionset>
   <sessionset>
      <direction>  egress  </direction>
      <condition>  EQUAL($session.neighbortype, "customer")  </condition>
      <task>
         <fragment>  egress_allow_all  </fragment>
      </task>
   </sessionset>
</service>
```

Figure 3.6: *Policy module* XML examples

Recall that a service consists of multiple *sessionsets* each of which uses a condition for selecting sets of BGP session and then applies a set of *fragments* to realize some route manipulations. This structure is reflected in the data model, under `<sessionset>`. It contains, with `<task>`, references to the appropriate *fragments* of the *back-end module* for realizing the route manipulation functionality, and with `<condition>` the ability to specify the desired condition.

Lets review how to select sets of BGP sessions. For a policy the selection depends on some parameters of the session. For a service it matters if the service is booked. In addition, the realization of the service might depend on some parameters of the session. Accordingly, useful selection criteria include, besides the direction of the session, any of the session parameters specified in the *network module*. These are accessible via cross-references using the following (PERL like) syntax: `$element_name.$subelement_name`. All elements of the network are available as such variables for defining services. As the direction of the BGP session, ingress or egress, in which some actions should be applied is orthogonal to the condition for selecting set of session we choose to explictly separate it. Accordingly, `<sessionset>` contains the subelements: `<direction>` and `<condition>` for selection purposes.

The `<condition>` allows the policy designer to select BGP sessions based on equality, on numerical comparisons (`LOWER`, `GREATER`, `EQUAL`), or/and on existence tests (`IF`), etc., on the session parameters, the booked services, and/or the parameters of the booked services. In addition, it is possible to create more complex conditions using the logical operations `AND`, `OR`, `NOT`. It is easily possible to expand this set of operators.

Sometimes it is useful to bundle the specification of which route manipulations should be performed, if the condition evaluates to true, with those that should be performed, if the condition evaluates to false. For example, the latter is useful to specify the behavior, if a service is not booked, while the former is useful to specify the behavior, if the service is booked. We provide this ability via `<task>` and `<default>`. `<default>` allows the policy designer to specify which *fragments* to apply to those sessions not selected by the condition tag in the same direction. To finish the specification of the services we need the ability to specify the names of the service parameters via `parameter`. Note, that parameters are only sensible for available services but not for enforced policies.

Now, that we have discussed how to define a routing policy, we show how to specify a routing policy consisting of a black-hole service and a peering policy. We structure our discussion by first specifying the service informally. Next we analyze how it can be represented using our schema. Finally, we explain how the chosen realization, see Figure 3.6, fullfils the informal specification.

Black-hole service:

Specification: A possible black-hole service offering might be: If a customer sends a route for a prefix tagged with a specific black-hole community then the AS rewrites the next hop of the route such that the traffic is discarded or could be analyzed. To avoid abuse this is only possible for some well-specified set of prefixes, P_s. (Typically consisting of some of the more specifics of the prefixes, P, that a customer is allowed to announce.)

Analysis: Such a service only requires a *sessionset* for the ingress direction. If the customer has booked the service and the black-hole community is present, then

the next hop has to be rewritten. Furthermore, care has to be taken that the prefixes in P_s, which are not part of P, are not rejected. If the customer has not booked the service then routes, tagged with the black-hole community, should be rejected.

Realization: One *sessionset* suffices. Its direction is ingress and the condition is used to select sessions that have booked the service. The `task` for all those sessions that match the condition is to use the *fragments* `ingress_blackhole_community` and `ingress_blackhole_accept`. This ensures that if the community is present the appropriate route manipulations happen and also accepts the prefixes specified in the blackhole_set. If the session has not booked the service, `ingress_blackhole_community_deny` is used to reject invalid routes: routes tagged with the black-hole community. Note that there is a difference between rejecting the route (which could lead to a withdraw, if the route was previously accepted) and an ignore (which would just remove the community that is not allowed). The appropriate action depends on the routeset.

Peering policy:

Specification: An AS may partition its set of neighbors into several classes such as, customers, peers, upstream, etc., and use the following peering policy: announce all routes to its customers; announce all routes learned from customers to its peers and its upstreams.

Analysis: This service requires *sessionsets* for ingress as well as for egress. On ingress the AS marks all routes with the neighbor type of the session over which it receives the route. On egress side it filters routes based on these neighbor type marks.

Realization: One *sessionset* is not sufficient, since different tasks have to be taken for ingress and egress and on egress for customers and peers respectively upstreams. On ingress no distinction between the sessions is needed, as all routes have to be tagged. Accordingly, no condition is needed and the tagging is performed with the *fragment* `ingress_mark_neighbortype`. On egress the distinction is based on the neighbortype of the session. The variable that enables access to the value of this session element is `$session.neighbortype`. The condition is based on the value of the variable. More specifically the value is tested against a fixed string: peer, customer, or upstream. If the neighbortype is peer or upstream the two *fragments* `egress_allow_customer` and `egress_allow_self` are used to ensure the appropriate action. If the neighbortype is customer no specific filtering is necessary.

3.3.3 Back-end module

The *back-end module* provides a library of *fragments* for performing route manipulations and their vendor-specific realizations for use by the *policy module*. Each *fragment* is a capsule for selecting prefix groups and then performing specific BGP operations. The task of realizing a *fragment* can be delegated to an expert for that vendor.

Fragments are the building blocks for our assembly system, the services. Each *fragment* is a configuration template that can consist of a *filter* for selecting routes and an *action* that

manipulates some of the attributes of the selected routes. This suffices as the service specification enables one to use multiple *fragments* in the same *sessionset*. Therefore, should a *fragment* become too complicated, it can be split into two *fragments*. Accordingly, *fragments* are easy to write and simple to verify.

A particularity that has to be addressed is the need of a uniform way to access information from the network and the *policy module* while dealing with the specifics of the router OSes. This is solved in the *back-end module* in the same way as the *policy module* via variables.

We further simplify the task of the *fragment* designer by delegating the handling of some naming conventions and some syntax particulars to the *configurator*. One example is the space of community values. A community value consists of two 16-bit values X and Y written as X:Y, where X is the ASN that defines the mean of the value Y. Given that an AS may want to support a whole range of community signaled services one needs an easily adaptable mapping between the service specifics and community values that can be used for *fragment* specification. The first approach would be to use variables. Yet, since the result can depend on another variable, we rather use a functional notation: e.g.,`community_value(···)`. The parameters to the "function" can be variables, that access parameters from the *network module*. This is useful for example when marking routes by the region in which they entered the AS via `community_value($session.region)`. On routers Asia this will be replaced to `community_value("Asia")`, which in turn gets transformed to the specific community value by evaluating the result of the `community_value`-function. Even though the specific value, e.g., for community-pattern-matching-expression, might depend on the vendor OS[2]. Another problem that can be circumvented using the functional notation is that for some vendors and some purposes, e.g., community filter list, one needs consistent names. Those can be generated in the same fashion, just using a different function name, (`community_filter_name("black-hole")`.

Another particularity is that, in writing policies, one does not want to distinguish, if a parameter used as a filter imposes a restriction to a specific prefix set, or an AS set, or is an expression on the AS path, etc.. Yet, for most vendors each of the above filters requires a different syntax. We solve this problem in the same manner as the naming problem, by using a functional notation of the following form: `filter($blackhole_set)`.

Given that each *fragment* has to contain the templates for each vendor, the top level XML elements currently include `<rpsl>`, `<ios>`, and `<junos>`. RPSL is treated as just another vendor. Once the system supports additional vendor OSes this list has to be expanded. An overview of the elements of the *back-end module* together with an example *fragment* is shown in Figure 3.7.

For RPSL it is easy to follow the idea that each *fragment* consists of filters and actions. Indeed, `<filter>` and `<action>` are the only subelements. Each one contains the RPSL code that realizes the filter or the action. Both filters and actions can embed variable references and function calls that are resolved by the *configurator*.

In Cisco IOS the basic setup of a BGP session, including overall session parameters, is realized within the "router bgp" block of the configuration tree. "Routemap" is the syntax

[2]We do not need to pass the OS as a parameter, since this information is available to the *configurator* via the *network module*.

Fragment	<fragment>
Name	<name> </name>
RPSL realization	<rpsl>
Filter specification	<filter> </filter>
Action specification	<action> </action>
	</rpsl>
IOS realization	<ios>
Session selection/parameter	<bgp> </bgp>
Filter/action specification	<routemap> </routemap>*
Filter list specification	<filterlist> </filterlist>*
	</ios>
JUNOS realization	<junos>
Session selection/parameter	<protocols> </protocols>*
Filter/action specification	<policy-options> </policy-options>*
	</junos>
	</fragment>*

```
<fragment>
<name> ingress_blackhole_community <\name>
  <rpsl>
    <action>
      next-hop=172.24.42.172;
      community.append(community_value("no-export"))
    </action>
    <filter>
      community.contains(community_value("black-hole"))
    </filter>
  </rpsl>
  <ios>
    <bgp>
      neighbor $remoteIPaddr route-map $routemapname_in in
    </bgp>
    <routemap>
      <map>
         route-map $routemapname_in permit $priority
            filter($blackhole_set)
            match community community_filter_name("black-hole")
            set ip next-hop 172.24.42.172
            set community community_value("no-export") additive
      </map>
      <routemapaction>
         continue
      </routemapaction>
    </routemap>
    <filterlist>
         ip community-list expanded community_filter_name("black-hole")
                                    permit community_value("black-hole")
    </filterlist>
  </ios>
  ...
</fragment>
```

Figure 3.7: Examples for *back-end module* entries

element for realizing filters and actions. Each routemap consists of a set of routemap entries. Each individual routemap entry allows the user to combine several kinds of different filters into one filter and to manipulate the attributes of those routes that pass the filter. A filter can, e.g., consist of a prefix list filter and a community list filter. An attribute manipulation can consist of modifying the next-hop and adding a community tag. Each routemap entry can either accept routes that pass its filters, reject routes that pass its filters, or continue the processing of the route at a routemap entry specified via the "continue" keyword. Accordingly, `<bgp>`, `<routemap>`, and `<filterlist>` are the subelements of `<ios>`. Since the "continue" feature is extremely helpful for combining routemap entries from various *fragments* into a combined routemap, `<routemap>` explicitly separates `<map>` from the `<routemapaction>`. Each entry can embed variable references and function calls that are resolved by the *configurator*.

In Juniper JunOS the high-level elements for realizing policies are "protocols" and "policy-options". This is reflected in the subelement tags. The actual filters and actions are specified within the policy-options. The session selection and session parameter specification in addition to the specification of the order in which the policy-options are applied is done within the protocol section.

Since there are a large number of possible kinds of *fragments* we propose to use a naming schema. One possible convention is: `<direction>_(<attribute>|<service>)_<operation>_<parameter>_<accept|deny>`. Here direction stands for ingress or egress. Attributes can be AS path or MED, while "services" may be black-hole, martians, or neighbortype. Operation can be set, add, filter, remove, mark, etc.. Parameter captures if a parameter is necessary or if the action is triggered by a community. Accept is the expected default behavior while deny is explicitly stated. This results in names such as `ingress_blackhole_community`, a *fragment* which is used by the black-hole service in the ingress direction. As the route manipulation of the *fragment* is triggered by a community the parameter is community.

Since *fragments* may manipulate the same attributes it is necessary to resolve conflicts. Should one *fragment* set the value of a route attribute to X and another set it to Y then one needs some method to resolve such conflicts. We propose to solve this problem in two steps. First we partition the *fragments* along the dimensions in which route manipulation occurs. The important aspect is that if two *fragments* belong to two different dimensions than no conflict is possible. Typically a dimension corresponds to one specific attribute. Additional dimensions arise due to certain prefix filters, e.g., a martians filter and a community filter.

The second step is to capture for each dimension the relationships between the *fragments* in a partial order. If two *fragments* manipulate the same object, e.g., *fragment A* via $X = a$ and *fragment* B via $X = b$, then the result of applying *fragment A* and *fragment B* to a session will be a if *fragment A* is less than *fragment B* for attribute X and b if *fragment B* is higher in the partial order. This implies that it is not possible for two *fragments* F_1 and F_2 to both manipulate attributes A and B with partial orders: $F_1 <_A F_2$ and $F_2 <_B F_1$. This is an illegal specification and will be rejected by the *configurator*. In principle this should never occur, since it indicates that the two services, using these *fragments*, are overwriting each others settings. If this is a desired behavior, the check can be circumvented by using four *fragments* instead of two. Two *fragments* for manipulating each of the two attributes.

3.3.4 Summary

The ability to delegate everything BGP-specific to the *back-end module* and everything network-specific to the *network module* enables the formulation of a routing policy at a level of abstraction previously unavailable. This is accentuated by the specifications of the sample policies, see Figure 3.7.

Our approach differs from other XML based languages such as for example NETCONF [52] and NetML [131]. NETCONF uses XML requests and queries to provide a simple mechanism for managing configuration data and system state of network devices. NetML uses XML to describe computer networks in a vendor-independent manner. While both provide nice abstractions of the various router configuration languages and highlight the differences in the approaches of the vendors, they do not have the ability or the goal of expressing abstract routing policy.

3.4 Configurator

In collaboration with Deutsche Telekom, we have developed a prototype, the *configurator*, that generates router *configlets* for the eBGP parts of each router in the network based on the information provided in the XML databases. In this section, we describe its design and operation.

Syntax and semantic checks: The tool starts by parsing the XML databases and using XSD schemas to perform integrity checks of the cross-references in the *network module* database: are the router names used in the bgpsession elements defined; are the bgpsession who's names are used in the bgpneighbor elements valid? Other consistency checks within the modules include checks for collisions in the name space, e.g., is the same ASN used for two different neighbors, etc.. Next the tool performs additional consistency checks between the modules. It checks that the *fragment* and *service* definitions only reference existing variables, meaning elements in the network database, call functions that are realized in the current version of the *configurator*, and if all *fragments* exist that are used in the services, etc..

Inheritance: Inheritance is a excellent method since it simplifies the specification of the network database and it ensures that it is possible to highlight where exceptions for specific BGP sessions are necessary. The *configurator* deals with inheritance by determining for each BGP session which services with which parameters are indeed booked. By storing this information separately the *configurator* still knows which service is session-specific and which service is neighbor-specific.

Configlet generation for RPSL: In principle the generation of the RPSL documentation could be done by defining multiple `import`/`export` statements on a per session basis (see Section 2.2 on page 18). Yet, the drawback of this approach is that it does not take advantage of RPSL's capabilities of abstraction by grouping objects using session sets or AS sets. Our goal is to use this ability to document which BGP sessions to which ASes have common parts in their policies.

Using the refine construct that RPSL offers, let us separate route manipulations that are independent of each others. Accordingly, each "dimension" of route manipulation is realized via

a refine block[3]. As each RPSL statement has to be added to some refine block we proceed by handling dimensions within the outer loop. The next loop addresses everything that needs to be done for this dimension. We start with all *fragments* in this dimension and determine the *sessionsets* and with the *sessionsets* the services that use these *fragments*. To resolve the conflicts within each dimension the *fragments* are processed in accordance with the partial order for the dimension. Once the appropriate *sessionsets* are determined it is possible to evaluate the *condition* within the *sessionset* to select the appropriate group of BGP sessions, construct the RPSL statements from the *fragments*, and append these to the refine block.

At this point we explore, if there is a way to partition the session group, e.g., by using RPSL session sets or by using one or more AS sets. One can use a RPSL session set if the RPSL statement for the *fragment* is the same for all sessions in the group. This is possible if the RPSL *fragment* elements do not contain any session specific variables. Otherwise we explore if it is possible to split the session group into subgroups, e.g., into AS sets. This is possible if the RPSL elements do not contain any neighbor specific variables and if the *condition* is not session specific. Otherwise even finer partitions are considered. In the worst case no subgroups exist and an RPSL statement for each session is generated. Once the subgroups, RPSL session sets, AS sets, etc., have been identified, the RPSL statement is added to the refine block after restricting it to the appropriate level.

Vendor-specific configlet generation: In contrast to the generation of the RPSL documentation the *configlet* generation for Juniper and Cisco routers is done on a per session basis. The *configlet* generation for Juniper routers proceeds in a similar fashion as the one for Cisco routers. We therefore only discuss the *configlet* generation for Cisco IOS. Our assumption is that the *fragments* of the *back-end module* can utilize the Cisco continue feature. Unfortunately the continue feature is currently only available in rather recent releases Cisco IOS[4]. The *configlet* generation for IOS proceeds in three steps: we first determine for each session which *fragments* have to be combined; next we generate a preliminary *configlet* for each session; finally, we deal with specifics of Cisco IOS and optimize the generated *configlets* with embedded RPSL expressions for filter generation.

The first step involves selecting for each `<enforced_policy>`, `<available_service>`, and each *sessionset* within the corresponding services, which bgpsessions fulfill the *condition* of the *sessionset*. Next we evaluate the service parameters for each session that meets the *condition*, (in the context of this bgpsession) and store the result together with the *fragments* listed under `<task>` in a list of *fragments* at the session. For all sessions that do not meet the *condition*, the evaluated parameters and the *fragments* under `<default>` are stored. In effect this list gathers for each session the names of all *fragments* with their parameters that are applied to this session.

In the second step we proceed session by session and direction by direction to generate preliminary *configlets* based on the list of *fragments*. In this step the *configurator* combines the *fragments* in the appropriate order and ensures that all variables, all function calls, and all parameter values of all subelements of `<ios>` are replaced with the appropriate values. The *fragments* are processed in an order that respects the partial order of the *fragments*. For

[3] In order to accept a route, there has to be a matching RPSL statement in each refine block. See Chapter 2.2 for more background information.

[4] Inbound support for Cisco BGP route-map continue was introduced in 12.0(24)S and support for outbound policies was introduces in 12.0(31)S.

each subelement the results are joined. This is a simple merge for `<bgp>` and `<list>`. Any conflict indicates that the *fragment* designer made a mistake and the *configurator* returns an error. For "route-maps" the join is not that simple and postponed to the final processing step. At this point the new piece is just added to the pieces from the other *fragments*.

Once all *fragments* have been processed the join of the route-map entries can be completed. Just chaining the individual route-map entries according to the partial order is not sufficient. For example, if *fragment A* and *fragment B* manipulate attribute x with $A <_x B$, one needs to ensure that attribute x will have the value of *fragment A* even if the filters in both *fragments* match. Even though the serialization ensures that the partial order is respected, it does not ensure that route filtering proceeds at the appropriate place: e.g., the first route-map entry from a *fragment* of the next dimension. This is accomplished by rewriting the continue entries of the joined route-map entry.

As the Cisco continue feature is not always available realized with regular route-maps (without continue) by computing a *convolution* of all route-map entries. A convolution explores all possible combinations of route-map entries and arranges them in an order that has the most restrictive filter at the beginning and the least restrictive one at end of the new route-map. A combination of two route-map entries is computed by joining their filters with a logical AND and their attribute manipulations with a logical OR operation. The latter is the case as long as the manipulations are orthogonal. Unfortunately route-map entries do not support arbitrary filter combinations using logical AND operation. For example it is not possible to combine two prefix filter lists A and B using a logical AND in the same route-map entry. Instead this can be realized by generating a third prefix filter list, $C = A$ AND B, for use in the route-map entry. We solve this complication by using RtConfig to generate the appropriate filter lists, specified via RPSL statements, and adjust the route-map entries to use the new filter.

Finally, duplicate entries, filters that are no longer used, and route-map entries that are never reached are removed. For example, if route-map entry 1 uses filter A and route-map entry 2 uses filter A then the combined entry from 1 and 2 suffices. The entry for only 1 and the entry for only 2 can be removed as it will never be reached. If entry 2 uses a different filter all three route-map entries have to be present.

RtConfig: At this point the output of the *configurator* is a documentation of the routing policy in RPSL and a set of *configlets* with embedded RPSL expressions for every bgpsession in the network. These embedded RPSL expressions need to be replaced with actual filter statements before the *configlet* can be uploaded to the router. For this purpose we rely on RtConfig. RtConfig is able to resolve all RPSL filter statements by using a local cache of RPSL objects for AS internal filters together with the IRR database. This implies that if a customer changes his entry in the IRR database, this change is automatically propagated to the *configlet* the next time the *configlet* is regenerated.

Upload: If there has been a change in the *configlet* it needs to be uploaded to the appropriate router. For this purpose it is combined with some base configuration code which ensures redistribution of routes, deals with static routes, etc.. The upload for Juniper router is nicely supported by their release management. The upload to Cisco routers is more troublesome.

3.5 Operational considerations

In this section we show examples of generated *configlets* and discuss some challenges that an operator faces.

3.5.1 Generated *configlets*: Examples

For the example of Figure 3.2 (page 28) the generated RPSL documentation is shown in Figure 3.8. The various dimensions are apparent in the different refine blocks for both directions, e.g., for ingress the route filtering is handled in a different dimension than the black-hole service. The differences between the default action and those based on the condition are easily seen for the black-hole service. If the black-hole service is not booked or used for a route that is not in the appropriate routeset (`RS-AS3-BLACKHOLE`), then the route is rejected. Otherwise the appropriate action is taken. The realization of different cases is nicely highlighted by the handling of med for AS 2 on session c2. Here, the value is set to 100 per default. If the route is part of the RS-2000 the value is 2000, if the route is part of RS-3500 the value is 3500. For all other sessions the med is set to 500. As the example is constructed to enable different services for different sessions, the *configurator* has only limited success in using session sets, e.g., `AS-CUST` and `AS-PEER` (from the peering policy) each consisting of a single AS.

The main difference between the generated RPSL and the vendor-specific code is, that in RPSL the full eBGP policy of the whole network can be specified within one statement. At the moment RPSL has the drawback that it is not capable of matching or deleting communities specified by patterns. All communities have to be listed explicitly. In some cases this is not feasible, e.g., the cleaning of internal communities at ingress. Therefore we currently lose some functionality.

The generated code for a Cisco router for BGP session $c1$ on $r1$ is shown in Figure 3.9. Each of the different *fragments* contributes some pieces to the "router bgp" and "route-map" subsection of the *configlet*. The first entry in the routemap_out part is the result of martians filter. The second one is the result of clearing internal communities. The final part is realizing the appropriate piece of the peering policy. The route-map for ingress filtering has a few additional pieces: entry 700 marks the received routes as coming from a customer (community `1:1`), entry 800 realizes black-holing. First the next hop is rewritten, then the black-hole community is added to the route (as route-map entry 600 deletes all communities).

To highlight the differences between the generated code for an IOS version that does not support the continue feature Figure 3.10 shows the route-maps of the generated code for the same session. Now each route-map entry consists of a self-contained set of route manipulations that are appropriate for the filters. In this case, the order of the route-map entries is crucial. It goes from the most restrictive filter to the least restrictive one.

3.5.2 Experiences

Our system is in production use at a `AS 3320`. It is used for generating the *configlets* for the eBGP sections of several hundred routers and it has proven to be beneficial to consider

```
aut-num: AS1
import: from AS-ANY accept ANY;
  refine {    # from ingress_route_filter:
     from AS2 accept AS2:RS-I;
     from AS3 accept RS-AS3-IMPORT;
     from AS2 accept RS-AS3-BLACKHOLE AND community.contains(65000:0);
  } refine {  # from blackhole
     from AS3 2.1.1.2 at 1.0.0.2 action next-hop=172.24.42.172;
          community.append(no_export);
          accept community.contains(65000:0) AND RS-AS3-BLACKHOLE;
     from AS-ANY accept not community.contains(65000:0);
  } refine {  # from ingress_peer
     from AS-CUST action community.append(1:1) accept ANY;
     from AS-PEER action community.append(1:2) accept ANY;
  }
export: to AS-ANY announce ANY;
  refine {    # from egress_route_filter:
     to AS2 announce AS2:RS-E;
     to AS3 announce RS-AS3-EXPORT;
  } refine {  # from egress_wellknowncommunites_filter
     to AS-ANY announce not community.contains(no_export, no_advertise);
  } refine {  # from peering
     to AS-CUST announce community.contains(1:0) OR
        community.contains(1:1) OR community.contains(1:2)
        OR community.contains(1:3);
     to AS-PEER announce community.contains(1:0)
        OR community.contains(1:1);
  } refine {  # from egress_med
     to AS2 2.1.1.2 at 1.0.0.2 action med=2000; announce RS-2000;
     to AS2 2.1.1.2 at 1.0.0.2 action med=3500; announce RS-3500;
     to AS2 2.1.1.2 at 1.0.0.2 action med=100; announce ANY;
     to AS-ANY action med=500; announce ANY;
  }
```

Figure 3.8: Generated documentation in RPSL.

the routing policy as a first class entity. Each service and each policy is now well specified, which increases the degree of transparency for the ISP. In addition, it is easy to add, expand, and change services and policies.

The flexibility of the system proved to be crucial during the migration phase: moving from a "legacy" state to an automatically generated state. All of the network specific information had to be gathered for the *network module*. In addition, the routing policy in use has to be expressed using our abstraction. The ability of using almost arbitrary router configuration commands simplified reproduction of the legacy state. Once the legacy state could be regenerated and uploaded to the routers the next phase of transitioning to the desired routing policy was possible. This involves, as a first step, the definition of the desired routing policy. After definition each of the policies and services has to be introduced into the network. This can involve multiple stages. After all it is impossible to change the configuration of all routers at exactly the same time. If the policy or service depends on some precondition, e.g., that all routes are marked with a community on ingress, then this precondition has to be introduced on all sessions before, in a second stage, the postcondition can be introduced, e.g., that routes are filtered based the community.

Specifying the desired routing policy is still not trivial as the system does not restrict the routing policy designer in realizing various variants. Among the choices to consider are: how to realize filters, e.g., prefix and community filters. Does the policy include the acceptance of all routes that are not filtered due to some reasons or another? Are all inappropriate communities removed in one filter or should each service, that is signaled via a community, implement its

```
router bgp 1
  neighbor 2.1.1.2 remote-as 2
  neighbor 2.1.1.2 next-hop-self
  neighbor 2.1.1.2 route-map c1_routemap_in in
  neighbor 2.1.1.2 route-map c1_routemap_out out
!
route-map c1_routemap_out deny 100
  match ip address prefix-list martians
route-map c1_routemap_out permit 200
  set comm-list out_fltr_communities delete
  continue 300
route-map c1_routemap_out permit 300
  match community export_all
!
route-map c1_routemap_in deny 500
  match ip address prefix-list martians
route-map c1_routemap_in permit 600
  set comm-list in_fltr_communities delete
  continue 700
route-map c1_routemap_in permit 700
  set community 1:1 additive
  continue 800
route-map c1_routemap_in permit 800
  match ip address prefix-list c1-blackhole
  match community blackhole
  set ip next-hop 172.24.42.172
  set community no-export additive
  continue 900
route-map c1_routemap_in permit 900
  match ip address prefix-list c1-import
route-map c1_routemap_in permit 910
  match ip address prefix-list c1-blackhole
  match community blackhole
!
ip community-list expanded in_fltr_communities permit _1:.*_
ip community-list expanded in_fltr_communities permit 64900:.*
ip community-list expanded out_fltr_communities permit 64900:.*
ip community-list expanded blackhole permit 65000:0_
ip community-list expanded export_all permit _1:[0123]_
!
ip prefix-list c1-import permit 2.1.1.0/22 ge 24 le 24
ip prefix-list c1-blackhole permit 2.1.1.0/24 ge 32 le 32
ip prefix-list c1-blackhole permit 2.1.1.0/24
ip prefix-list martians permit ...
```

Figure 3.9: Generated *configlet* for $c1$ using routemap feature continue.

part of the community filter? It is possible to realize any of these options within our system.

We found that using vendor-specific code inside the *fragments* is useful as the designer has complete control over the generated *configlets*. *Fragments* can be extended, rewritten, repartitioned, and rearranged as needed. One aspect to be aware of is the choice of embedding more functionality inside a single *fragment* vs. using multiple *fragments*. The first enables more control over the generated *configlet* and therefore its optimality, while the second improves reusability. Most policies and services are realized using just one, two, or at most three *fragments*. Using the suggested naming schema is helpful for associating *fragments* with dimensions and expressing the conflicts between the *fragments*, currently the most awkward part of the system.

Separating the network-specific parts into its own database has proven to be essential as it separates the task of the network administrator from those of the routing policy designer. In addition, the documentation of the current state of the network as well as the routing policy (in the *policy module* as well as in RPSL) is extremely helpful.

```
route-map c1_routemap_in deny 10
  match ip address prefix-list martians
route-map c1_routemap_in permit 76
  match ip address prefix-list c1-import-c1-blackhole
  match community blackhole
  set comm-list in_fltr_communities delete
  set ip next-hop 172.24.42.172
  set community 1:1 no-export additive
route-map c1_routemap_in permit 77
  match ip address prefix-list c1-blackhole
  match community blackhole
  set comm-list in_fltr_communities delete
  set ip next-hop 172.24.42.172
  set community 1:1 no-export additive
route-map c1_routemap_in permit 78
  match ip address prefix-list c1-import
  set comm-list in_fltr_communities delete
  set community 1:1 additive
!
route-map c1_routemap_out deny 10
  match ip address prefix-list martians
route-map c1_routemap_out permit 15
  match community export_all
  set comm-list out_fltr_communities delete
!
ip prefix-list c1-import-c1-blackhole permit 2.1.1.0/24
```

Figure 3.10: Generated *configlet* for $c1$ after convolution (without continue).

3.6 Summary

In the past networking research has tended to focus on the individual parts of the network rather than on the network as a whole. In this thesis we propose a system that can help raise the level of abstraction for a small but crucial piece of the overall Internet: the routing policy of one AS.

For this purpose we tackle the problem of identifying the concepts underlying an AS-wide routing policy. Based on this understanding we propose a system for managing AS-wide routing policies as first class entities. This includes providing a flexible, extensible, and scalable framework for formulating the routing policy, eliminating the need of manual configuration of IP routers for the purpose of realizing the policy, and respecting the division of responsibilities within the ISP.

As such we have presented a data model that enables the definition of an abstract routing policy but also allows its concrete realization on the routers in the network. It is now possible to express the enforced policies and available services of a routing policy precisely as well as in a compact format. The experiences gathered from the production use at a large ISP have shown that transitioning from a network configuration grown bottom up to one with a well documented routing policy that is specified top down is possible.

Clearly, the work reported herein has not exhausted the problem area, and there is much more that can be done. One limitation of the current system is that it does not have support for iBGP. Other avenues for expanding the capability of the system are support for IPv6 and VPNs as well as support for additional router vendors.

4 BGP Dynamics

Timothy G. Griffin states: "The following scenario MUST take place within the next few years: The Inter-domain routing system will enter a state of non-convergence that is so disruptive as to effectively bring down large portions of the Internet. The problem will be due to unforeseen global interactions of locally defined routing policies. Furthermore, no one ISP will have enough knowledge to identify and debug the problem." [132]

While we have seen in the last chapter how policies are realized statically, we now look at the dynamics that such a complex system of about $20,000$ competing ASes induce. Thus, we now delve into some BGP details and discuss what kind of instability creators exist, how instabilities propagate through the actual network and what kind of updates may be visible at an observation point. This builds the necessary framework for the following chapters, where we will look into the characteristics of BGP dynamics.

4.1 Instability creators

A *BGP instability* is an event that impacts inter-AS routing, see Table 4.1. We exclude from the notion of an "event" the receipt of an eBGP update message. Rather, we consider the eBGP update message as a consequence of some instability. That is, in response to a BGP instability, a BGP-speaking router initiates a BGP update that propagates an attribute change from one BGP peer to another. BGP instabilities can have their origin at the source of the prefix, in the input filters, the decision process, the output filters, or through the availability of BGP sessions. While the filters are limited to the BGP attributes, the decision process also uses the following other resources: link availability, node reachability, IGP cost, and next-hop IP addresses.

Accordingly, the BGP instabilities can be initiated by: changes to the availability of BGP sessions, the session filters, the link and/or node availability, introduction or withdrawal of prefixes including aggregation, IGP cost changes, or IP address changes. Typical examples for each of these are given in Table 4.1. Note that one kind of change, e.g., a node failure, may imply other failures, e.g., multiple link failures, which can in turn imply other changes, e.g., IGP cost changes.

Next we consider what kind of BGP updates these BGP instabilities impose. Here the first question is which prefixes will see any updates, referred to as *relevant prefixes*. An instability is relevant to a prefix if an attribute of its best path or if its filter policy is changed. In terms of blaming an AS for an instability, one has to distinguish between changes within an AS, called *internal changes*, and between ASes, called *external changes*. Typical internal changes are those associated with iBGP sessions. Others are IGP traffic engineering operations changing IGP metrics. Typical external changes are changes to eBGP sessions, e.g., for the purpose

Instability	Examples
BGP session availability	session establishment/teardown/reset
BGP session filters	filter changes and/or BGP attribute manipulations usually imply session (soft-)reset or graceful restart
IGP costs changes	IGP metric changes, link or node failures/repairs
IP address changes	renumbering, link or node failures/repairs
link/node availability	link failures/repairs, node failures/repairs may cause BGP session availability changes and IGP cost changes
originator changes	addition/deletion of network prefixes
route flap damping	delay of the propagation of updates

Table 4.1: BGP instability and their typical causes.

of traffic engineering, and may include subaggregation, or aggregation of prefixes, changing filter rules, AS path prepending, etc.. The difference between internal and external changes is that the latter usually only impacts the prefixes whose best paths include the affected session and therefore both ASes. Internal changes can impact prefixes with diverse next hop and previous hop ASes.

The next question is what kind of updates a prefix will experience. One important factor is the diversity of routes available to the best path selection process. A link or BGP session failure can disrupt the connectivity between two ASes and affect many prefixes. The existence of an alternative route causes the selection of a new best path, which will not be propagated if it has the same attribute values or if it is caught in the output filter. Otherwise the AS announces the existence of an alternative route. This route may differ from the old one in either the AS path, the next hop, or other attribute changes. An AS path change is necessary if reachability via the old AS path is no longer given, e.g., if two ASes have a single eBGP session and it fails, or if an internal link failure causes a network split, or if the reachability via the new AS path is more attractive.

Yet, by design not all instabilities impact reachability, e.g., peering usually requires BGP sessions at at least three diverse locations and ASes usually have multiple upstream providers. This diversity implies that the addition of a new route or the withdraw of a route usually just adds one more variant to the best path selection process. For example if two eBGP sessions exist between two ASes one may expect to learn two routes to each prefix routed via these sessions which, if a consistent routing policy is used, will have the same AS path. Accordingly, the BGP decision process may choose between multiple routes with the same AS path. In addition, if the prefix is reachable via another AS, further alternatives are available to the decision process. If the routes have an AS path of the same length, the decision about which route is best depends on the MED values, the IGP distance metrics, and the next hop IP addresses. Accordingly, if a better route becomes available or the best route becomes unavailable, this either lead to a AS path change, a next hop change with or without AS path change, or no change at all if there are multiple peerings between the same routers[1]. In this case, each router may make a different decision which implies that AS path changes are likely to occur for only a subset of the relevant ASes.

Since the IGP metric and the router-ID are used as tie-breakers in the BGP decision process, changes to these cause instabilities to those prefixes using this AS as a transit AS. While IP address changes are expected to be rare, intra-domain traffic tweaking is more widespread. Accordingly, a sizable number of prefixes may see AS path and/or next hop changes. Inter-

[1] Note that this ignores the steps above AS path change in the BGP decision process, e.g., local preference.

Instability/Condition	BGP updates	expected number of affected prefixes	responsible AS	comment
eBGP session availability				
single session	AS path changes	all relevant prefixes	both ASes	
multiple sessions	next hop changes	subset of rel. prefixes	both ASes	no alt. AS path of equal length
multiple sessions	AS path/next hop changes	subset of rel. prefixes	both ASes	alt. AS path of equal length
iBGP session availability				
reachability impacted	AS path changes	all relevant prefixes	AS	
reachability not impacted	next hop changes	subset of rel. prefixes	AS	
eBGP session filter	attribute changes	small subset of rel. prefixes	both ASes	
eBGP attribute changes	attribute changes	small subset of rel. prefixes	both ASes	
IGP cost change	next hop changes	subset of rel. prefixes	AS	tie breaker in BGP decision
	AS path changes	subset of rel. prefixes	AS	alt. AS path of equal length
IP address changes	next hop changes	subset of rel. prefixes	AS	tie breaker in BGP decision
	AS path changes	subset of rel. prefixes	AS	alt. AS path of equal length
link availability				
internal no session change	next hop/AS path change	subset of rel. prefixes	none	via IGP cost changes
internal with session change	AS path changes	all relevant prefixes	AS	via reachability problems
external no session change	none	none	none	unlikely
external with session change	next hop/AS path changes	all relevant prefixes	AS	via eBGP multiple session
node availability				= multiple "link availability"
originator changes				
single homed	new updates/withdraws	single prefix	originator AS	
multiple homed	attribute changes	single prefix	originator AS	

Table 4.2: Effects of BGP instability

domain traffic engineering [74] takes advantage of the full spectrum of options that BGP provides including but not limited to AS path prepending, filtering, local preference, prefix deaggregation, prefix aggregation, IGP metric changes, MED changes, eBGP session parameter changes. But since automatic tools are still a rarity, most of the tuning is still done by hand.

In summary, most instability events cause BGP updates to a number of prefixes at about the same time. But not all of these have to result in an AS path change. Other instability events are of concern to only individual prefixes. Note that human misconfigurations of BGP [19] are either unintended changes impacting single prefixes, IGP costs, or whole BGP sessions. Table 4.2 tabulates the various possibilities of BGP instabilities.

4.2 Instability propagation

In this Section we consider the effects of a routing instability in terms of how BGP updates propagate through the Internet, and where they are observable. While some BGP updates change almost all attributes, quite a few only change a single attribute. We classify updates according to the attribute change that has the biggest impact with regard to how far the update will have to be propagated, see Table 4.3.

Next we define an abstraction of the actual AS topology that we use in our arguments below. Each reasonably sized AS consists of a number of routers that have between them full iBGP connectivity, either via a full iBGP mesh, route reflectors, or confederations. Accordingly, we model each AS as a clique, one node for each router and an edge for each node pair. Each eBGP session between AS 1 and 2 corresponds to an edge between a node of the clique of AS 1 and a node of AS 2. For simplicity and to ensure that AS-internal effects are captured, we assume that each AS has enough nodes so that no two eBGP peering sessions are terminated

Class of updates	subclass	discussion
Local pref changes	multi-homed customer	might cause next hop change
	route selection between peer/upstream/customer	might cause next hop change should cause AS path change
AS path changes	withdraw	the only route is no longer available corresponds to "bad news".
	new "better" route	corresponds to "good news" and can mean a new route with - a shorter AS path is available (ignoring local pref) - same length AS path is available (ignoring local pref) with new MED smaller and same next hop AS / new IGP cost or ID smaller if next hop AS changes - longer AS path length if local pref or weight is used
	implicit withdraw old route "better"	corresponds to "bad news" and can mean the route with - a shorter AS path no longer available (ignoring local pref) - same length AS path is available (ignoring local pref) with new MED larger and same next hop AS / new IGP cost or ID larger if next hop AS changes - shorter AS path length if local pref or weight is used
Origin changes	IGP/Incomplete to eBGP	implies changes to the AS path, i.e., current AS is no longer the originator
	eBGP to Incomplete/IGP	implies changes to the AS path, i.e., current AS is now the originator
	Incomplete to IGP	change of status, likely together with next hop change
	IGP to Incomplete	change of status, likely together with next hop change
MED changes		MEDs are comparable for paths from the same AS MED changes may reorder paths from the same AS which may cause next hop changes MED changes may change ties between path from different ASes which may cause AS path changes
Next hop changes	without AS path changes	new route uses different eBGP or iBGP session between same peers
Community changes		need to be propagated since they are transitive. Agreement on semantic of global community values missing

Table 4.3: Effects of BGP updates

at the same node. Now consider some prefix p and its routing table entries at all routers. The graph that is induced by choosing the edges of those sessions over which the router received the update and directing them towards the router is a directed acyclic graph (DAG), as long as there are no temporary loops induced by BGP. Any changes to the BGP sessions may impose changes to the DAG by adding or deleting edges or changing their direction, and all updates for this prefix p have to traverse a subset of this DAG. Hereby one has to keep in mind that each update can only traverse each edge in one direction and that each router will only propagate information about prefix p if its best route has changed.

Next we consider what this implies for our above classification of updates. Pure next hop changes matter for the current AS and in some cases may have to be propagated to the neighbor ASes. For example sometimes the ingress is marked with a community, if not filtered at the egress, this results in update propagation. In the worst case, AS path changes and withdrawals have to be propagated along the same subgraph. But in most cases, due to the high connectivity of the Internet, other alternative paths exist. In this case, the update has to reach only those nodes that benefit from the new alternative path or those nodes that have to now choose an alternative path.

In summary, while one expects BGP updates to several prefixes if a change to an eBGP session is the instability originator, some or all of the updates may be rather local. But they can also impose major non-localizable BGP updates, e.g., if AS path changes are involved. This may depend on the specific policy of the AS, the ISP's topology, etc.. Changes to individual prefixes may have only local impact or a global one.

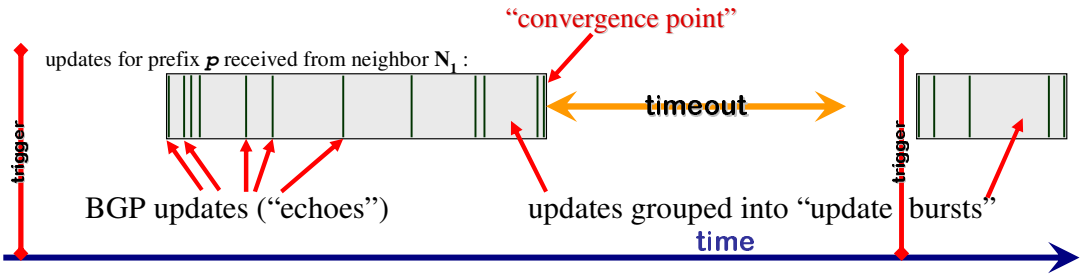


Figure 4.1: Illustration of BGP update clustering.

4.3 BGP Convergence Properties

While BGP dynamics are studied extensively in the last few years (see Section 2.3.2, page 22, for a discussion of related work), there are still a number of open questions. Despite the huge commercial success of the Internet, there are only a limited number of tools that help troubleshooting network problems. As a result ISPs often notice problems only when a customer calls and complains. This is due to the complex interactions between the independent systems, including the unknown local policies, internal topologies and settings of parameters, etc.. Yet, these factors influence the behavior of the routing system (i.e., best path selection). Given all those unknown factors, it is hard to understand the root cause of a problem within the Internet (see Chapters 4 and 5).

Therefore it is crucial to understand the characteristics of the raw BGP data that are observable in the Internet. Not only operators can benefit from tools that detect problematic routing conditions (e.g., hijacked or oscillating prefixes), vendors may improve the router code (e.g., based on the insights of the propagation patterns), also researchers can better sense problems and develop solutions. In addition, replacement protocols should be designed with an in-depth understanding what worked today, what did not, and why.

To approach this we start by discussing a methodology for clustering BGP updates on a per prefix and per peer basis in Section 4.4. After a short discussion of the data sets in Section 4.5 we use the methodology to look at BGP beacons in Section 4.6. As time and location of the beacon event is known, it is possible to study convergence properties of the Internet. From the insights gathered from the beacon analysis we study the characteristics of all BGP updates in Section 4.7. Finally we summerize our findings in Section 4.8.

4.4 Methodology

To better understand what is happening in the Internet, we need a notion of what convergence is, how to measure convergence and how derive this from raw BGP update traces.

The difficulties already starts with defining the term "convergence". A lot of practioners call convergence the time needed to build a full table after a router reboot or a hard session reset. Others define convergence at the router level as the time needed for a router to respond to a new announcement or withdraw. We follow the notion of Labovitz et al. [84] and call the *convergence point* of a prefix that moment, when all routers in the Internet have reached a stable routing state towards that prefix.

Recall from Section 2.1.1 (see page 14) that instabilities can lead to path explorations involving many BGP updates to spread across a significant time period. Thus, not all updates are equally important (e.g., consider the amount of traffic that is flowing along a route that gets replaced two seconds after it was announced, compared to one that is stable for several weeks). So, what are "important" updates and what are "less-important" updates?

Recall, that the goal is still to have constant traffic flows. Therefore operational practice is to optimize routing to achieve a stable state [90]. Because of that we can group updates observed at a given observation point and for a given prefix into a burst of updates, just as one would group packets into flows [133], using a timeout. With that we identify those updates that remain stable on that observation point for a time less that the specified timeout. Figure 4.1 illustrates this concept. The vertical lines symbolize BGP updates, which can be observed at an observation point N_1 and are triggered by an instability event. Accordingly, the last update in the burst approximates a *stable route*, it is also called the "*convergence point*" or "*new_stable_route*". Intermediate updates, e.g., due to path exploration, are called "*echoes*" throughout this thesis. And the valid route before the beginning of an update burst is the *old_stable_route* (or *previous_stable_route*). It is the new_stable_route of the previous burst at this observation point. If no update burst was previously observed, we take the appropriate route from the table dump.

Typically neither the cause of the instability nor its time nor its location are known. Yet, in the case of the BGP beacons, researchers artifically inject faults at well defined times. This means that the time is known and we say that the *beacon duration* is the time starting with the beacon event and ending at the convergence point. This is not to be confused with the *burst duration*, which is the time from the first update in a burst until the last update in that burst. We use this latter terminology if the time of the triggering event is unknown. In effect the burst timeout allows us to separate events.

The choice of the timeout is crucial as is illustrated in Figure 4.2. On one side, if two triggering events are very close in time a large timeout would group updates from the two events together. For example, as one would hope, that after a link failure the repair is performed quickly, but with a large timeout the updates from the failure and the repair are together in one event. On the other side, if a small timeout is used, then events may be separated even though they still belong to the same convergence process. In this chapter we are interested in characterizing the convergence properties of BGP. To be able to study the impact of route flap damping on convergence, we use a large timeout, typically larger than one hour. In the next chapter (Chapter 5) our motivation is different as we consider multiple events due to route flap damping as new events. Therefore we chose timeouts ranging from 2 to 16 minutes.

With this methodology it is possible to identify persistent flapping prefixes (e.g., [76, 124, 134, 135]).

4.5 Data sets

Our characterization work is based on raw external BGP routing table dumps and update traces that we obtain from Ripe [48], and Routeviews [49]. Throughout this chapter we only present results in an exemplary fashion for the following raw data sets.

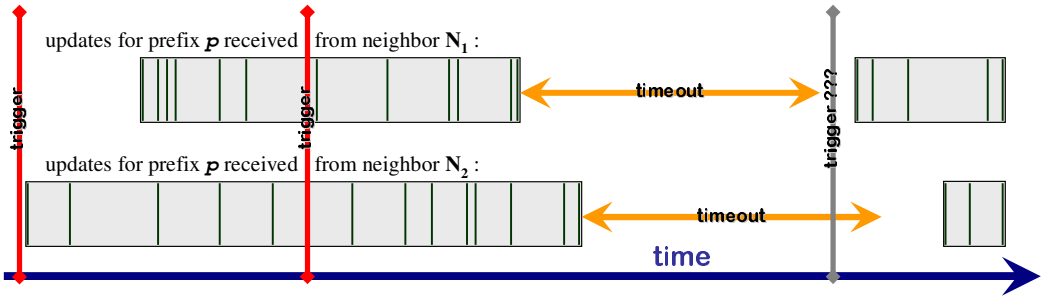


Figure 4.2: Limitations of BGP update bursts.

The beacon study presented in Section 4.6 is based on BGP traces starting October 1, 2002 to January 31, 2003. 13 BGP beacon prefixes are included in this characterization[2]. This results in 617,299 beacon events (including announcement and withdraw events). Note that not all beacons are visible on all observation points. Indeed, only about 40% of the expected beacon events are observable.

The raw data characterization study presented in Section 4.7 is based on 11 to 14 full feeds collected by the RIPE RRC00 [48] collector in the time from January 1, 2003 to April 24, 2003. The table dumps contains between 100*k* - 123*k* prefixes. In January (1/1 - 1/31) about 69 million updates are recored, in February (2/1-2/28) about 61 million updates are collected, in March (3/1 - 3/31) about 47 million updates are studied, and in April (4/1- 4/24) about 44 million updates. The trace ends on 4/24 2003 at 17h UTC. Note that there is a gap on March 6 for about four hours (6:08-10:13 UTC) due to an outage of the collector. We also consider two specific peering sessions from RRC00, starting 01/14/02, 1am to 01/20/02, 1:10am.

The estimated error, i.e., missing updates, in all traces is less than 1%. We estimated the error by loading a routing table dump in a virtual RIB, then applying all updates to this RIB, and comparing at the end of the trace a routing table dump with the virtual RIB.

4.6 BGP Beacons

The beacons are a starting basis for our studies as the time and location of the triggering event is known. Our findings are consistent with Mao et al. [101]. For more details see [102].

We analyze 617,299 BGP beacon events, 301,295 of them are announce events (*A-events*), while 316,004 are withdraw events (*W-events*). 272,605 (90%) of the A-events show "only" updates within the first two minutes after the triggering event. Yet converging a W-event usually takes some more time. After two minutes only 48.5% reached a stable state and it takes 220 seconds to convergence 90% of all withdraw events. After 6 minutes 304,848 (96.5%) of the withdraw events are converged.

Figure 4.3 and Figure 4.4 show the convergence time of all analyzed beacon events. Figure 4.3 shows a smoothed density plot of the beacon event durations. The *x* axis plots the seconds since the triggering beacon event, while the *y* axis shows the density function. Note

[2]Note that update traces before November 9, 2002 (0:00 GMT) from the Routeviews collector are excluded because of inaccurate time synchronization of the collector. In addition the beacon operated by Andrew Partan starting November 21, 14:00 and David Meyer's beacon before November 16 cannot be used due to a schedule change.

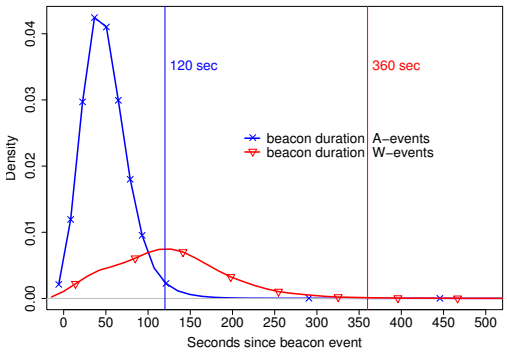


Figure 4.3: Beacon durations of all events. [102]

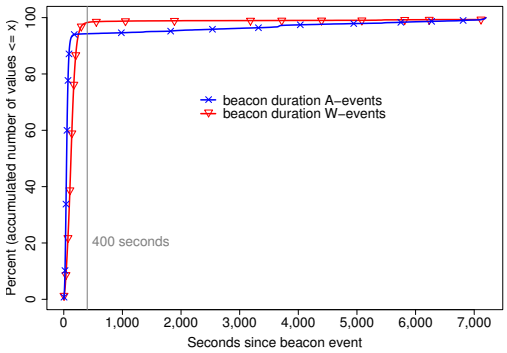


Figure 4.4: CDF of beacon durations. [102]

that the plot is cut off at 500 seconds for clarity but there are beacon events that did not converge within the 2 hours. (This is the maximum duration, because a beacon event is triggered every two hours, therefore for longer convergence times, the two consecutive events are not distinguishable anymore.) This can be seen in Figure 4.4, which shows the Cumulative Distribution Function (CDF) of the beacon duration of both A-events and W-events. This CDF shows on the x axis the duration in seconds and on the y axis the probability that an event converged **before** the given duration.

Next, in Section 4.6.1, we study the prevalent convergence behavior, while in Section 4.6.2 we look at those events that show long converge times.

4.6.1 Prevalent behavior

We say that an beacon event shows prevalent convergence behavior if it converges within *two minutes* in the case of an A-event, and if it converges within *six minutes* in the case of an W-event. With this notion 95.3% of all recorded events show prevalent convergence behavior. (Figure 4.3 show vertical lines at two and six minutes.) Of course, we make sure that the last update that is observed is of the corresponding type. Which means an A- (W-) event converges with an network reachable (unreachable) message. Note we actually observed events ending with the "wrong" type: 2.1% (1.5%) of all A-events (W-events) converged within the prevalent time of two (six) minutes, but showed as last update an withdraw (announcement). We suspect that this is due to errors in the collection process.

The "good news" for the Internet is that 57% of the fast converging A-events consist of exactly one announcement. Yet the maximum number of BGP update messages is 659 during the time of one event. Even more intriguing is that 1,615 (0.6%) of the prevalent A-events show withdraws (up to 3)!

The median number of updates for W-events is 3 (triple of the median of A-events). The maximum is even 825 updates in one W-event. Only 22.7% of the prevalent W-events carry **no** announcement, i.e., do not show path exploration[3].

[3]From all 2,350,428 beacon updates, only 511,497 are withdrawals, i.e., 1 update out of 5 is a withdrawal.

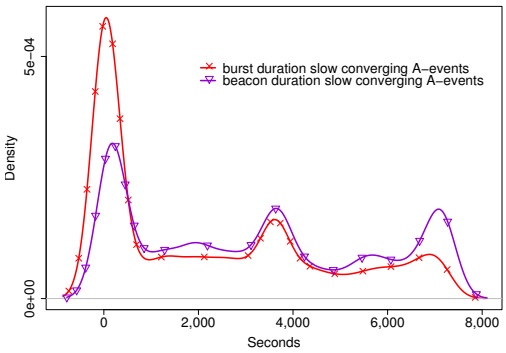


Figure 4.5: Burst duration and beacon duration of slow converging A-events. [102]

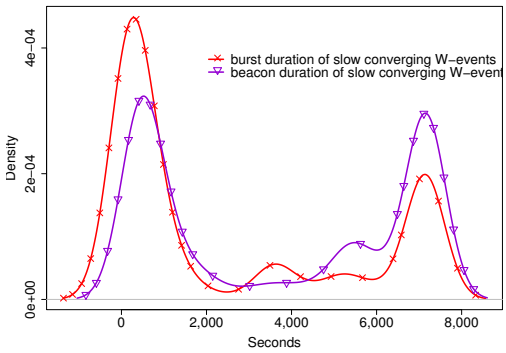


Figure 4.6: Burst duration and beacon duration of slow converging W-events. [102]

4.6.2 Slow convergence events

In this section we study the "remaining" 28,820 (4.7%) events, which show long convergence times. Note that this includes possible unrelated events occurring in the Internet, for example a link failure between the beacon prefix and the observation point could show updates at the observation point that are not due to slow convergence triggered by the beacon event. As this is not distinguishable from a slow convergence triggered by the beacon, we look at slow convergence beacon statistics in general. The median number of updates within such a slow converging A-event is 3, while the maximum is 1,575. Note that 29.7% of the those events actually contain withdrawals! The median number of updates within a long converging W-event is 5, and the maximum is 241. Only 8.8% of those events do not contain any announcement.

Figure 4.5 shows the smoothed density of the duration of the slow converging A-events, and Figure 4.6 shows the same densities for the long W-events. Note that both plots show the beacon duration[4] as well as the burst duration[5] to be able too compare the differences. This is because in the global data we lack the information of the time of the triggering events, thus we can only measure burst durations. Yet the beacons can provide a rough approximation about how large the error can be (see [136] for additional information).

Figure 4.7 shows the density of interarrival times within slow converging events. The median interarrival time of updates in slow converging A-events is 28s, and in W-events is 26s. Both median values are in the order of the MRAI timer. Yet the mean is 484.1s for slow converging A-events, and for W-events it is 365.5s. We have marked values at 600, 1,800 and 3,600 seconds (see shaded vertical lines), which represent recommendations by RIPE-229 [44] for route-flap damping. Thus our suspicion is that those peaks correlate with route flap damping parameters. Yet not all peaks can be easily explained.

Still we are interested in looking deeper in this mystery of slow BGP convergence. Rather intriguing is that 53.4% of all slow converging events show only one update burst within the first 500 seconds after the beacon event. Furthermore, even 27.8% of all beacon events show only one burst that duration is at most two (six) minutes for A-events (W-events). This means

[4] Recall from Section 4.4 (page 54), that the beacon duration is the time from the known triggering beacon event to the last recorded update in that event.

[5] The burst duration is the time between the first and the last update in a burst. All updates within the two-hour interval are treated as one update burst.

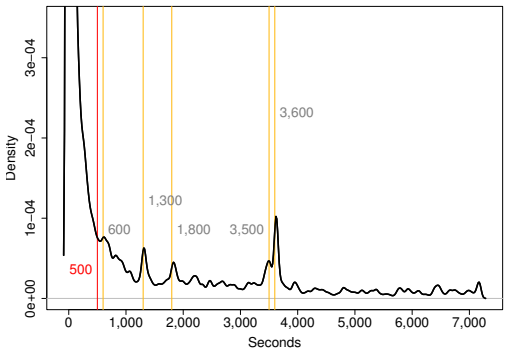


Figure 4.7: Interarrival times between updates in slow converging events. [102]

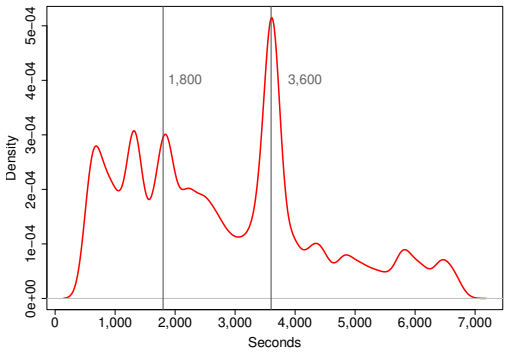


Figure 4.8: Interarrival times between bursts in slow converging events. [102]

that the event would have been classified as prevalent ("fast converging") but the onset from the triggering beacon event put it in the categories of slow converging events. In addition, another 22.96% show exactly two fast converging bursts during one beacon event. As an example consider route flap damping where a few updates are observable before the prefix is dampened and after the suppression is released another convergence process is triggered (by the router that dampened that prefix).

To understand the characteristics we look at the interarrival times between the bursts. Figure 4.8 shows a smoothed density plot of interarrival times between update bursts. The two vertical lines, at $1,800$ seconds (30 min) and at $3,600$ seconds (an hour) mark again the damping parameters recommended by RIPE-229 [44] and the maximum suppression time by most vendors. Thus this plot affirms that slow convergence is in parts caused by ill-applied route-flap damping [46].

4.7 BGP dynamics

The purpose of this section is to characterize BGP dynamics. Accordingly we start with looking at the convergence process. For this we use the notion of update bursts. Then we move onward and take a deeper look at the impact of the MRAI and route flap damping on the overall convergence behavior of the Internet and we conclude with some statistics about different attribute properties observable different peering sessions.

Our motivation is that each update burst should summarize all updates caused by one or multiple instability events and therefore capture the BGP convergence process. We are interested in understanding the characteristics of update bursts such as duration, number of updates and arrival process. Correspondingly Figures 4.9, 4.10 show the density of the logarithm of the duration and number of updates of update bursts. Based on the results by Mao et al. [45], we use a timeout value of a bit larger than one hour ($4,000$s). While a specific timeout value changes the curves, we found that the general characteristics do not change.

Note that Figure 4.9 shows only bursts that have at least two updates (otherwise the duration is 0). About 33.8% of all bursts consist of only one update in the burst and therefore are removed from the density computation. Remarkable is that the effects of the MRAI and multiples of the MRAI are clearly predominant in update bursts. The curve increases between

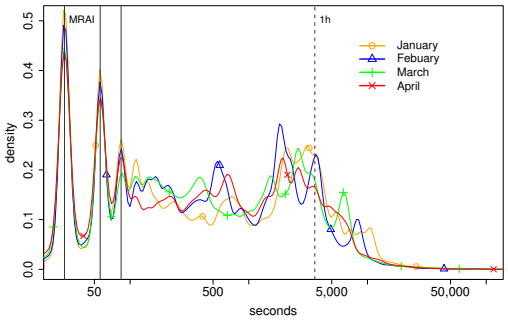


Figure 4.9: Duration of bursts.

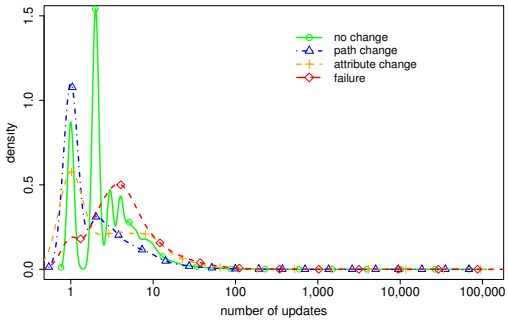


Figure 4.10: Number of updates in burst.

30 min, 1 hour and flattens at about 3 hours. This are quite likely the effects of route flap damping. Still update bursts can be quite long. We find that there are most of the time some prefixes that are flapping for serval days, weeks and even sometimes over many month. We identify in the data a set of two prefixes (operated by different ASes) that were flapping over four month with more than one update approximately every hour. (Note that the duration of the trace analysis was four month, the duration of the divergence could actually be much longer). Such events are not uncommon in context of issues such as MED oscillation [76, 124, 134]. Yet we recommend to use our methodology to identify such prefixes and notify the ISPs. A flapping prefix can degrade performance and pollutes routers in terms of CPU utilization and network bandwidth, still it is observable in the traces that oscillating prefixes remain undetected for quite a long time.

To understand the types of changes we compare updates from the previous stable state and the new stable state. To study how these instability events are related we distinguish the following categories: “no change”, “path change”, “attribute change”, “failure”. “*No change*” is predominant (77.8%) and means the last update in the burst has exactly the same set of attributes as the update in the previous stable route. This can be for example due to a failure somewhere on the path to the prefix, the failure is repaired within the timeout window and converges back to the old best path. The second category is “*path change*”. This contributes in 13.3% to the result and means that the AS path is not the same when comparing old and new stable routes. If the AS paths remains the same but any other attributes change, e.g., community values, then we say it is an “*attribute change*”. This happens in 4.1% of the cases. Finally if the old or new stable path is a withdraw while the other is an announcement we say this is a “*failure/repair*”-event. This contributes to the overall in 4.8%.

Figure 4.10 shows a smoothed density distribution of the number of updates in a burst on a logarithmic scale. While most bursts consists only of one or two updates up to 10 updates is not unusual. Still a burst can consists of quite a number of updates. The distribution is consistent with a heavy-tailed distribution [82]. We also find a correlation between the duration of a burst and the number of updates (correlation > 0.92). Furthermore, it is interesting that withdraw events (failure category) shows typically more updates in a burst, than bursts changing attributes or the AS path. This also underlines the path exploration observation before the actual withdraw is recorded. The fact that a lot of consecutive update bursts end with the same update indicates that most prefixes, even if they experience an instability event, converge to a main route. We observe that instabilities last only for a short time period and are contained within an update burst, which supports that our methodology is useful for finding instability events.

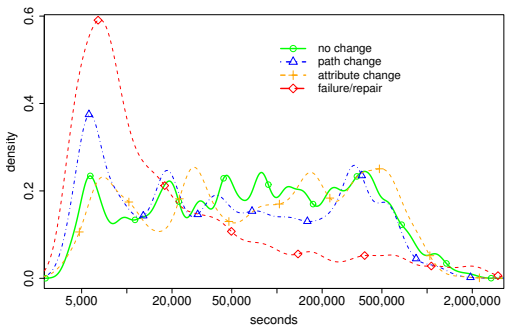


Figure 4.11: Burst interarrival time with kind of change.

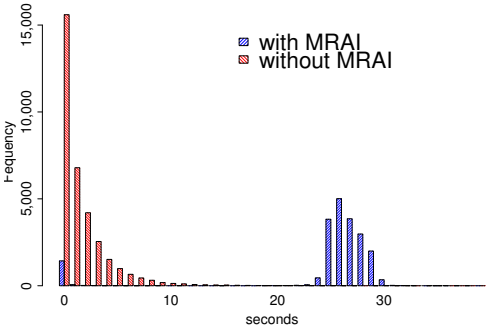


Figure 4.12: Interarrival times of between echoes.

Still, some failures cannot be repaired within 1 hour (burst timeout) and therefore involve two update bursts (captured by failure/repair). This is shown in Figure 4.11. This plots shows the interarrival times of bursts, i.e., when the next burst starts? This plot again distinguishes the categories “no change”, “path change”, “attribute change” and “failure/repair”. The red dashed curve (failure) answers the question above: If a failure cannot be repaired within 1 hour (which would put the burst in the “no change”-category), it is typically repaired within 3 to 6 hours. This can explain the high peak at the left side of Figure 4.11. The remaining curves actually do not show predominant peaks, and thus support the assumption that after an instability ends it is more or less random until the next instability affects the prefix.

Next we take a closer look at two factors contributing to long convergence times: the effects of MRAI and the impact of route flap damping.

Figure 4.12 shows a histogram of interarrival times of updates over a peering session. This means while we usually compute interarrival times on a per prefix **and** per peer basis, we solely consider here consecutive update packets arriving over a peering session. We distinguish peers that have an active MRAI and compare them to peers that do not show the effects of an MRAI timer. Figure 4.12 clearly shows that jittered behavior of 8 sessions with MRAI timer. Yet, only 3 sessions in that plot have no MRAI timer. The updates arrive over the session nearly immediately and the number of messages is much higher. This is a typical behavior because the MRAI collects updates coming from various internal neighbors, while the router without MRAI passes nearly every change in the best path selection process on to the eBGP neighbor. See [11] for related work.

Let us now investigate the effects of route flap damping in a thought experiment. We are interested in understanding the delay induced by a router, that is doing route flap damping and how this relates to the “echoes” that are typical for distance vector protocols. We approach this question from a practical viewpoint, by taking a router, a Cisco GSR 12008 [54]. We activate Cisco’s default route flap damping (Half-life time: 15 mins, decay time: 2320 secs, max suppress penalty: 12,000, max suppress time: 60 mins, suppress penalty: 2,000, reuse penalty: 750). In a test-lab we send update bursts to the router. The update bursts consists of a different number of updates (up to 40). Furthermore, the update probes are differently spaced (i.e., 1 sec, 10 sec, 30 sec, 1 min, 2 min). Figure 4.13 shows the results of this experiment. On the x axis is the number of probes in the update bursts. The y axis shows the times outgoing BGP updates are observed. The different points (see legend) represent the various spacings. A burst is targeted towards the router until all updates are sent, they are sent in the pace of the “echo spacing”. So, consider for example a burst of only 6 updates

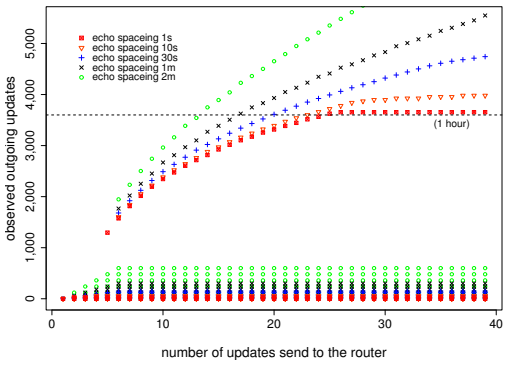


Figure 4.13: Durations of Route Flap Damping.

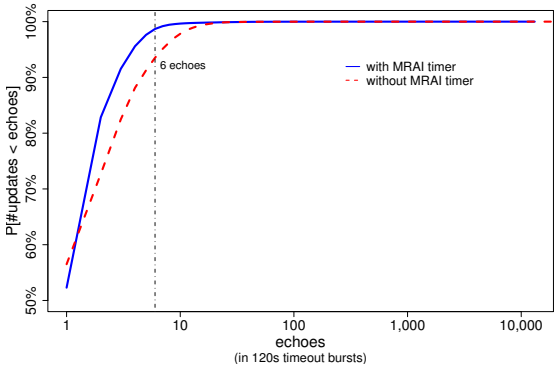


Figure 4.14: Number of echoes in a burst.

where all updates are spaced 30 seconds apart (illustrated in Figure 4.13 by the blue crosses at value 6). This means the experiment sending the updates to the router lasts 3 minutes (6 echos x 30 seconds), yet updates are coming out of the router for 1,680 seconds (28 min!). The prefix was suppressed (= withdrawn) for over 25 minutes. Note that damping can be triggered already after the fourth updates, yet if updates are not very closely spaced then the decay does not cause damping immediately.[6]

Now lets continue our thought experiment: We are interested how often this happens in reality? And if it matters if there is a router with MRAI or without MRAI timer on the other side? [7] We analyze the recorded data from the remote collector RRC00 as if the collector were an actual router with Cisco's default route flap damping enabled. We group updates into bursts with a timeout of two minutes. This provides us with groups of updates that are at most two minutes spaced. Note that this is a lower bound approximation for damping effects, as updates can be spaced even closer and therefore trigger damping even faster as well as bursts with a slightly larger spacings are not evaluated (because they are separated by the burst timeout).

Figure 4.14 shows a CDF of number of updates in a burst (burst timeout two minutes). This means we show on the y axis the probability that the number of updates in a burst is less than the value on the logarithmically scaled x axis. Again, we distinguish peers with MRAI (solid blue curve) and peers without (dashed red curve). Note that the distribution is heavy-tailed, which means there are a few prefixes that are subjected to a lot of updates. This can be due to oscillations, yet most prefixes are "well-behaved". The vertical line at 6 represents the number of echoes that surely triggers route suppression on a router with Cisco's defaults. This means that 8.3% (2.4%) of the prefixes that observed updates would get dampened on a BGP session where the remote router does not (does) use a MRAI timer. Note, that even though the number of well behaved prefixes that are subjected to damping is larger without MRAI, still the MRAI timer does not prevent suppression of prefixes that are in the normal convergence process.

Finally, we illustrate update propagation and filters by comparing the properties of two different BGP peering sessions. Figures 4.15, 4.16 show the **stacked relative distribution of updates over time** for two peerings sessions from the data set of 2002. While we observe that

[6] For example 4 updates spaced 1 second apart triggers damping already after the fourth updates – if the updates are spaced 2 minutes apart, then 6 updates are "needed" to trigger damping.

[7] Recall, Cisco has per default a MRAI timer of jittered 25-30 seconds. Juniper's out-delay is disabled by default.

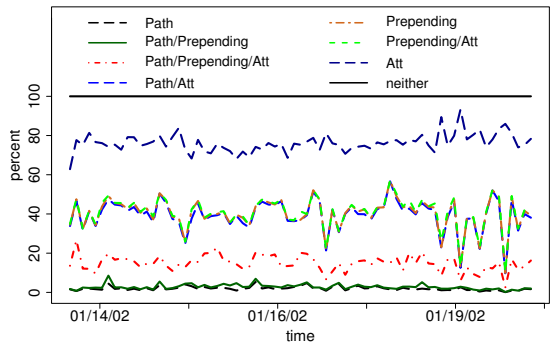


Figure 4.15: Relative # of updates.

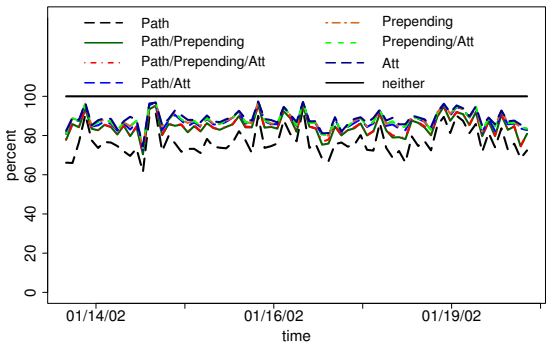


Figure 4.16: Relative # of updates.

for some peers, most updates involve mainly AS path changes, for other peers, combinations of path changes and attribute changes explain most updates. This depends on the policy of the peer. If the peer announces communities or other attributes, that shows different patterns on the BGP session, as in the case of the peer in Figure 4.15 (approximately $\frac{1}{4}$ of all updates are just changing community values). We will revisit this in a more generalized context in Chapter 7, when we build a workload generator for BGP that reassembles the observable properties of BGP on a session.

4.8 Summary

In summary we find, that BGP convergence in the Internet happens within two minutes. For failure events prevalent times are a bit longer, in the order of four to six minutes. Yet oscillating and diverging prefixes can be observed regularly.

A large factor to slow Internet convergence is the MRAI timer and route flap damping. We find, as well as other researchers [45, 46], that even "well-behaved" prefixes are subjected to route flap damping. This is because of too tight parameter settings that ignore the fact that BGP may send multiple updates for one event. While the distribution of the durations of the update bursts is consistent with heavy-tailed distributions the distribution of the interarrival times is not. This shows that our simple clustering technique is capable of grouping updates belonging to one event or related events (such as a failure and the corresponding repair of the failure) together and is able to identify stable states.

Working with BGP updates can sometimes be perplexing as a lot of policy decisions interplay with unfortunate administrative settings, vendor bugs and a still not well understood behavior of BGP dynamics.

5 Locating Internet Routing Instabilities

In the last chapter we looked at BGP dynamics and studied the convergence properties. This raises the question of the origins of all these updates (i.e., the location of the event that triggered updates), how they are correlated and if this can be inferred just by observing the control plane of the Internet. In this chapter we propose a methodology for locating the origins of routing instabilities. Our approach is to correlate information from updates across observation points (views) and across prefixes. In contrast to others [92–94] we propose to first correlate across time, then views, and finally prefixes. Each instability (any change of BGP advertisement over an eBGP session) implies that some BGP attribute changes for some prefixes are propagated via BGP updates throughout the autonomous system (AS) topology. *Our main insight is that if there is an AS path change, then some instability has to have occurred on one of two AS paths: the previous best path or the new best path.* Furthermore, if there is only an attribute change, then the instability has to be on the AS path. Using multiple vantage points, one can then pinpoint an instability. Under certain conditions (see Section 5.1.3) this instability corresponds to the original cause of the routing change.

We verify our approach in a novel way. In particular, we use a simulator (Section 5.4) which is based on an AS topology derived from actual BGP updates, and which uses BGP policies that are compatible with the inferred peering/customer/upstream relationships among the ASes. In the simulation, network and protocol behavior are ideal. Through simulation we learn what inference quality is achievable and how it is correlated with the number of observation points and the location of the observation points.

We then apply our methodology and evaluation technique to the same actual BGP updates gathered at more than $1,100$ observation points located in more than 650 ASes including the ones from RIPE RIS [48], University of Oregon RouteView [49], and more than 700 feeds are from Akamai Technologies (Section 5.3). To cope with the complexity of BGP we develop several heuristics (Section 5.2) to deal with the limitations of real BGP updates, such as update propagation, AS path exploration, MRAI timer, route-flap damping, absent updates, multiple instability events, as well as missing information. Overall we find (Sections 5.4 and 5.5) that we can pinpoint a likely origin of instability to a single AS or a session between two ASes in most cases even without correlating across prefixes. For further validation, we correlated the inferred instability with router syslog data from a tier-1 ISP. We are able to confirm that 75% of the inferences where this ISP is identified to be responsible for originating an instability coincides with a BGP session reset.

To summarize our contributions: We present a methodology for identifying origin of instability visible in BGP routing changes along three dimensions: *time*, *view*, and *prefixes*. Our approach is thorough, as we take into consideration complex BGP operational issues, but mainly it is simple and intuitive. It is based on how BGP path selection operates – some routing instability lies on either the previous or the changed stable paths. To show improvement over previous work, we also illustrate through detailed examples that simplified assumptions

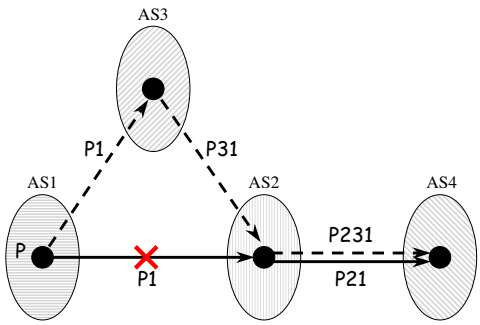


Figure 5.1: Example AS topology.

do not hold in practice. Furthermore, a main distinction of our work is that we use simulation as a validation methodology on an accurate, fairly complete AS topology to understand when we can and cannot narrow down the instability origin and the effect of vantage points on the inference. Finally, we apply our inference methodology on a large set of BGP data from diverse vantage points.

5.1 Ideal methodology

The goal of this section is to propose a methodology for inferring the origin of routing instabilities from their effects – the results of the BGP convergence process. Each instability may cause BGP updates to propagate through the Internet which can be observed at various monitoring points throughout the network. We use these updates to identify the instability origins. Moreover, the methodology is applicable to any other path vector routing protocols.

We refer to the location of a routing instability, either internal to an AS or between two ASes with BGP peering session(s), as an *instability origin*, and refer to the resulting sequence of BGP update messages as an *instability burst*. The specific instability burst observed at particular points on the Internet via monitoring sessions differs according to the location of the observation point (also referred to as view), the instability origin, the policies of the ASes along the AS path, the effects of timing imposed on the message ordering, and the AS topology itself. When the instability cause is due to an event internal to a given AS, excluding eBGP sessions to its neighbors, we say the cause is located in the given AS. When the cause of the instability is due to an event at an eBGP session between two given ASes, we say the cause is located at the edge between the two given ASes. (Note that in the latter case we do not try to determine which router at either end of the eBGP session initiated the event.)

5.1.1 Basic methodology

Let us consider the example AS topology in Figure 5.1 where AS1 is originating a route to prefix P. Assume the single link between AS1 and AS2 fails. In this case, the best BGP route at AS2 and AS4 changes from the solidly marked one to the dashed one. Given eBGP monitoring sessions to AS2, AS3 and AS4 (not shown in the figure), one will observe BGP updates at AS2, similar to the ones propagated to AS4, but none at AS3. The best path propagated by AS2 changes from P:21 to P:231. This is the kind of information that we take advantage of. In this case we can narrow the cause of the routing instability to AS2, or the

```
foreach instability event of prefix p
    foreach observation point o
        if route change with path change: from r_p to r_n
            r_b = best_path(r_p, r_n)
            candidate_set c_o = candidates(r_b)
        if route change without path change: r_p == r_n
            r_b = best_path(r_n)
            candidate_set c_o = candidates(r_b)
    instability candidates = ∩c_o
```

Figure 5.2: Per prefix – ideal methodology for locating instabilities.

edge between AS1 and AS2. The main idea is that when there is a change in the best BGP path, the origin of instability is either on the new path or on the old path or induces another instability on either of the two paths. Furthermore, the original or the induced instability must be on whichever of these two paths is "better" when compared head-to-head. To see this, notice that if the old path was better, then there would be no path change without instability on the old path. On the other hand, if the new path is better, then there must have been some instability on the new path. While it is not always obvious to an outside observer which of the two paths (old and new) is better, it is possible to derive a set of candidates for the instability by taking the union of the two paths. Alternative heuristics that are more aggressive are presented in Section 5.2. For example, here one may presume that the best path is the one with shorter path length: P:21. Similarly, the eBGP monitor at AS4 sees previous and new paths of P:421 and P:4321, and may presume that the best path is P:421.

Using information from multiple monitoring sessions helps narrow down the origin of the instability. Assume that the instability under consideration is the only instability during some time period. Then all path changes for the prefix P are due to this instability. This implies that the instability is visible at each observation point receiving BGP updates for P, which means that it is present in the intersection of the corresponding candidate sets. In the present example, the intersection from the eBGP monitoring sessions at AS2 and AS4 yields the candidate set of AS1, AS2, and the edge between AS1 and AS2. This basic approach is summarized in Figure 5.2.

This ideal methodology assumes the following:

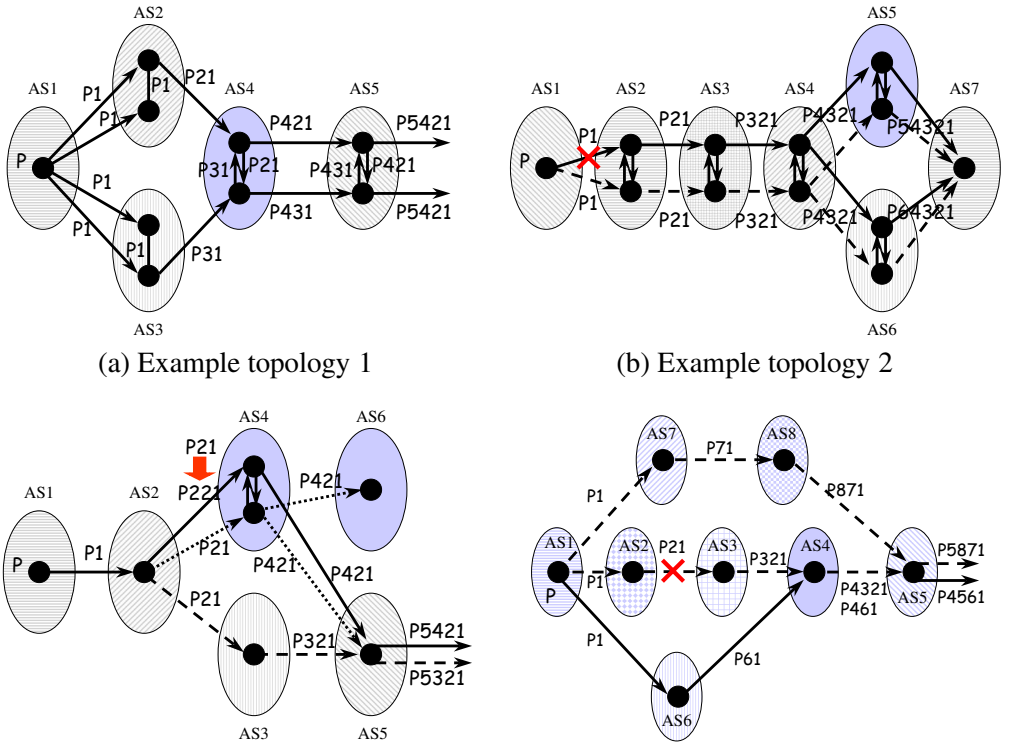
1. All updates caused by an instability event are identifiable.
2. At any time each prefix is only hit by one instability event.
3. BGP convergence finishes within some time period.
4. We can determine which paths are stable.
5. We can determine which of two BGP paths is better.
6. There are no *induced* instabilities (see Section 5.1.2).

While any of the above may not apply with actual BGP update data, it is possible to develop heuristics to deal with each violation of these assumptions as described in Section 5.2. Furthermore, it is possible to evaluate the methodology using simulations, see Section 5.4. This enables us to calibrate our expectations.

5.1.2 Cautions

Next we illustrate using simple examples why the details and the assumptions matter when trying to locate routing instabilities[1].

[1] For additional discussion see [35].



(a) Example topology 1

(b) Example topology 2

(c) Example topology 3

(d) Example topology 4

Figure 5.3: Example AS topologies

Caution on excluding candidate ASes: Suppose that, at a given observation point and for a given prefix P one sees previous and new stable paths of P:7,6,5,4,3,2,1 and P:7,6,5,8,3,2,1 respectively. One might think that AS 7 or 6, or 2 or 1 could not be the cause of the routing change, and thus these shared segments of the two paths could be excluded from the candidate set. However, such inference can be erroneous and is not made in the "ideal methodology". The following two examples show that if the previously best and the new best path share segments, it can be important to include the shared segments in the candidate set. This is where our methodology differs from the approach by Chang et al. [93] which can incorrectly exclude some instability originators.

Consider the example shown in Figure 5.3(a). Customer AS1 is multi-homed to two providers, AS2 and AS3. Both providers, AS2 and AS3, have the same upstream provider AS4. And AS4 peers with AS5 at two peering points. AS4 learns of the given prefix from AS2, and may propagate the path P:421 to AS5 on the top of the two peering sessions. Likewise, AS4 also learns of the given prefix from AS3, and may propagate the path P:431 to AS5 on the second of the two peering sessions, the bottom one. With cold potato routing, AS5 chooses to announce the single route P:5421 to other ASes, including an eBGP session with an observation point (not shown in the figure). Now, due to, say, an internal failure within AS5 or an operator in AS5 intentionally changing IGP costs, the route announced by AS5 to other ASes changes from P:5421 to P:5431. Thus AS5 is the instability originator even though the AS path change is at a different location, i.e., from 421 to 431. Thus the tie-breakers cause situations in which changes within a remote AS can lead to AS path changes in the initial (closer to the origin AS) segment of the AS path.

Furthermore, consider the example shown in Figure 5.3(b). Here the customer, AS1, is again multi-homed but to only a single provider and originates prefix P. AS2, as well as AS3, and AS4 all use cold potato routing in the sense that they use the IGP metrics to initialize the MED values within the BGP updates. In this case it might well be that AS5 uses a route which is propagated along the solidly marked path while AS6 is using the dashed one. Lastly, based on received MED values, next hop IP's and IGP costs, AS7 chooses to announce path P:754321 on an eBGP session to some observation point (not shown in the figure). Now assume that the solid link between AS1 and AS2 fails. In this case the next hop of the BGP update from AS2 to AS3 together with the MED value will change. This causes all routers within AS3 to change their best path to the dashed path. This implies that the BGP update from AS3 to AS4 will have a different next hop and a different MED value. This will cause AS5 and AS6 to announce new next hop and different MED values to AS7. Since the next hop and MED values received by AS7 have changed, AS7 may change its preference from AS5 to AS6, and announce a new path of P:764321 to the observation point. Thus, a link failure in or between some AS near the origin AS, i.e., AS1 to AS2, can cause an AS path change at a subsequent location on the path, i.e., from P:754 to P:764. If one imagines a slightly more complex internal topology, even changes to IGP metrics within an AS can have such an effect. Using IGP metrics as MED values creates a link between internal changes and external effects and therefore between distant ASes. Similar effects are possible using communities.

Caution on instability propagation: Figure 5.3(c) shows the danger of assuming that all instabilities are propagated. In this specific case, AS6 uses the dotted route to prefix P, P:6421, while AS5 uses the solid one P:5421. Now suppose that AS2 does AS path prepending on one of the eBGP sessions with AS4, and that this causes AS4 to switch its best route for P to the dotted one. This change has no impact on AS6 since its route does not change. AS5 will receive updates since the IGP/MED values within AS4 changed. This may cause AS5 to switch to the dashed path via AS3, P:5321. Hence we have a situation where the best path of AS6, in the sense of AS-level path P:6421, has an instability, but AS6 will not receive a corresponding BGP update. This can be achieved via IGP/MED coupling and filters, e.g., using communities. In essence this problem corresponds to the previous problem.

Caution regarding induced updates: Figure 5.3(d) shows the danger of assuming that the origin of all instabilities is either on the new or on the old path. In this specific case, AS4 prefers the route P:321 for prefix P instead of the route P:61. Accordingly it advertises the route P:4321 to AS5, and AS5 advertises P:5871 to an observer. If the link between AS2 and AS3 goes down, AS4 revises its advertisement to AS5 to the route P:461. If now AS5 prefers the route P:461 over the route P:871 it will advertise the route P:5461 to an observer. Thus the observer sees the route to P change from path P:5871 to P:5461, even though the original failure is the link between AS2 and AS3. In this case the original failure *induced* or triggered a route change at AS5. While the routing decision at AS4 may seem unorthodox, it is nevertheless coherent in the sense that AS4 uses a consistent ranking of the paths. Induced updates can occur if the ranking of routes differs between providers. Our methodology is capable of locating the AS where the route change is induced, but may not be able to locate the original cause of the instability. On the one hand this is disappointing, yet on the other hand locating the induced instability already reduces the problem and is valuable in itself. The problem introduced by induced updates is that the intersections can be empty, if a subset of the observers point towards the original instability and another subset to an induced update, or

even incorrect, e.g., if A is the instability origin AS, B the AS at which an update is induced, and A, C is the subset that one subset of the observer identifies, and B, C is identified by another subset. Nevertheless it is the case that each set of observers can be partitioned in such a way that the intersection of the union of the AS paths will include either one AS at which an update is induced or the original instability.

5.1.3 Identifying link changes

The cautionary examples highlight that the union and the intersection rules are only heuristics. Yet, these are sensible heuristics and we now provide some formal justification. In particular, we analyze the effectiveness of the union heuristic in the simplified model of BGP that is realized by our simulator. Each of the theorems in this section relies on one or more of the following assumptions.

Assumption 5.1.1. *The simplified BGP model assumes:*

a) The only events in the network are link failures and link restorations, and the network fully converges to new routes between successive events.
b) Routes are chosen based on the AS Path attribute only. Other attributes such as MED and next hop are not considered in calculating local preferences. There is at most one peering session between any pair of ASes.
c) For each destination and for each AS, routes are chosen based on a total order over all possible AS paths to the destination. Although the list of paths available to an AS may change over time, the total order over all possible paths never changes.

The following theorems relate to the union rule. Suppose that an event has occurred, and that as a result, the AS path from an observer (AS O) to a destination (AS D) has changed.

Theorem 5.1.2. *Suppose Assumptions 5.1.1 hold. If observer O sees its path to destination D change, then on either the old path or the new path, at least one AS changes the advertisement for D that it sends to its predecessor on the path.*

Proof: If no AS on either the old or new paths changes its advertisement, then both paths were already available to O, and remain available. Since, by Assumption 5.1.1(c), paths are chosen according to a fixed total ordering, the old path remains preferred over the new path. □

Observe that, by Theorem 5.1.2, on either the old path or the new path from O to D, there is a maximal prefix of ASes such that every AS on the path changed its advertisement to its predecessor on the path. By definition, the last AS on the prefix did not receive a new advertisement for D from its successor on this path. Call this last AS on the prefix Y, its successor Z, and its predecessor X.

Theorem 5.1.3. *Suppose Assumptions 5.1.1 hold. If Y is on the old path, then it either observed a link failure on the old path or received a new advertisement for a path to D from outside the old path. If Y is on the new path, then it either observed a link restoration on the new path, or it received an advertisement withdrawing a path to D from outside the new path.*

Proof: Suppose that X, Y, and Z lie on the old path, and Y changed its advertisement to its predecessor X on the old path. Since Y is receiving the same advertisement for D from its

successor Z on the old path, then either the link between X and Y failed (and hence Y could no longer advertise across it), or Y must have learned of a new path to D from outside the old path that it prefers over the old path. If, on the other hand, X, Y, and Z lie on the new path, then either the link between Y and Z was restored, or a path to D that Y prefers over the new path was withdrawn from outside the new path. □

Theorem 5.1.4. *Suppose Assumptions 5.1.1 hold. Consider the common prefix of the old and new paths from O to D. If Y appears on this prefix, it can only appear as the AS closest to D.*

Proof: The proof is by contradiction. If Y is on the common prefix with respect to the old path (but not the AS closest to D), then it must appear (in the same position) with respect to the new path, and vice versa. Since the link between Y and X has neither failed nor been restored (it appears on both the old and new paths), it must be that Z either learned of a new path from outside old path, or saw a path withdrawn from outside the new path. But both the old path and new path were available to Y before the event, and are still available to Y after the event, and (by Assumption 5.1.1(c)) no event observed by Y can change its preference of the old path over the new path. □

Theorem 5.1.5. *Suppose Assumptions 5.1.1 hold. Consider the common suffix of the old and new paths from O to D. If Y lies on this suffix, then it must appear as the AS farthest from D.*

Proof: The proof is by contradiction. If Y appears on the common suffix but is not farthest from D, then Y appears in the same position with respect to both the old and new paths. On both paths, Y receives the same advertisement from its successor Z, and sends the same advertisement to its predecessor X. This contradicts the definition of Y. □

5.1.4 Consideration of multiple prefixes

So far we have only considered two of the possible three dimensions for inferring the origin of routing instabilities: *time, views,* but not multiple *prefixes*. Since it is quite likely that multiple prefixes use the same BGP session/same link/same AS on their AS path, a failure to any of the latter will cause changes to multiple best paths. This implies that if a prefix is affected by only a single instability during some time, then we can identify correlated events. One approach is to use a **Greedy heuristic** which starts with a set $\mathscr{P}$ that includes all prefixes with instabilities during this time window. For all prefixes within the set $\mathscr{P}$, count how often each AS topology component appears across the instability candidate sets associated with these prefixes. Choose the most frequented AS topology component E as the most likely instability cause/origin, and subtract the prefixes from $\mathscr{P}$ that include E in their candidate set. The algorithm continues until $\mathscr{P}$ is empty which means that all prefixes have been assigned an instability "origin".

5.2 Adopted methodology

Beyond opening new questions for the general evaluation of BGP, the details examined in Chapter 4 impose certain adaptations of the proposed basic methodology as well as the introduction of additional heuristics. The final approach is outlined in Figure 5.4 and 5.5. The first reflects the necessary adaptations while the second corresponds in essence to the ideal methodology (see Figure 5.2 on page 65).

```
## preprocessing per observation point
## (condense updates per prefix)
foreach prefix p          ## condense updates per prefix
    foreach observation point o
        U = updates(o) − flapping(o)
        burst_set_p = update_burst(U, timeout)
        foreach b in burst_set
            r_p = as_path(old_stable_route(b))
            r_n = as_path(new_stable_route(b))
            r_b = best_path_set(r_p, r_n)
            candidate_set c_ob = candidates(r_b)
## identify event set E
## across observation points and prefixes p
foreach time-unit t and foreach prefix p
    E_p = E_p ∪ new_event(t);
foreach event e ∈ E_p
    event_burst_set_e = associate_event_bursts(burst_set_p, e)
## condense bursts to identify instability origins
foreach event e and foreach prefix p
    foreach observation point o
        foreach (burst b, o) in event_burst_set_e
            candidate_set c_o = ∪c_ob
    instability candidates = ∩c_o
```

Figure 5.4: Per prefix – adapted methodology for locating instabilities.

5.2.1 Candidate sets

Since instabilities can originate within an AS or between ASes our basic units are edges either between two ASes or within an AS. The **candidate set** of an AS path consists of an edge for each AS and an edge for each pair of consecutive entries on the path. Note that typically the path received at the monitoring point does not contain the AS in which the monitoring point resides, for brevity called the monitoring AS. However, should the path indeed contain the monitoring AS, then we can exclude that AS if care is taken to exclude all updates associated with session resets on the monitoring session. For example, the candidate set for the path 4321 where AS4 is the monitoring AS is: candidates(4321) $= \{(1,1), (1,2), (2,2), (2,3), (3,3)\}$.

Best path: In general (see Chapter 4), it is hard to determine which route is the better one. For updates with path changes, this corresponds to the problem of deciding which of the two AS paths is the better path. Accordingly, the **standard** heuristic uses the conservative approach of including the union of the edges from both AS paths. This yields, ignoring induced updates, a lower bound with respect to pinpointing instability origins. For example, if monitor AS4 sees updates 4321 and 421, this results in best_path_set(421, 4321) $= \{421, 4321\}$. Should one of the two stable routes be a withdrawal then the best_path_set is the AS path of the other. If both stable routes are withdrawn then we do not gain any information and the best_path_set consequently contains all possible paths. If the AS path does not change then the best_path_set corresponds to one of the two paths.

If one ignores the impact of local preference and if the AS paths are of different length then

```
## identify correlated events CE across prefixes
foreach time-unit t
        CE = CE ∪ new_correlated_event(t);
foreach correlated_event ce ∈ CE
        event_set_ce = associate_ce_events(ce)
## Greedy heuristic for clustering instabilities
foreach correlated event ce and event e ∈ event_set_ce
        𝒫 = ∪ prefix(e)
while (𝒫 != {})
        reset counts to 0
        foreach p ∈ 𝒫
                increase count(instability_candidates(event(p))
        i = instability with count(i) == max(counts)
        𝒫 = 𝒫 − {p with i ∈ instability_candidates(event(p))}
        print instability i with prefixes 𝒬
```

Figure 5.5: Across prefix – adapted methodology.

the better path is the shorter one. Accordingly, the **best path** heuristic only considers the shorter one, e.g., best_path_set(421, 4321) $= \{421\}$. (In the case of equal path length, the standard heuristic is used.)

Shared path segments: The example in Figure 5.3(b) shows that if the new and the old stable AS path have the same initial segments, it is in general not permissible to exclude it. Yet for an instability in the joined initial segment to cause the later AS path change, the instability has to be propagated through multiple ASes without path changes. This is possible via IGP/MED interactions or by specific filter combinations. Since experience shows that such combinations across multiple ASes are unlikely, one can expect that a heuristic, called **initial path**, which excludes the initial segments up to but not including the edges to the divergence point, does not do too badly. For example, with the initial path heuristic we have at AS7: best_path_set(754321, 764321) $= \{7543,7643\} - 3$. This notation means that we can exclude the edge $(3,3)$ from the candidate set. This heuristic is especially helpful when only non-transitive attribute changes occur, e.g., next hop, local preference, and originator changes. With regards to changes to transitive attributes, e.g., communities, more care may be necessary.

The example in Figure 5.3 shows that, if the new and the old stable AS paths have the same final segments, it is in general not possible to exclude it. The problem sketched there is due to peering at multiple connections which is common between larger providers. Therefore, applying a heuristic that excludes the final path segment, called **final path**, can be dangerous in the sense of excluding the cause of the instability. Nevertheless exploring it is of interest. For example with the final path heuristic we have at AS7: best_path_set(765321, 765421) $= \{65421,65421\} - 6$.

Summary: Note that the standard heuristic tries to never exclude the cause of an instability or the induced instability. The heuristics: best, initial, and final paths can exclude some ASes and AS pairs that might have caused an instability. On the other hand they provide the benefit of narrowing the candidate sets. Accordingly, we are interested in evaluating the benefits and dangers of using the heuristics, especially since related work [93] by default assume that the initial and the final path heuristic are applicable.

5.2.2 Events

So far we have adapted the ideal methodology of Figure 5.2 (page 65) to include specifics about how to determine old/new stable routes, route and path changes as well as best paths calculations (see Chapter 4, page 54). But we are missing a way to identify instability events.

Note that all non-local instabilities of a prefix, e.g., those that cause AS path changes, have to be propagated along a subgraph of the DAG of this prefix. This implies that an instability may be visible at multiple observation points within the DAG subgraph at about the same time. After excluding most effects due to path exploration, propagation delays, and BGP timers by computing per prefix update bursts, we now identify the start and end times of events and associate each burst with an event.

When identifying events we still have to deal with timing problems: within BGP, as well as with the propagation to the observation points, as well as with the "accurate" time synchronization of the monitors. To identify for each prefix the event start and end times together with the appropriate update bursts, those that started within this time window, the two heuristics **new_event** and **associate_event_burst** are used. One way of grouping bursts into events is similar to grouping updates into bursts or packets into flows using a relative timeout. The first event is initialized with the start time and the finish time of the earliest update burst across all observation points. If the time difference between the event finish time and the start time of the next update burst is less than the timeout value, the event end time is set to the end time of this next burst. This implies that the quiet period between the events has to be greater than the timeout value. This approach is referred to as **relative timeout**. A major drawback is that events can span a long time period and may therefore contain multiple actual events. An alternative is to use a **static timeout**. But now the sensitivity of the timeout becomes a problem. An appropriate value for one prefix may not be a good one for another. Furthermore, a static timeout may separate two related update burst into different events (see Section 4.2 on page 55).

Accordingly, we also pursue the third option of an **adaptive timeout**. It consists of two steps: During an initial period which starts with the beginning of the event and ends at start + timeout, a relative timeout of timeout$/2$ value is used. Then a relative timeout of 0 is used. The first part desensitizes the specific choice of timeout values and takes advantage of the nice properties of relative timeouts. The second part together with excluding continuously flapping updates ensures that events are not too long and that parallel bursts are associated with the same events.

5.2.3 Correlated events

Chapter 4 clearly outlines that BGP updates to multiple prefixes are often correlated. Accordingly, the next step is to correlate the located instability origins across multiple prefixes and identify clusters. The **Greedy heuristic** outlined in Figure 5.5 provides a simple approach, but it requires us to identify which prefixes experience correlated events. We solve this problem by grouping events per edge to correlated events in the same manner using the same heuristics as for identifying events. But to prevent long events from attracting all other events we put a limit on how much each event can extend the per edge clusters. The number of events in each edge clusters is used in the ranking of the Greedy heuristic.

5.3 Data sets

Our work relies on external BGP routing tables dumps and update traces obtained from RIPE [48], Routeviews [49], a local ISP, and Akamai Technology. Throughout this chapter we only present results in an exemplary fashion for the following raw data sets.

BGP update traces: from 12/04/03, 00:00 GMT to 12/16/03, 00:00 GMT, consisting of more than 343,600,000 updates from more than 1,100 different peering sessions to more than 650 ASes, including Tier 1 ISPs, major European ISPs, Asian ISPs, as well as stub ASes. Some ASes provide full feeds while others are partial feeds. We have multiple monitoring sessions to about 43.3% of the monitored ASes.

Basic statistics: Overall the number of observed prefixes is 276,556 of which 28,110 are from the private address space. The latter are excluded from further consideration. Of the remaining, included prefixes, 42.7% were, at some point in time on at least one observation point, subject to AS path prepending, which is a popular policy used for traffic engineering purposes. In terms of inconsistencies we found that 4,038 or 1.46% of the prefixes had multiple originating ASes. Also, 11,507 of the pairs of (observation points, prefixes) were continuously receiving updates in the sense that they have at no inter-update time larger than 2 hours for more than 1 day.

Inferred AS topology: We took one day of BGP table and update data on December 10, 2003 for the purpose of analyzing AS relationships and inferring AS paths for simulation purposes as described below. 3,428,464 distinct AS paths (after ignoring AS prepending) are used as input to the relationship inference algorithm. The graph consists of 16,757 nodes with 45,376 edges. Based on the relationship inference, we have 30,653 customer-provider relationships, and 1,532 pairs of ASes are found to have peering relationships.

5.4 What if – simulations

To understand the accuracy of our algorithm for inferring the location and the cause of routing instability, we validate via extensive simulations on the inferred AS topology. We make use of RouteScope [137] in inferring all valid policy paths between two ASes. RouteScope uses a simple algorithm based on shortest AS hop count for inferring AS paths between two end systems, without access to either host, by using information from BGP tables collected from multiple vantage points. Given the collection of AS paths from BGP tables, the AS relationship inference algorithm by Battista et al. [138] is used to identify all valid policy paths. Valid AS paths are assumed to go through paths in the form of *CustomerProvider* PeerPeer? ProviderCustomer** (denoted as *AS path rule*), where "*" represents zero or more occurrences of an AS edge and "?" represents zero or a single occurrence of an AS edge.

Based on the inferred AS relationships, edges in the AS graph are grouped into the following four categories: (i) custom-provider link (UP link), (ii) provider-custom link (DOWN link), (iii) peering links (FLAT link), and (iv) unknown AS relationship. For the last type, we replace the edge with one UP link and one DOWN link, effectively removing any restriction on the inclusion of edges with unknown AS relationships. We repeated our analysis with all such edges excluded from the AS graph. The results are very similar to what we report here. The accuracy of RouteScope in predicting AS paths from several selected ASes to the

entire Internet is around 85%. Inaccuracy stems from the following reasons: (1) Inaccuracy in AS relationship inference. (2) AS prepending effect is ignored. (3) Special routing policies for particular prefixes. We emphasize that such inaccuracy does not affect our evaluation methodology, as we aim to have a reasonably accurate AS topology to study whether our algorithm can precisely identify the location of simulated failures, given the AS paths selected before and after the failure.

5.4.1 Controlled experiments

We perform the following set of controlled experiments. RouteScope can infer a set of most preferred valid policy paths between any two AS pairs in the AS graph. Oftentimes, multiple AS paths appear to have the same preference, i.e., with the same AS path length and of the same type (customer, peer, or provider routes). To understand the effect of an arbitrary link failure, we randomly select a set of observation points and destination points by picking from tier-1 ISPs, tier-2 ISPs, ISPs with other ranks (based on ranking algorithm in [107]), and stub ASes based on a fixed proportion. We also attempt to include the observation points from which we have the BGP feeds as part of the source ASes.

Given the selection of source and destination AS nodes, we study the effect of a failure by computing the set of best AS paths before and after the failure. We remove the inter-AS link affected by the failure. In practice, there may be multiple peering links between two ASes, especially two large providers. We simplify this by assuming a single link and thus simulate the worst case scenario. We select 100 failures strategically by considering a variety of combinations of ASes in different parts of the Internet hierarchy, e.g., between two tier-1 ASes, a tier-1 AS and a stub AS, two tier-2 ASes, etc.. Given the set of equal cost paths between two ASes, we impose a selection among all equal cost paths to make sure the routing decision is consistent. For instance, if AS X selects a route advertised by AS Z, and AS Y chooses a route from AS X, Y's route must also go through Z.

To our surprise, just randomly selecting 100 destinations and observation points can make it hard to observe any changes in the best paths. Contrary to previous claims, in our simulations, we found that BGP failures are fairly well-isolated due to redundant paths, see Section 4.2. We plan to explore this further by understanding the properties of topologies where a particular failure between two ISPs of given ranks can affect.

5.4.2 Results

Given the results of the failure, we apply our heuristics (see Section 5.2) to the data sets to compute possible instability candidate sets. We note that none of the heuristics has ever excluded the failed edge from the resulting instability set indicating that the approach is sound. Indeed it is not surprising that none of the examples from Section 5.1.2 materialize since the simulation scenario mainly follows the assumptions except for best path. Preferring customer and peering relationships over upstream providers may cause the best path heuristic to fail.

To explore whether the location of the failure has any impact on our ability to locate it, we divide the failed links into three classes: "top tier" (between tier-1's and tier-2's or between a tier-1 and a tier-3), "middle tier" (between tier-2's, tier-3's or tier-4's), "bottom tier" (all

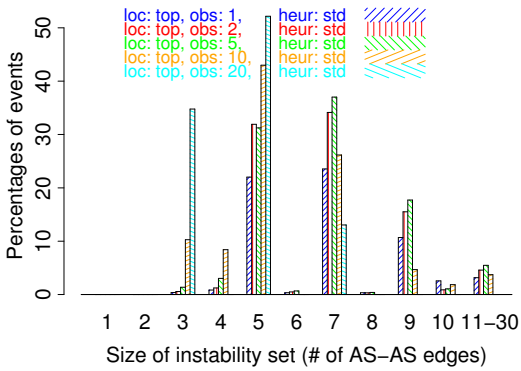


Figure 5.6: Simulation: instability set size hist. for # of obs. (heur.: standard, loc.: top).

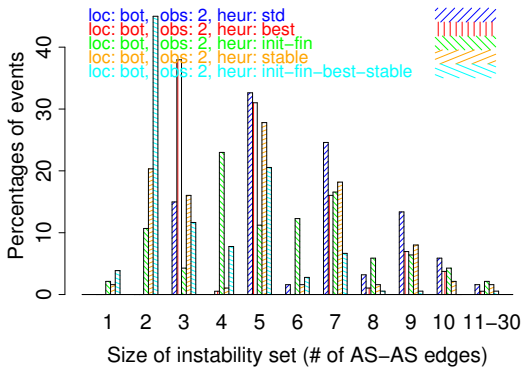


Figure 5.7: Simulation: instability set size hist. for various heuristics (obs: 2).

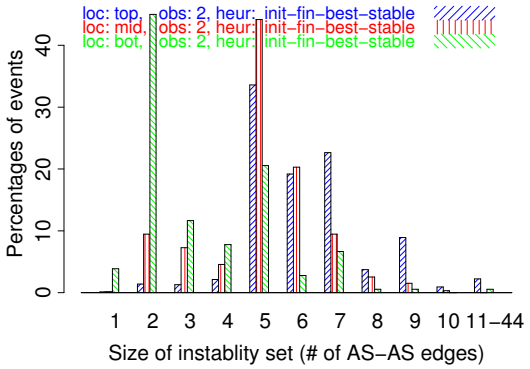


Figure 5.8: Simulation: instability set size hist. for failure locations (heur.: all, obs.: 2).

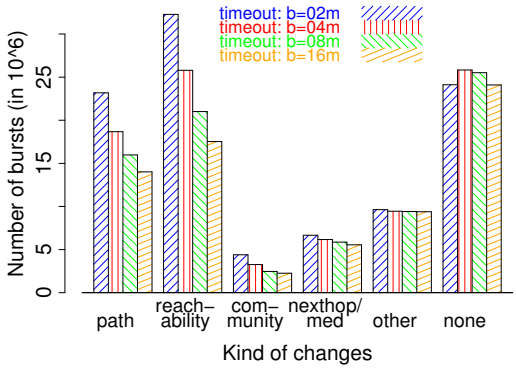


Figure 5.9: Stable route differences for various timeouts.

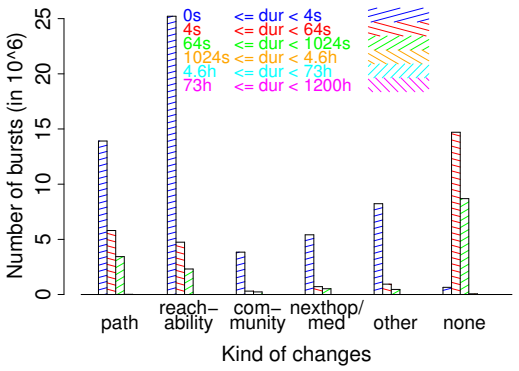


Figure 5.10: Stable route differences for burst length with 2 minute timeouts.

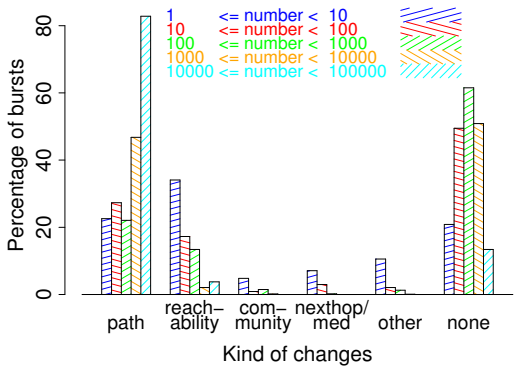


Figure 5.11: Stable route differences by # of updates in burst for 2 minute timeouts.

others). Unfortunately the observation that random selection of points makes it hard to observe any changes also implies that the number of observation points that observe any best path changes is limited. This is extreme for the bottom class where of the 9,112 events, about 15% are observed at multiple observation points. For the middle tier, this value is close to 40% and for the top tier 69%. While this might limit the potential gains of our methodology, Figure 5.6 shows that it is possible to derive instability set sizes with the conservative standard heuristic of $5-7$ for more than 68% of the failures with only two observation points. With 10 observation points this increases to almost 88% and almost 100% with further observation points. This implies that it is easier to track those failures that are percolating through larger parts of the Internet.

Narrowing the instability set size to three is the best one could hope for since the candidate is at a peering link which causes the instability set to already contain three edges. This set can only be further reduced by using the other heuristics. Figure 5.7 shows this benefits for the bottom class. The best path heuristic helps to decrease the instability set size from two inter-AS links (= 5 AS-AS edges) to one (= 3 AS-AS edges) for more than 20% of the events. The stable, the initial, and final heuristics exclude more edges so that even the precise instability origin can be determined. Overall the combination of all heuristics reduces the instability set to less than 5 AS-AS edges including intra AS edges for more than 88% of the cases. Figure 5.8 shows that using our heuristics with five observation points, one can pinpoint the origin for more than 80% of the events down to less than 5 edges for bottom tier, 6 edges for middle tier and 7 edges for the top tier. Overall this indicates that the huge redundancy within the core of the Internet adds additional complexity to pinpointing the origins. On the positive side these edges are used by many different prefixes so that the Greedy heuristic will capture the appropriate edges, which it indeed does for these failures.

5.5 What is – data analysis

Having shown that our methodology is sound in the "what if" world, we can now apply it to the "real world". For this we need to determine how sensitive the results are to the heuristics as well as their parameters. Furthermore, each step provides us with useful information about how far BGP updates spread and their impact radius. For the purpose of the evaluation, we partition the approach into the following four stages: (1) update burst calculation, (2) grouping of bursts to events (3) instability candidate calculation for each event, and (4) correlation of events.

5.5.1 Update bursts

The update burst calculation is used to identify prior and post stable routes and one of the more interesting questions is how do these stable routes differ. We classify the changes into the following groups: *path* summarizes all updates that have changes in their AS path, *reachability* counts those prefix bursts where a route either became available or was withdrawn, *community* sums up those without path and reachability changes but with a change in their community attribute, *nexthop/med* includes those with either nexthop or med but no AS path or community change, *other* captures all other attribute changes not within any of the previous classes, while *none* pools those where all attribute values of the two stable routes remain

equal. Note that if, e.g., the AS path and the MED changes then the burst is counted as a path change.

Figure 5.9 shows a histogram of the number of bursts per group for different timeout choices. While more than 23% of the 2 minute bursts include a path change, more than 24% do not result in a change, even though within the burst there was a change. Furthermore, a significant fraction only propagates attribute changes that should mainly have local impact. On the other hand, about 5% of the bursts with pure community changes have to be propagated through the full reachability graph unless some providers filter such community values.

We decided to study timeouts ranging from 2 to 16 minutes. The smallest value, 2 minutes, can be sufficient to group updates caused by path exploration together into one burst. Section 4.6 (page 55) shows that most beacon announcements converge within a two-minute window. Yet since there are various ways in which BGP updates can be delayed, including MRAI timer, route reflectors, etc., not all path exploration can be captured with timeout values of 2 minutes. Accordingly, we study larger timeouts of 4, 8 and, 16 minutes as well. The problem with large timeouts is that they can group the results of several instabilities together, e.g., an instability together with the instability repair, which may correspond to combining bursts with smaller timeouts of group path or group reachability to one of group none. The decrease in the absolute number of bursts as well as the above average decreases in the path and reachability groups is apparent in Figure 5.9. The larger timeout values are especially problematic since they are larger than the delays imposed by route flap damping. Furthermore, they limit our ability to pinpoint the exact time of the instability which is needed for the next steps for determining the origin of the instability.

In terms of the duration of the instabilities, Figure 5.10 shows a histogram of the number of bursts per group for a timeout choice of 2 minutes, by duration of the burst. That a large fraction of the bursts lasts less than 64 seconds, independent of the timeout value, is an indication that most timeout values are reasonable. On the other hand we note that the longer a burst lasts the more likely it is to recover its old stable route. This is (not shown) even more dominant for larger timeout values. The bursts in the community, nexthop/med, and other groups tend to be significantly shorter than others. This may be an indication that these bursts are not the result of a path exploration. As the duration of a burst is somewhat correlated with the number of updates in a burst, it can be expected that most bursts contain a fairly small number of updates. Figure 5.11 shows the result of first grouping bursts according to the number of updates that they contain and then computing the relative distribution across the groups. Most of these involve a change in either reachability or in the AS path. The fact that most bursts in the community, nexthop/med, and other group are dominated by small bursts is another indication that no extensive path exploration takes place. Yet, longer timeout values cause the numbers of updates within bursts to increase and these are likely to be in group none. Since we want to pinpoint instabilities in time and since bursts of group none have lost most of their information about the location of the instability, we proceed with update bursts of 2 and 4 minutes.

5.5.2 Events

This stage associates bursts from various observation points with events using various timeout heuristics: relative, static, and adaptive timeout. With regard to choosing parameters, the

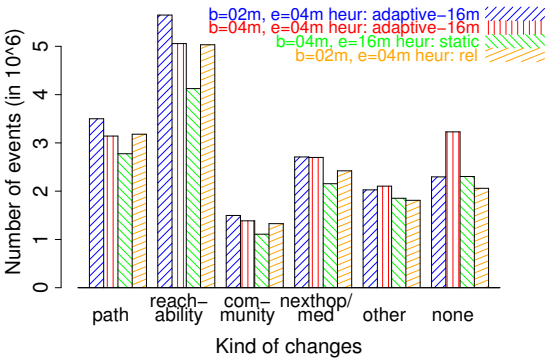


Figure 5.12: Event characterization for various timeout heuristics.

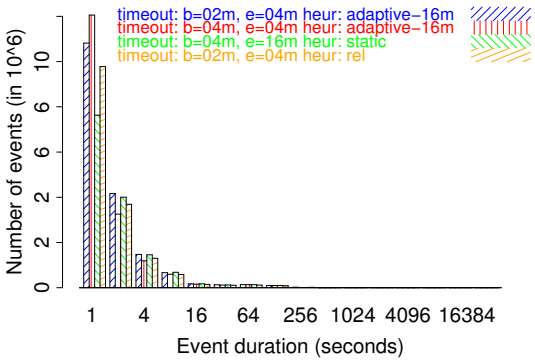


Figure 5.13: Event duration for various timeout heuristics .

event associated timeout should be compatible with the update burst timeout. Choosing an event timeout less than the burst timeout indicates that one is more stringent for grouping events than for updates and can create situations where a burst should be part of two events. To avoid this, we choose to be more lenient and use a timeout greater than or equal to the burst timeout. Still the timeouts should not be too large to avoid grouping those bursts together that were separated by the timeout of the burst calculation. Accordingly we select the following parameter sets for relative: (bursts=2m,events=4m), (b=4m,e=8m); static: (b=2m,e=8m), (b=4m,e=16m); adaptive: (b=2m, e:(max=16m,rel=4m), (b=4m, e:(max=16m,rel=4m)).

Overall we notice that the specific choice of parameters and heuristic does not appear to cause major differences. For example, Figure 5.12 shows a grouping of the events into similar categories. An event belongs to group “path” if at least one of its bursts belongs to this group. An event belongs to group “reachability” if no burst belongs to group “path” and at least one burst belongs to group “reachability,” etc.. Some observations about bursts carry over to events, e.g., events are again dominated by path and reachability changes. But while more than 24% (28%) of the 2 (4) minute bursts are in group none, less than 14% (18%) of the events are. In comparison, note that the fractions of events with community, next-hop, or other attribute change have increased significantly.

One of the reasons for proposing to use the static and adaptive timeouts is that we want to limit the duration of each event in order to pinpoint the origin of instability, either in the next step or the final step of event correlation. This is indeed the case for these heuristics. Figure 5.13 show a histogram of the event durations for a subset of the parameter choices. Note that most events, just as most bursts, are short, yet a few last for a long time. While the events identified by the relative heuristic can be long (more than 4 hours) the ones generated by the static heuristic are indeed less than 16 minutes. The adaptive timeout events are somewhere in between since they separate active from quiet periods and are patient enough for active periods to finish.

The next question motivated by our experiences with the simulated failures is how many observation points observe each change. More than 66.6% of instabilities are only observable at a single observation point and 16.1% at two. Overall 95.3% are observable at less than 10 observation points. This again confirms that BGP indeed provides significant isolation against routing updates.

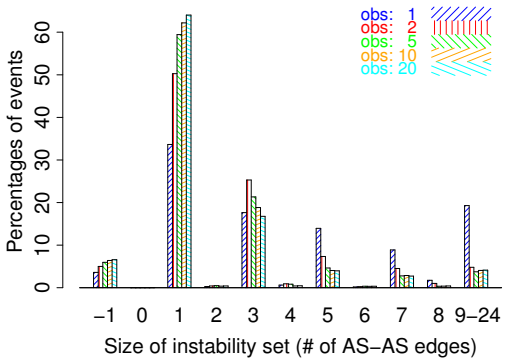


Figure 5.14: Beacons: instability set size hist. for # of obs. (adaptive (b=2m,e: (max=16m, rel=4m)); heur.: standard).

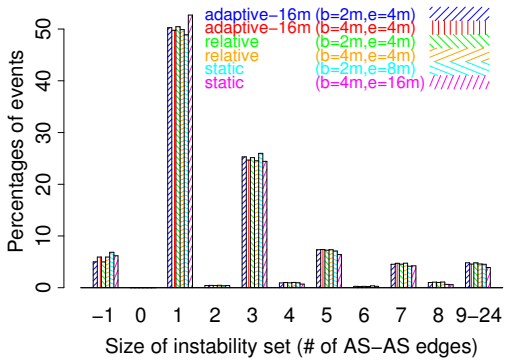


Figure 5.15: Beacons: instability set sizes hist. for timeout heuristics (obs.: 2; heur.: standard).

5.5.3 Instability candidates

Once the bursts are grouped to events, we can apply path heuristics to compute the instability candidates for each event to compare with simulation results.

5.5.3.1 Beacons

We first apply our heuristics to BGP beacon prefixes. Our experiences with the simulation results have shown that we can usually narrow the candidate sets to an instability candidate set of about $3-7$ edges for more than 70% of the examples. For the beacons we can do even better. With the adaptive event timeout (b=4m,e:(max=16m,rel=4m)) heuristic and at least two observation points we can narrow the instability set to three or fewer AS edges for more than 76% of the events (see Figure 5.14). We verified that each instability set contains the edge for the origin AS. While in theory each beacon should be observable at all observation points, this is not the case. For example, some events consists of only withdrawals and the previous event is also a withdrawal. These kinds of events are captured in the category labeled "-1" and sum to about 5%. Furthermore, with the adaptive event timeout, more than 31% of the events are only observable at a single location. This can be explained by BGP update delays, e.g., due to route flap damping, filtering at intermediate peers, time synchronization problems of the collectors, or other instabilities that affect beacon prefixes. Using the static (b=4m, e=16m) heuristic reduces this to only 18% since it ensures that appropriate events are clustered while other unrelated events are separated from the beacon events. Thus the percentage of events with an instability set of three or fewer AS edges increases to 77.6%. Note that identifying three AS edges usually includes the edges of two ASes and the edge between the two ASes. For more than 50% of the beacon events both heuristics let us identify the origin AS correctly as the instability creator. Increasing the number of observation points to at least two increases this to more than 64%, a rather nice success rate. Indeed if one considers a set of four ASes good enough for pinpointing the instability, we succeed for more than 90% of the cases with only two observation points. Figure 5.14 highlights again the benefit of having information at multiple observation points. Figure 5.15 accentuates that while the timing heuristics differ, e.g., in terms of the number of events that are only observed

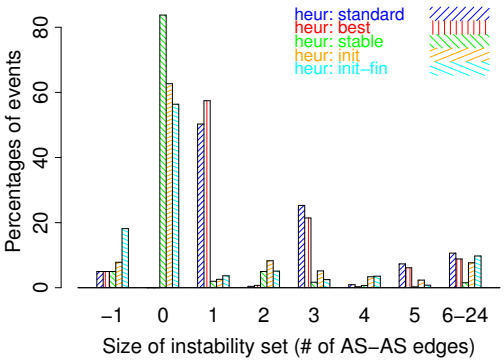


Figure 5.16: Beacons: instability set size hist. for heuristics (adaptive: (b=2m,e:(max=16m,rel=4m)); obs: 2).

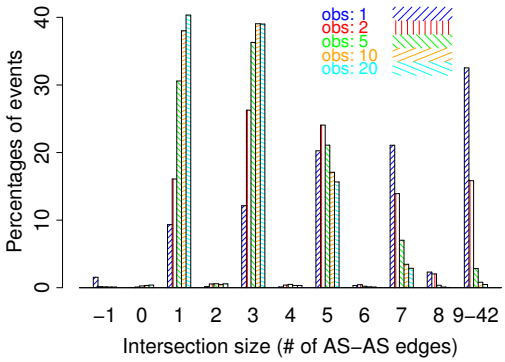


Figure 5.17: Instability set size hist. for various # of obs. (adaptive: (b=2m, e:(max=16m,rel=4m)); heur.: standard).

at a single observation point, they still generate results of a similar accuracy level in terms of the instability sets they identify.

Figure 5.16 shows the histogram of the sizes of the instability sets for the different path heuristics. The conservative approach of the "standard" heuristic is rather successful, and the "best path" heuristics improves the accuracy by more than 7%. But the "stable", "initial", and "final path" heuristics prove to be disastrous. The intersection size is empty in more than 55% and for stable up to 83% of the cases. This indicates that the examples shown in Section 5.1 are not just possible but that BGP features that create similar results are in use in the Internet. Note that the results are similar for different choices of timeout heuristics.

5.5.3.2 All prefixes

Next we apply our path heuristics to all prefixes and consider the same set of plots as for the BGP beacons (Figures 5.17, 5.18, 5.19). Clearly the results are not quite as good as for the BGP beacons. On the other hand being able to pinpoint the origin of an instability, which is observed at two observation points, to three AS edges for more than 42% of the cases, and to five AS edges for about 70% for all timeout heuristics, and more than 76% for some, is quite impressive and shows that we have made significant progress towards understanding the origin of BGP instabilities. We checked that almost all of the time, 99.9%, instability sets with three AS edges correspond to those that surround a BGP peering location, e.g., AS1 and AS2 are peers and the instability set contains (AS1-AS1, AS1-AS2, AS2-AS2). Furthermore, most of the time the AS edges in the instability set are continuous on some AS path.

Furthermore, with increasing number of observation points, the ability to pinpoint increases (see Figure 5.17). But most important, the instability set is hardly ever reduced to size 0, which confirms that using the standard methodology is a safe approach for reducing the candidate set size. Indeed with more than 5 observation points it is possible to reduce the size to five edges for almost 90% of the events for all timeout heuristics without increasing the fraction with zero size instability sets.

Comparing the various timeout heuristics, Figure 5.18 shows that the impact of the specific methods increases but is not that dramatic with the exception of the static heuristic based on 4

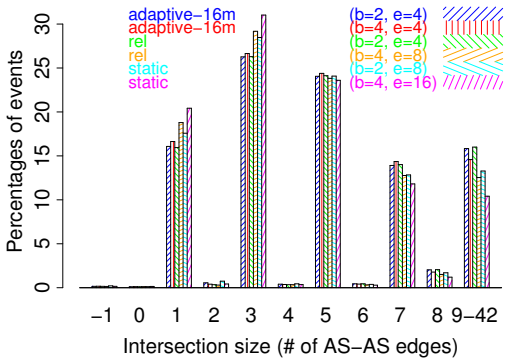


Figure 5.18: Instability set size hist. for timeout heuristics (obs.: 2; heur.: standard).

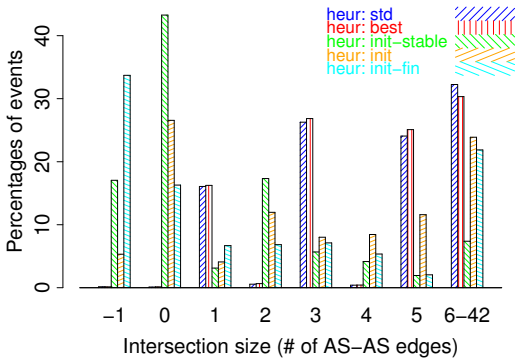


Figure 5.19: Instability set size hist. for various heuristics (adaptive: (b=2m, e:(max=16m,rel=4m)); obs.: 2).

minute bursts. This indicates that timeout values of 2 to 4 minutes used in the other heuristics may not yet be optimally chosen.

The impact of the path heuristics (Figure 5.19) is on the one hand positive, e.g., for "best" path, but on the other hand again disappointing for "initial," "final," and "stable". Regarding the origin of instabilities, we further inspect the instability sets with up to and including three edges for the adaptive heuristic (b=2m,e:(max=16m,rel=4m) including all observation points. For these, 30.4% of the time the origin AS is the only AS in the instability set. For 66.3%, the origin AS is one of the edges. This leaves us with 3% that are unconnected to the origin AS. Of these, .4% included only a single AS, while 1.3% include multiple ASes. The others 1.3% include some inter AS link.

5.5.4 Event correlation across prefixes

So far we have seen that it is possible for most events that involve more than a single observation point to identify a reasonable-sized instability candidate set. To locate a plausible origin for the remaining events we can take advantage of our knowledge of BGP instabilities, see Table 4.2. Most of the plausible events cause updates to multiple prefixes at about the same time. Given that the instability set computation has already narrowed the origin of the instabilities, majority decisions help us now to further pinpoint the instability origins using the Greedy heuristic, see Section 5.2.3.

Using rather aggressive timeouts of 4 minutes to determine which edges are considered correlated for each event, and artificially shortening the duration of each event to a maximum of 16 minutes helps us to ensure that we mainly catch correlated events. In each time period, selecting the edge that is involved in the most instabilities is easy as the number of events differ by a rather significant factor, usually larger than 1.5. We have even observed factors of up to 3. Indeed the distribution of correlated events by edge appears to be consistent with a Zipf distribution, which justifies using the Greedy heuristic. This also explains why the Greedy heuristic is rather successful in identifying instabilities. Once the most likely candidates have been identified the Greedy heuristic may have to break ties. We choose to not break ties but rather include all edges as possible instability origins.

After applying the Greedy heuristic, with the adaptive timeout heuristic, we are able to associate 93.4% of the prefixes to a single AS as a possible origin for the instability. A single AS corresponds to the intra-AS edge. In more than 97.2% of the cases Greedy narrows the possible instability origin to at most three AS edges. Even if the instability origin includes three AS edges, the instability origin points to a single peering connection in more than 47.5%. As the edges correspond to the two intra-AS edges and the link between them. In the other 52.5%, the instability origin seems to lie inside any of the three involved ASes. If we require that each correlated event has at least 100 prefixes, we are able to associate 96.3% of the prefixes with a single instability origin.

5.5.5 Validation

Given that Greedy appears to be able to further pinpoint the instability origin, the obvious question is if the caveats and dangers highlighted by the examples in Section 5.1.2 have been sufficiently addressed or if the heuristic misguided us. We address this issue by further validating our results in two ways. First we use syslog data from a large tier-1 ISP to see if we can correlate times when Greedy identifies the AS edge *A-A* (A is the AS number of the tier-1 ISP) as instability origin with session resets within the tier-1 ISP. Second we again take advantage of our simulator by comparing the inferred origin of instability with simulated results. For this purpose we select some instabilities identified by Greedy to simulate and then compare the outcome.

For the first validation step we selected 35 events for which Greedy identified *A-A* as instability origin and the syslog data is available. We checked them against the appropriate router syslog data from the ISP. (Note that the syslog data is not available for the entire week.) We say that we find a session reset that corresponds to the event if at most the 5 minutes of the instability event overlaps with the session reset time window – starting 1 minute before the time session reset occurs shown in the syslog and ending 1 minute after that. Adding 1 minute to the time window addresses potential clock synchronization issue and the delay in observing the change at the BGP monitoring points due to BGP rate limiting timers and propagation delays. The results are rather promising as we can find related session resets for 26 or 74% of them. One should not expect to find all events since some may not have been caused by session resets. Furthermore, note that not all session resets will cause updates as discussed in Chapter 4.

For the second validation step we selected 20 events for which Greedy identified a single AS-AS edge as the instability origin. The corresponding simulation first excludes those prefix origins that do not use this AS-AS edge on one of their best path under the simplified simulator BGP model. Note that otherwise this will introduce some errors due to inaccurate policy inference. It then fails the appropriate edge, or approximates the failure inside an AS by deleting all edges containing the corresponding AS. This effectively assumes that all paths going through the AS are affected. We find that the heuristics when applied to the outcome of these simulations identify the same AS-AS edge in 90% of the cases improving our confidence in the appropriateness of the heuristics.

5.6 Summary

Trying to identify the origin of global Internet routing instabilities poses challenges that stem from the complexity of the BGP decision process, the challenging problem of achieving a globally optimal routing via local routing configuration by various administrators as well as the global traffic dynamics. In this thesis we propose a methodology for identifying the origin of routing instabilities by examining and correlating BGP updates along three dimensions: *time, views,* and *prefixes* and show how it can be adapted to account for the complexities of BGP. By applying our heuristics first to an ideal world where we control the failure and the observation points and then to a huge amount of actual BGP updates, we show that the methodology is sound and accounts for cases that have been previously ignored. Indeed with only two observation points, we are able to pinpoint the origin of instabilities due to beacons to no more than three AS edges for more than 76% of the cases. This increases to only five AS edges when considering all prefixes. Relying on Zipf like characteristics of correlated events across prefixes, the Greedy heuristic is capable to further pinpoint the origin to a single AS for more than 93% of the prefixes. Accordingly we conclude that despite the intricacy of ISP routing policies, and the issues regarding propagation, or lack thereof, of BGP update messages, and complexity of the Internet topology, we have demonstrated significant ability at narrowing down the location of BGP instabilities.

6 Measuring BGP Pass-Through Times

Even though we studied the BGP convergence process extensivly in the last chapters, a remaining unknown component of the update delay is the contribution of each router along the path. A detailed exploration of the delay of each router and how it relates to factors, such as CPU load, number of BGP peers, etc., can help explain these observations.

We propose to explore pass-through times of BGP updates using a black-box testing approach in a controlled environment with appropriate instrumentation. To this end, we have setup a test framework that consists of:

- Device under test (DUT): a router.
- Load framework: that can be used to impose a specific, pre-defined load on the DUT. In our case it consists of several PCs and an Agilent router tester. BGP workloads can be generated by the PCs as well as the router tester. The router tester is used to generate controlled rate data traffic. Tests are repeated with both PC as well as router tester generated BGP workloads.
- Instrumentation framework: that allows us to measure not just the pass-through times of BGP updates but also the load imposed by the load framework. It consists of several packet-level monitors, periodic router queries and the router tester.

The testbed has wider applicability. For example, it can be used to explore other router measures, such as line card FIB convergence.

Our methodology goes beyond RFC compliance tests and benchmarks, e.g., BMWG [116], in that we do not consider pass-through times in isolation. Rather, we investigate the correlation between BGP pass-through time and router load using a variety of stressors. BGP load is affected by such variables as (1) number of peers, (2) routing table size, and (3) BGP update rate. Non-BGP related tasks are many. We do not attempt to account for all of them individually. Instead, we impose a specific background traffic load which causes the ip_input task's CPU usage to increase. To measure the CPU load we periodically sample the cumulative route processor's load. With controlled experiments we can then use previously obtained CPU load data to infer the BGP portion of the CPU load. Our initial results are about establishing a baseline for pass-through times. We also explore some aspects of stress response. A stress situation arises when the imposed load reaches the limits of DUT.

Additional parameters that may effect pass-through time include the configured I/O queue length for BGP processes, ACLs' complexity and hit ratio, BGP policy settings such as route-maps, peer-groups, filter-lists and/or communities, as well as AS prepending, etc. These are beyond the scope of this thesis.

Parameter sensitivity tests and reasonable configurations were used to reduce the otherwise many experiments, given the large number of parameter value permutations.

Our experiments show that in general the DUT handles BGP stress well, which is in line with recent findings [139]. Yet the per hop BGP processing delays can be significant. Updates

are only processed every 200ms even when the MRAI timer is inactive. Activating the MRAI timer adds further delay components causing higher delays occur with increasing MRAI timer values. DUT targeted traffic, even at low data rates, drastically impacts CPU load and accordingly pass-through delays. We notice that update the rate is not nearly as significant a factor for causing delays as is the number of peers. Yet high update rates occurring concurrently on multiple peers, as is happening with after router reboots, can cause problems.

The rest of the chapter is structured as follows. In Section 6.1 we describe our methodology for measuring pass-through times and imposing a controlled router CPU load. Next, in Section 6.2, we describe in detail the configuration and tools that constitute our test framework. We then describe our experiments and report their results in Section 6.3. Section 6.4 concludes this chapter and comments on the practical learnings of this work as applied to current operational practice and future routing protocol design.

6.1 Test methodology

Three topics need detailed explanation: measuring pass-through time, separating router processing delay from MRAI timer delay, and imposing a controllable load on the DUT.

6.1.1 Measuring pass-through times

There are three common approaches to network measurement: passively observing regular traffic, actively evaluating injecting traffic at end points within the injecting application, and passively measuring actively injected traffic. Since we operate in a testbed the first option is not applicable. Accordingly, we use specifically designed updates as active probes together with a monitoring BGP session. The timings of the probes and the resulting response updates generated by the DUT and directed to the monitoring session are passively measured using dedicated and synchronized packet capture.

Using special updates gives us the ability to impose patterns with certain characteristics, such as ensuring that the DUT will relay it to the monitoring session. The alternative approach is to replay captured BGP update traces. While this may provide useful background traffic it suffers from several shortcomings. First the pass-through time of a router depends on the settings of several BGP specific parameters such as the value of the MRAI timer. In order to distinguish the delay due to this timer from the delay due to the router we need certain update patterns which may or may not be present in regular BGP traces. Second, not all incoming BGP updates trigger an outgoing BGP update. Therefore it is hard to tell which BGP updates are discarded or are combined into other updates. Furthermore, the amount of work associated with each BGP update will vary depending on its content with respect to the router's configuration.

The simplest form of a BGP update probe pattern is comprised of a single new prefix. Since the prefix is new, the update has to be propagated to all neighbors and the monitor session, policy permitting. The drawback is that the routing table size grows ad infinitum and that the available memory becomes an unintended co-variable. The table size can be controlled via explicit or implicit withdrawals. We use implicit ones since BGP withdrawal processing differs from update processing and the results would have to be separated. Implicit withdrawals

on the other hand are indistinguishable from other updates. Each time a probe update for a prefix is to be sent we randomly choose an AS path length that differs from the previous path length. This ensures that the update is propagated to the monitor session and that the quality of the path improves or deteriorates with the same probability. The BGP update rate for both probes and traces can be controlled. In summary probe patters are convenient for active measurements while replaying BGP traces is appropriate for generating a realistic load on a router.

Pass-through times are not constant. They vary with the makeup of the updates and the background load caused by other factors. Accordingly, we obtain sample values by measuring the time difference between in-bound (into the DUT) probe injections and out-bound (onto the monitoring BGP session) update propagation. To avoid time synchronization errors the packet monitors are dedicated, line rate capable, capture only cards with highly accurate, synchronized clocks. Furthermore, all other BGP sessions are terminated on the same machine, either the PC or the router tester.

Throughout this chapter we use $10,000$ prefixes from the $96/8$ range as probe prefixes. The central part of the experiments last for 15 minutes and our probing rate is a rather low 1 update a second. This guarantees that the TCP throughput will never be problematic. We started by using 10 probing sessions but realized that interactions with a periodic timer limited the accuracy of the estimates. The periodicity of the timer in question is $200ms$ which with 10 probes per second gave us only 2 samples per timer. To increase the rate of samples per timer to 10 we increased the number of probe sessions to 50. Each probe session uses a different random offset within the second for sending its probe. This ensures that the probing is done at exponentially spaced but fixed intervals within the second. A histogram of the resulting pass-through times is plotted in Figure 6.1. Interestingly, the pass-through times vary from $2.4ms$ to about $200ms$ with some ranging up to $400ms$. The average pass-through time is $101ms$. The even distribution in the range of $2ms$ to $200ms$ indicates some kind of timer. Indeed, closer inspection reveals that the router limits the update processing to 5 times a second. This should result in a theoretical upper bound on the pass-through times of $200ms$. Yet some of the updates are held back for one update processing cycle. This results in pass-through time greater than $210ms$ for 1.3% of the probes.

To determine how the pass-through time compares with the one-way packet delays we also measured the one-way delay experienced by IP packets of three typical packet sizes (64, 576, 1500 bytes) that were sent during the same experiment, see Figure 6.1 for the 1500 byte packets. The average delays are, $0.028ms$, $0.092ms$ and $0.205ms$, hence, significantly shorter. This shows that a BGP update is delayed 11 times longer in the best case and 500 times longer on average than a simple IP packet. While this might seem significant, the total accumulated delay, at $100ms$ a hop along a 20 hop router path, would be rather small at under 2 seconds. This indicates that we have to consider additional factors.

6.1.2 MRAI delay

The purpose of the MRAI is to limit the number of updates for each prefix/session for a peer to one every x seconds. A typical value for x is 28 seconds (see Section 2.1.1 on page 14). If the timer fires at time $t - x$ and t then all updates received and processed within this interval of size x are batched and sent shortly after time t. Based on this observation an upper bound

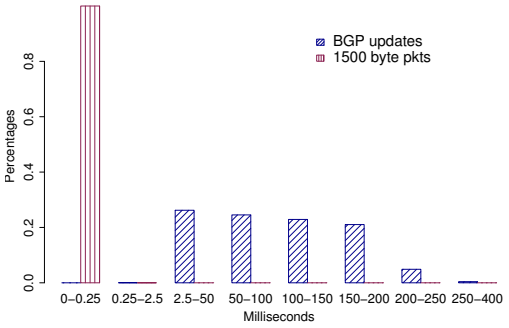


Figure 6.1: Histogram of pass-through times together with one-way packet delays for typical packet sizes 64, 576, and 1500.

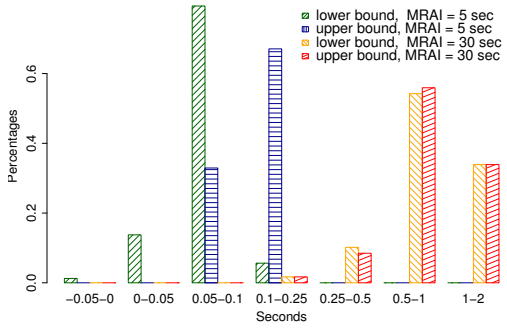


Figure 6.2: Histogram of upper and lower bounds on pass-through times for MRAI values of 5 and 30 seconds.

for the pass-through time can be derived, even when the MRAI timer is active: for each MRAI timer interval consider those probes with minimal pass-through delay. A lower bound is derivable from the probe with largest pass-through delay within the interval. This probe arrived too late to be included in the previous MRAI timer interval.

Figure 6.2 shows the histogram of the pass-through times for two different MRAI timer values: 5s and the default Cisco value, which is roughly 30 seconds. Note that with the MRAI timer in place the minimal measured pass-through times are 71ms and 122ms. On the other hand the maximum values for the lower bounds are 0.121 and 1.72 seconds! This indicates that an update might have to wait a significant amount of time before it is processed even if it reaches the router at a good moment. This is especially the case for larger MRAI values where the average pass-through time increases from 109ms for the 5s MRAI timer to 883ms for the default MRAI value. Note that each experiment is run for the same duration. Accordingly the number of samples for the pass-through time decreases as the MRAI value increases.

Overall it seems that even for small MRAI values the timer interactions between MRAI and the BGP update processing timer increases the minimum pass-through time significantly. As the MRAI value is increased the minimum pass-through time also increases and will clearly dominate any link delays. Furthermore, inspection of the probes on the monitoring session reveals that the order of the update probes is not maintained. This means that a probe that was sent 10 seconds later than another might be observed earlier on the monitoring session.

6.1.3 Controlled background CPU load

Imposing a controllable background CPU load on the DUT is necessary in order to study how it responds to stress. The goal is to identify a set of tasks that generate a constant CPU load independent of BGP. This is difficult as it implies generating an input load that is uniformly served by a task running at a uniform priority. This is rarely the case. In Cisco IOS the BGP processes (and any routing tasks) have higher scheduling priority than almost everything else targeted at the CPU. The IOS ip_input task is a high priority process whose CPU use is related to the rate of a packet stream directed to the DUT's main IP address.

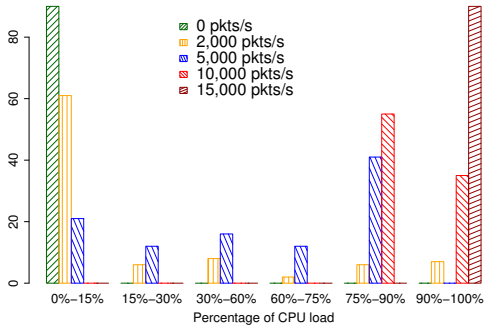


Figure 6.3: Histogram of CPU load estimates for packet rates of $2k, 5k, 10k$ and $15k$ directed to the router IP.

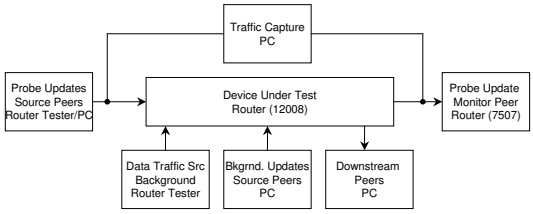


Figure 6.4: Test-bed setup for router testing.

Another problem is measuring the CPU load. The CPU load of a Cisco router can be queried in two ways: via a command at the telnet interface of via an SNMP query. Unfortunately, the default priorities of both telnet and SNMP are lower than those of BGP and the packet processing tasks. Accordingly, for calibration only, we raised the priority of SNMP task and then measured the CPU load both via the command line as well as via SNMP for 5 rates: $2k, 5k, 10k, 15k$ pkt/s. Both estimates aligned quite well with the only problem that the command line interface did not deliver any values under high loads due to starvation. Figure 6.3 shows a histogram of the CPU load. $2k$ packets impose almost no load. $5k$ is already significant. $10k$ is almost critical while $15k$ is well beyond critical. Note that a $15k$ packet rate corresponds to a bit rate of 0.36 Mbits, which is rather modest for a high speed interface on a high end router. This should encourage providers to filter internal destination addresses on *all* incoming connections.

6.2 Test framework

The testbed shown in Figure 6.4 illustrates its functional building blocks. The physical layout is more complex and not shown here for the sake clarity and brevity.

The device under test (DUT) is a Cisco 12008 GSR [54] equipped with: 256MB memory, 512KB of L2 cache, 200MHz GRP CPU, four Gigabit SX and 8 Fast Ethernet interfaces. It runs IOS version 12.0(26)S. An Agilent RT900 router tester is used for selective experiment calibration and to generate data traffic. BGP updates, probes and background, are generated by a PC. The monitoring peer runs on a Cisco 7507. Probe update traffic from the PC into the DUT is captured by Endace DAG cards [140]. Outgoing probe update traffic from the DUT to the monitoring peer is also captured by an Endace DAG card. All cards are synchronized.

The DUT is subjected to three traffic types: BGP update probes, BGP background activity updates and non-routed data traffic directed to the DUT. Probe updates are used to compute DUT pass-through times. We create a BGP activity noise floor by generating separate update streams, called background updates, that are in turn propagated to multiple downstream peers.

Data traffic is used to indirectly control the CPU load and hence the time alloted to BGP processing.

DAG generated time-stamps are used to compute the DUT pass-through time. We use tethereal to decode and reconstruct the BGP TCP sessions from the capture files. To ease the configuration and setup of each experiment various scripts automatically configure the PCs, the router tester, and the routers, then start the experiments and after it is done start the evaluation. Unless specified otherwise each experiment lasts for 15 minutes actual time but the evaluation is not started for another 15 in order to retrieve all updates.

6.3 Pass-through times

Section 6.1 introduces our methodology for measuring pass-through times and shows how to impose a background load on the DUT. In this section we explore how pass-through times change as the demand on the router increases. Due to the large number of parameters we cannot test all combinations. Rather, we perform a number of tests to explore the variables to which pass-through times are sensitive, including the background CPU load, the number of sessions in combination with the BGP update rate, and the complexity of the BGP table in combination with the BGP update rate.

More precisely in a first step we combine BGP pass-through probes with the background CPU load. Next we increase the CPU load by adding $100/250$ additional BGP sessions and a total of 500 BGP updates a second. This experiment uses a regular pattern of updates similar to the probes. Based on this calibration of our expectation we explore the load that is imposed by actual measured BGP tables. The next two experiments differ in that one uses small BGP tables containing between $15,000$ - $30,000$ prefixes while the other uses large BGP tables containing between $110,000$ - $130,000$ updates. Due to the memory requirements of this table the number of additional sessions is reduced to 2. This provides us with a setup to explore different BGP update rates: as fast as possible (resembles BGP session resets), 200 updates and 20 updates a second.

6.3.1 Pass-through times vs. background CPU load

This set of experiments is designed to show how the background CPU load influences the BGP pass-through delays. Accordingly we combine the approach for measuring BGP pass-through delays via active probes with that of imposing a controlled background CPU load via a controlled packet stream directed to the DUT's IP address, see Section 6.1. We use a packet stream of 0, $2k$, and $10k$ packets as the first two impose no additional or just minimal load while the latter is already almost critical.

The histogram of the resulting pass-through times is shown in Figure 6.5. While the differences may at first appear minor the CPU load nevertheless has an impact. It causes a delayed invocation of the BGP update processing task which is reflected in the increase of the number of updates with a pass-through time larger than $210ms$. With no additional load only 1.3% of the updates are in this category. With $2k$ packets this increases to 2.15% and for $10k$ packets to 3.73%. Note that a probe rate of 50 updates a second coupled with 5 invocations of the BGP update processing task every second should create delays longer than $200ms$ for at most

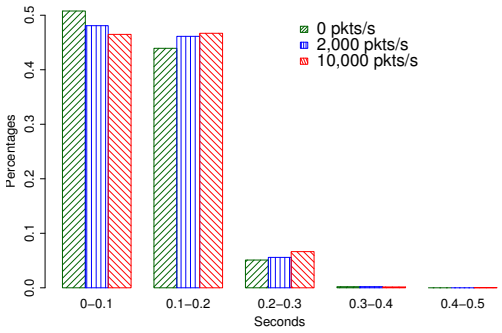


Figure 6.5: Histogram of pass-through times subject to different levels of background traffic (0, 2k, 10k pkts/second).

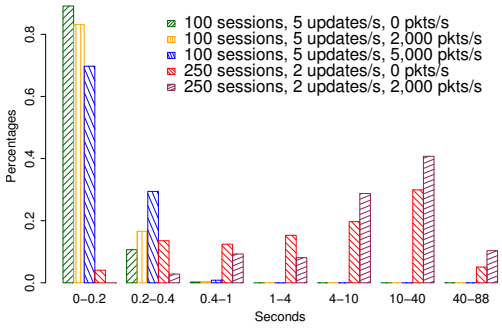


Figure 6.6: Histogram of pass-through times subject to different # of sessions (100/200) and background traffic (0, 2k).

10% of the probes. Overall we conclude that the increased CPU load delays the invocation of the BGP update processing task and therefore increases the pass-through delays. Yet, due to the timer, the delay increase is on average rather small: from 101ms to 106ms to 110ms.

6.3.2 Pass-through times vs. number of sessions

This set of experiments is designed to explore if the number of sessions has an impact on the BGP pass-through times. So far our load mix consisted of the active BGP probes and the packets which cause a background CPU load. Next we add additional BGP sessions (100/250) and 500 updates a second to this mix. The simplest additional sessions are similar to our probe sessions with the exception that we increase the update rates to 2 and 5 updates per second respectively.

Figure 6.6 shows a histogram of the resulting BGP pass-through times. Interestingly adding 100 sessions poses no significant problem to the router. Yet adding 250 sessions causes way too much load on the router even without background traffic. Note that Cisco recommends to keep the number of sessions for this specific router below 150. Adding to the 100 session experiment a background CPU load of 2k (5k) increases the CPU load from a 5 minute average of roughly 67% to 83% and then to 95%. That CPU loads are not summable is an indication for the possibility of saving some CPU by delaying the processing of the BGP updates. The additional BGP sessions increase the average pass-through times to 116ms. The CPU load is then responsible for the increase to 130ms and respectively 160ms. The increase of the percentage of BGP probes that take longer than 200ms is even more dramatic: first from 1.3% to 7.4% and then with the packet load to 12.5% and 25.9%. Still the maximum pass-through times are reasonable small at less than 800ms.

The further increase of the number of sessions to 250 causes a multitude of problematic effects. First the router is no longer capable of processing the updates so that it can send TCP acknowledgments on time. Accordingly the number of TCP retransmissions increases from almost none, less than 0.15%, to 2.5% of the BGP probe updates. Second the number of probes propagated to the monitoring sessions is drastically reduced. With 250 sessions the router does not propagate updates for 39.8% of the probes. This problem is aggravated

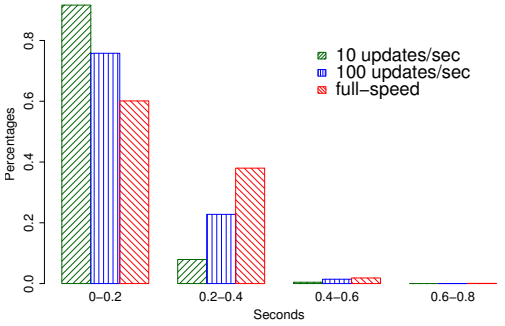


Figure 6.7: Histogram of pass-through times as update rate increases (small table, 2 sessions).

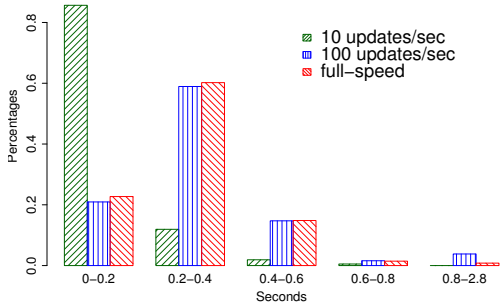


Figure 6.8: Histogram of pass-through times as update rate increases (large table, 2 sessions).

(49.2%) by adding $2k$ packets of background traffic. While this reduces the number of samples of the pass-through times their values are now in a different class with average pass-through times of $9,987ms$ and $15,820ms$. The pass-through times increase by several orders of magnitude.

6.3.3 Pass-through times vs. BGP table size and update rate

So far all tests consisted of either probe updates or artificial patterns. Accordingly we now replace these artificial BGP sessions with actual BGP updates. For this purpose we have selected two sets of two BGP routing tables dumps, one containing $37,847/15,471$ entries and the other containing $128,753/113,403$ entries. Note that routing tables generally differ in terms of their address space overlap, their size and their update rate. In this preliminary work we want to study the impact of the BGP update rate on the pass-through times for different table sizes, we opted for some regular pattern. The first pattern, called "full-speed", corresponds to continuous session resets and is realized by repeatedly sending the content of the BGP table to the router as fast as possible. The second pattern, called "100 updates/sec", is similar but limits the rate of updates to 100 BGP updates a second. The third pattern, called "10 updates/sec", further reduces the rate of updates to 10 BGP updates a second. As it is well known that session resets impose a large load on the router one may expect larger pass-through times. As one hopes that the router can perform session resets at a rate of 100 updates a second the second pattern should be simpler and not impose quite such a large load. The 10 updates a second can be expected to impose even less load than our update probes and therefore should provide us with a base line.

Figure 6.7 and 6.8 show the histograms of the pass-through times for experiments with the two small tables, Figure 6.7, and the two larger tables, Figure 6.8. As expected the pass-through times for "10 updates/sec" is with an average of $111ms$ for the small table only slightly increased. The impact of the large table is visible in its average of $127ms$. For the small table the full speed update rate is significantly higher than 100 updates/sec and imposes a CPU load of 88% to 60%. This difference in terms of update rate is not as large for the full table. Here the full patter generates a CPU load of 100% as one would hope for. For the small table the average pass-through times increase significantly from $147ms$ to 181ms. If this may

not seem like much there is huge danger hiding here, the one of missing BGP keep-alives. In both "100 updates/sec" experiments and the "full-speed" experiment for the large table the router did not manage to send its keep-alive in time. Therefore these experiments terminated prematurely. Overall the maximum pass-through time in the small table experiments are reasonable with a maximum of less than 710*ms* and only 35% greater than 210*ms*. For the more realistic cases with the large tables this changes. Here the maximum pass-through times increase to 2.8 seconds and the percentages larger than 210*ms* increases to 76%.

6.4 Summary

Our results show that it is possible to determine the pass-through times using a black box testing approach. In general we find that BGP pass-through times are rather small with average delays well less than 150*ms*. Yet there are situations where large BGP convergence times may not just stem from protocol related parameters, such as MRAI and route flap damping. The pass-through time plays a role as well.

Even when the MRAI timer is disabled and the router is otherwise idle, periodic BGP update processing every 200*ms* can add as much as 400*ms* to the propagation time. Increasing MRAI values appear to trigger timer interactions between the periodic processing of updates and the timer itself which causes progressively larger delays. For example, when using the default MRAI timer value even the average estimate increases to 883*ms* but more importantly the lower bound estimation can yield values for up to 8 seconds. This indicates that there is an additional penalty for enabling MRAI beyond the MRAI delay itself. Furthermore, we have observed out of order arrival of updates which suggests that using multiple prefixes for load balancing or fail-over may not always function as expected. This bears more detailed verification.

Low packet rate data traffic targeted at the DUT can impose a critical load on the router and in extreme cases this can add several seconds to BGP processing delay. These results reinforce the importance of filtering traffic directed to the infrastructure.

As expected, increasing the update rate does have an effect on processing time, but it is not nearly as significant as adding new peers. Worth noting is that concurrent frequent updates on multiple peers may cause problems. For example, 53 peers generating 150 updates per second can cause the router to miss sending a KEEPALIVE in time, thus resulting in a session reset.

Overall the in general small pass-through times indicate that the current generation of routers may enable us to rethink some of the timer designs/artifacts in the current BGP setup. Yet care is needed to not trigger the extreme situations outlined above. Furthermore, additional analysis is needed to better understand the parameters affecting BGP processing rate, such as FIB updates, line card CPU loads, BGP update contents and other configuration related parameters already mentioned above.

7 Towards more Realistic Router Testing

This chapter is based on the observation, that operational problems in the backbone of a network rarely stem from simple protocol implementations errors. Well designed test-suites are available to assure a minimum of functionality and interoperability. Yet, despite the wealth and breadth of test tools, network equipment continues to fail in the field. Conventional wisdom suggests that this is an expected outcome of a product release cycle that is too short, compared to circuit switched equipment, and that the development of network equipment is proceeding at a faster rate than test equipment. That may be partly true, but the reality is that creating field conditions for packet-switched test setups are not well understood. There is an increasing discrepancy between field and testing conditions.

Today, network equipment is becoming more and more complex, it is computationally distributed across the main CPU and line card CPUs and it runs many different services on the same device at the same time. This causes problems or even failures that arise from the interactions of the various functionalities. This is particularly acute for network protocol interactions, because, given the large number of possibilities, they are difficult to test.

Therefore it is very necessary to go beyond simple RFC compliance tests. The ideal network equipment testing platform also needs to map protocol idiosyncrasies to the way in which test engineers work and think. Envision an easy to use test tool that is able to create workloads for:

- control plane traffic per protocol with varying levels of activity
- data plane traffic per protocol with varying levels of activity
- protocol mix across layers 2-7

This means that the workload generators produce "realistic" looking signaling and data traffic, and thus emulates placing the system under test in different "virtual locations" with different characteristics.

Imagine a tool that is easy to use and at the same time able to create complex test-scenarios, where a test engineer can set high-level conditions. With such a tool, complex multi-service test scenarios can be created without the pain of dealing with all the details of each protocol. Tests of involving many protocols can be performed automatically and reproduce the variability that is observable in an operational backbone. Constructing more and more complex test setups is possible, which adds confidence to the robustness of the router design and implementation.

In this chapter we illustrates how such a tool can bring more variability into a test-lab at the example of a BGP workload generator.

7.1 Design goals

In this section we discuss the basics of our approach. We start by looking at the test framework in Section 7.1.1, provide an overview over the workload generation in Section 7.1.2, and finally discuss some requirements for the tool in Section 7.1.3.

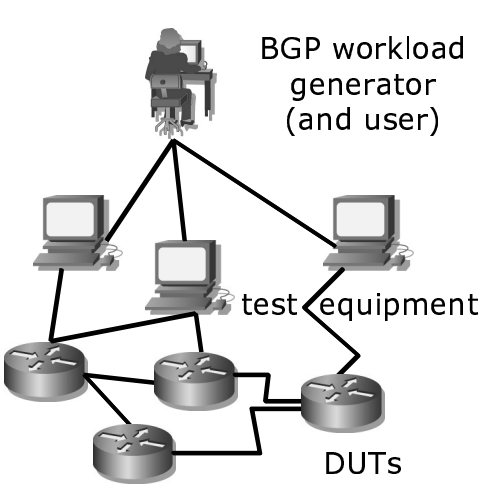


Figure 7.1: Test-bed for router testing.

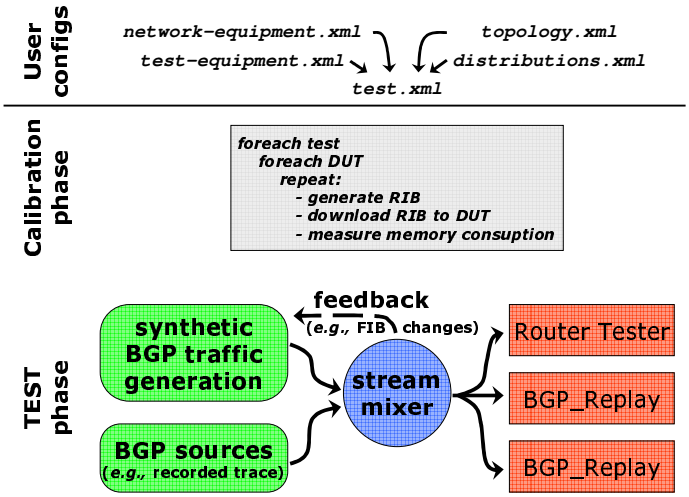


Figure 7.2: Overview of synthetic BGP traffic algorithm.

7.1.1 Test framework

Our goal is to be able to study system limits and protocol or component interactions in a controlled environment, in an appropriate test-bed. Internal test-labs are not uncommon for vendors, telecom operators and large enterprises. They are established, at great expense, for the purpose of certifying new equipment, testing new service configurations and performing capacity planning. These test-beds reflect, on a small scale, the main topological characteristics of the production network. In addition to switching nodes, they are outfitted with traffic playback, generation and measurement equipment.

Figure 7.1 illustrates the functional building blocks (see also RFC 2544 [141]). We suppose that the test lab consists of one or more *devices under test* (DUTs). In our case a DUT is a BGP-speaking device such as a router. To conduct the test we need some sort of *test equipment*. This, for example, can be an Router Tester or a few PCs. Typically the test equipment includes some data traffic load generators. Furthermore, some network *monitoring equipment* (e.g., Endace DAG cards [140]) is commonly used to evaluate the results. (Note that our routing workload generation does not rely on such data traffic generation and monitoring equipment.)

7.1.2 BGP workload generation

This section is divided in two sub-parts: We first provide a rough overview over the tool and then discuss our workload ingredients.

7.1.2.1 Generator tool

To generate our BGP workload we build a tool that generates synthetical traces that can be targeted at the DUTs. Our design goals state that the tool has to be user-friendly, highly flexible and support well-specified test conditions. In other words, it should be easy for a test engineer to instantiate a complex BGP test.

With this we are trying the balance act between a tool that is able to create very specific tests that vendors and operators are asking for, and that at the same time assumes reasonable defaults *if* the test engineer is "a bit vague" in the test specification. Thus, whenever the user does not describe a precise test scenario, our tool emulates "typical BGP" by emulating the dynamics of the outside world. With this we bring some of the variability of today's Internet into the test-lab, without loosing the test engineer in the intricacies of the protocol test setup.

To accomplish this goal, we cannot simply replay recorded BGP update traces, because we still want to be able to respect specific test-settings provided by the user (e.g., experiments with different memory loads of a router). Nor can we derive such a trace by simulation, because it is not possible to emulate thousands of BGP finite state machines on the test equipment. To address those problems we generate a synthetic assembled BGP update stream based on probability distributions. These distributions are used as input to our workload generator and can be derived from a characterization of real BGP traces or constructed manually. As an illustration consider the characteristics of the Routing Information Base (RIB). For the generation process the input-distributions provide the details about how to construct the RIB (e.g., number of prefixes, AS path lengths, etc.). Furthermore, they can be sub-divided into classes at a finer granularity. For example eBGP classes include "customer", "peer", "upstream"; for iBGP those classes reflect the behavior of Route Reflectors; and for Virtual Private Networks (VPNs) they are based on different customer types. While the characteristics of the tables and so the distributions are different, the generation algorithm does not have to change. With that we are able to increase the realism of testing methods and to create scalable workload conditions that reflects todays protocol behavior. Furthermore, a test engineer may approximate growth and evolution by modifying these distributions.

The tool first creates an initial test setup that respects user wishes (e.g., "for BGP I need an exchange point router with 5 peers, running at normal workload"). At the start of the actual test, the DUT is automatically placed in an environment that meets these initial conditions. During the test, BGP updates are created that reflect the dynamics in the Internet with respect to some given metrics, or alternatively conditions specified by the user. The tool comes with a set of *normal conditions*. These reflect the statistical distributions of protocol metrics that are derived from an in-depth analysis of publicly available BGP update streams. Of course these distributions can be varied within or outside the normal range. Therefore such a tool is also able to investigate questions such as: "What if today's Internet grows further?", "How long will my routers be able to cope with that growth?", "When do I have to replace my routers?", "Can I add another full feed to the router at an IXP?", "Can I withstand an attack?", "What about abuse – are my routers prepared?", or "What happens if a disaster strikes? Can I still offer service?".

7.1.2.2 Workload ingredients

Our tool provides *"knobs"*[1] to the test engineer for adjusting the load scenarios. For example the statistical distributions could range from "normal" situations, over "growth" to "disastrous" network conditions[2]. This lets explore the different influences of nearly all parameter combinations in a systematic fashion.

[1] A "knob" is a user selectable BGP traffic generator parameter.

[2] Note that our methodology does not try to trigger specific failures (e.g., router crashes). The design goal is to facilitate the work of the test engineer by generating a workload based on probability distributions.

To find the appropriate "knobs", or *key protocol characteristics*, we identify essential components of BGP traffic. Recall from Chapter 4 that the cause of a routing instability is described by the notion of an *instability creator* or *instability origin* and the temporal characteristic of the resulting sequence of updates by the notion of an *instability burst*. We view the prefixes of the table as nodes in a graph, where edges capture the structure (the nesting) of the prefixes. Regarding BGP attributes, the AS path reflects the interconnectivity of the various ASes and their peering policies. But instead of modeling the AS topology [142, 143] and their peering policies [31, 129, 144] explicitly, we focus on *AS path properties* as seen via a single peering session. After all we are looking for a workload model that can stimulate a system under test or a simulation, but not a full BGP simulation/emulation as for example provided by C-BGP [74].

Our findings from Chapters 4 and 5 show that most instability events cause BGP updates to a number of prefixes at about the same time. But not all of these have to result in an AS path change. Other instability events are of concern to only individual prefixes. While one expects BGP updates to several prefixes if a change to an eBGP session is the instability originator, some or all of the updates may be rather local, e.g., in case they involve only next hop changes. But they can also impose major non-localizable BGP updates, e.g., if AS path changes are involved. This may depend on the specific policy of the AS, the ISP's topology, etc.. To capture the correlations within an instability burst we focus on the *attribute changes* between updates for the same prefix.

Ideally we build our workload model around instability event updates and update sequences. Unfortunately, distinguishing between instability event updates and related updates is an unresolved problem (see Section 5.5.2). But a second look reveals that to a system under test it does not matter if the update burst is the result of one or *n* instability events. What matters is the number of updates it has to handle and the relationship between the updates. Thus each instability creator is either generating a single update burst, in case of a prefix, or a set of update bursts, in the case of an AS. For example, we can express BGP protocol divergence [124] for a prefix as a single update burst that lasts for the duration of the test and consists of a large number of updates.

In summary, our "knobs", provide the *input variables* for our tool. They include for example: memory consumption of the RIB, number of sessions, neighbor type, number of prefixes total or per session, number of updates the FIB change rate. Note that some of the input variables are mainly metrics, such a the memory consumption of the RIB, which can only be measured and are difficult to set. Therefore in the next section we look at the requirements that for out tool.

7.1.3 Requirements

We have two important requirements that must be met before a test can be preformed: First, the DUTs have to be correctly configured. This can either be done manually by the test engineer or automatically by our script. Secondly, we need to solve all dependencies between the input variables.

7.1.3.1 Manual vs. auto-configuration

The device under test has to be properly configured. No BGP session can establish between the DUT and the test equipment, unless both sides are properly configured. While we assume

that our script has full control over the test equipment, it may not have **any** control over the DUT. As it is very inconvenient for the test engineer to setup a potential large number of BGP sessions manually, we offer a mechanism to automatically configure routers[3]. If the user chooses to configure the DUTs automatically, then the script has to have the information so that it can login to the router.

7.1.3.2 Dependencies of variables

Ideally, all our variables should be independent. Although the variables are chosen and the algorithm is designed with this goal in mind, this is not always possible. Some of them are dependent on each other and some on the hardware. Consider the following example: one of our metrics is "memory consumption of the RIB". A test engineer may ask for a memory consumption of 200 MB, but only wants a total of 10 prefixes in the routing table. Clearly this is not possible, because RIB memory consumption, number of prefixes and number of peers are strongly correlated. Yet, sometimes a user asks for one variable as input, while in another context another variable is more important for the test. Therefore, we have to offer a larger spectrum of variables to allow for flexibility. But once one variable is set, the ranges of values for other parameters may be reduced. As this reduction may depend on the specific hardware our script may have to calibrate the workload with the router architecture.

To deal with metrics as input we have to perform a calibration phase to assure that the parameters – specified by the user or provided by the distributions – are actually "supported" on the DUT. The calibration phase is also needed to determine the range from which the algorithm can pick values. This approach allows for a very flexible specification of user requests, but has the disadvantage that some variables can become dependent. Given the inter-dependence of controlled variables, the generator needs to have the leeway to delimit, whenever necessary, some or all other values. In practice this implies that only some of the user selections can be respected. Note that a user specification of a variable always overwrites the probability distributions derived from observed data.

7.1.4 Summary

We outline the design for a tool that can help network operators in a wide variety of equipment testing: testing software implementations, performance, scalability, data-plane convergence, and benchmarking. A "normal" or "initial" setting for the workload generator is derived from publicly available traces. But they can also be derived from measurements at specific locations (e.g., a collected trace at a specific route reflector), or they can be modified to reflect network growth. Furthermore, in the future the algorithm can be updated with a new set of "default" settings.

The design goal of the algorithm is that whenever the test engineer is "a bit vague" in describing his/her test setup, a value will be picked in accordance to the distributions. With this approach, the test setup gets easier (for the test engineer) and at the same time we bring the dynamics and variability of the real Internet into test labs.

[3]This is based on the RANCID [64] tool and we currently support only Cisco routers. Juniper is forthcoming.

• memory consumption for RIB[4] `(in MB and/or number of prefixes)`
• BGP neighbors `(number and neighbor type)`
• instability event rate `(events per minute)`
• update rate `(updates per minute)`
• update impact percentage[4] `(percent and number of FIB changes)`

Table 7.1: Key variables for today's router architectures.

• Packet forwarding latency `(in microseconds)`
• Switch-over-time `(in microseconds)`
• Convergence time `(in microseconds)`
• Packet loss and misrouted packets `(in number of packets)`

Table 7.2: Key metrics of today's router architectures.

7.2 Test metrics

This section discusses the variables and metrics that we consider to be the key ingredients of our workload generator. We start with those variables that we consider to be most important for our tool, and then explain all other turnable "knobs" that an test engineer can influence. The next section (Section 7.3) provides more details about the design and the realization of an algorithm that fulfills the design goals.

7.2.1 Key variables

The main challenge of this work is to find a good trade-off: on the one hand our goal is to allow a large number of variables to influence the test scenarios, but on the other hand there needs to be a realizable algorithm, which is capable of resolving dependencies.

Therefore, as a first step, we identify key metrics and variables relevant for today's router architectures, see Table 7.1. Note that some variables are on a per-session basis (e.g., number of prefixes), others on a per-router-interface basis (e.g., changes in the Forwarding Information Base (FIB)), others on a per-DUT basis (e.g., memory consumption) and some even applicable to DUTs involved in the test (e.g., total number of prefixes).

In addition, Table 7.2 lists metrics that are crucial for today's router architecture, but which have too many influencing factors. Therefore we do not consider them as input variables, yet we recommend that those variables should be measured in appropriate tests. This includes the one-hop packet forwarding latency, switch-over-time, convergence time, as well as packet loss and misrouted packets. The one-hop packet forwarding latency $t_{latency}$ is defined as:

$$t_{latency} = \frac{1}{n}\sum_{i=1}^{n}[t_b(i) - t_a(i)]$$

where $t_b(i)$ is the time at which the i^{th} packet was observed on the outgoing link and $t_a(i)$ is the time at which the i^{th} packet was sent to the DUT.

The switch-over-time is the time difference between the moment that the router receiving a new BGP update, which is going to "win" in the best path selection process on one interface,

[4] Note that memory consumption and number of FIB changes are metrics but for our algorithm they are also considered as input variable.

until the moment that the traffic actually start flowing out of that interface. This means the switch-over-times, $t_{switch-over-time}$ is defined as:

$$t_{switch-over-time} = \frac{1}{n}\sum_{i=1}^{n}[t_O(P_i) - t_A(P_i)]$$

where $t_O(P_i)$ is the time at which the first packet with destination prefix P_i was observed on the link where P_i is announced, and $t_A(P_i)$, the time at which prefix P_i was announced. Note that is is very similar to the definition of convergence time. Convergence time has two components $t_{announcement\ convergence}$ and $t_{withdrawal\ convergence}$:

$$t_{announcement\ convergence} = \frac{1}{n}\sum_{i=1}^{n}[t_{\bar{O}}(P_i) - t_A(P_i)]$$

where $t_{\bar{O}}(P_i)$ is the time at which the first packet with destination prefix P_i was observed, and $t_A(P_i)$, the time at which prefix P_i was announced. Contrary to the switch-over-times, this assumes that either P_i was not in the RIB, or that the FIB was re-computed. The second component is defined as:

$$t_{withdrawal\ convergence} = \frac{1}{n}\sum_{i=1}^{n}[t_{\bar{\bar{O}}}(P_i) - t_W(P_i)]$$

where $t_W(P_i)$ is the time at which prefix P_i was withdrawn and $t_{\bar{\bar{O}}}(P_i)$ is the time at which a data traffic stream with destination P_i was *not* seen for a given time period.

A test setup can always measure lost and misrouted packets. The number of lost packets $n_{lost\ packets}$ is defined as:

$$n_{lost\ packets} = \frac{1}{n}\sum_{i=1}^{n} n(P_i)[t_O(P_i) - t_A(P_i)]$$

where $n(P_i)$ is the number of bytes per second generated for a data traffic stream with destination P_i.

Correspondingly the number of misrouted packets $n_{misrouted\ packets}$ is defined as:

$$n_{misrouted\ packets} = \frac{1}{n}\sum_{i=1}^{n} n(P_i)[t_{\bar{O}}(P_i) - t_W(P_i)]$$

Let us revisit Table 7.1 in more details: The purpose of a routing protocol is to find the "best" path through the network to a destination. Especially in times of changes, a fast convergence is needed to fulfill that goal. Leaving aside protocol-dependent timers that slow down convergence, there are two essential resources that govern convergence: memory and CPU availability. The total number of prefixes that needs to be stored on the DUT affects memory consumption. In addition, the number of neighbors has a strong influence. BGP-generated CPU load is related to update frequency and update impact. An update's impact is the amount of work that needs to be performed by the central CPU to process it. This ranges from low (e.g., immediate discard due to filtering) to high (e.g., FIB is changed and an update is propagated). Furthermore, a large number of FIB changes may increase latency and impose packet loss. Note that this information is not in publicly available BGP traces, therefore it is very difficult to estimate a number of FIB changes by data characterization. This underlines once again that observable changes in publicly available update files do not necessarily reveal those metrics that trouble a router or the network.

Note that memory consumption and update impact depends on previous calibration. If no calibration is possible or if the user does not specify those variables, their values is **not** determined by a statistical distribution, but instead is the result of the generation process of other input distributions.

Having discussed the key variables, we now consider all details that we need to create the synthetic workload. The prefixes and the attributes reflect the virtual network environment in which the router is placed (Section 7.2.2). The updates reflect the dynamics which the router is subjected to (Section 7.2.3).

7.2.2 RIB construction metrics

Consider an example how a user may chose different kinds of RIBs. First, we assume that the test engineer starts with the "knob", memory consumption. There he/she can specify approximately how much memory should be used for the RIB and what percentage should be used for prefixes vs. for attribute values. Next the user may specify how many and what type of peering sessions he wants from a range of plausible values.

Alternatively the user can start by specifying the total number of prefixes that the RIB of the DUT should contain. The default value is the number of prefixes in the default-free routing table of the Internet. Next he/she will have to specify the number and types of BGP peering sessions. The type of peering session determines for how many prefixes updates will be sent across that session. This impacts the DUT since the DUT has to store the reachability information. If desired, the user can specify the number of prefixes on a per session basis.

This section outlines the parameters that impact the generation of a RIB in our workload model. Our algorithm changes the memory consumption on the router via the number of prefixes, the length of the AS path, the number of communities attached to a prefix, the diversity of the attributes and the overlap between the sessions. To realize this we distinguish several classes to model these characteristics (such classes could be for example "small customers", "large customers", "peer", "upstream", etc.).

To model this RIB construction, we start with a concept of a *"globally advertised address space"* [145, 146] [5]. There are several ways how prefixes may materialize inside the RIB of a router. Providers which offer *full-feeds*[6] to customers claim that they can carry the traffic to all addresses advertised in the Internet. A peer/customer typically limits the announced address space to its own and customer routes of the peer/customer. To reflect this in our workload model we use the notion of *overlap*. With overlap we mean two or more sessions that announce the same prefix[7]. The overlap of prefixes between two full feeds is typically large, the overlap of peering sessions is low (unless there are two peering feeds receiving routes from a multihomed customer of those two peers). The overlap between two customer sessions is almost non-existing.

Next, consider iBGP. There are several ways to set up iBGP inside a network. One is in a full-mesh. In this case every BGP-speaking router must be connected with every other BGP-speaking router. Another possibility are route reflectors (RRs). A BGP speaker connects only to a RR[8] and receives a summarized view of the routing information. Note that Route Reflection may include multiple hierarchies.

The main difference for our testing purpose is that a RR only forwards its best routes to a RR-client and other RRs. Therefore the distributions that generate the RIB (and the update streams as well) have to reflect the behavior of the session at each RR level. Thus we need classes of distributions that grasp the properties of the RR-clients, and the characteristics between RRs – if multiple levels are present, then each hierarchy has to be modeled by a separate set of distributions. This means again, while our basic algorithm that creates the

[5] Note that we ignore the tiny fraction of "dark address space" [88]. that is observable in the real Internet.
[6] A set of BGP routes that provide reachability to all advertised address space in the Internet.
[7] Of course only the prefix has to be the same, BGP attributes can differ.
[8] Typically it connects to more than one RR for reliability.

prefixes for the routing table remains the same, the characteristics of the table change at the various levels of RRs. This is reflected in our algorithm by using a different set of distributions that creates the workload similar to the observed properties at the corresponding place in the real network.

Finally, for mpBGP, we have to go one step further. Here is an identifier attached to each route, enabling a distinction between various VPNs[9]. Each VPN is associated with one or more VPN routing/forwarding instances (VRFs). The route distinguisher is used to place "bounds" around a VPN so that the same prefixes can be used in different VPNs without conflicts. This requires us to pay some extra attention to VPNs. Again, classes of VPN customers can be constructed that reflects the properties of the tables (and update dynamics) of VPNs.

Let us consider the metrics and their inter-dependencies in more detail:
eBGP: Our approach starts with generating the "globally advertised address space" and then assigns a subset of the IP address space to each neighbor based on its type, e.g.,

- "Customer" (small # IPs: 1-1,000 prefixes)
- "Peer" (medium # IPs: 1,001-90,000 prefixes[10])
- "Upstream" (large # IPs: 90,001-∞ prefixes)

To keep our model simple and scalable, we recall what matters for a router: the DUT needs to represent and store the RIB in a certain data structure – and this uses memory on the router. Hence what consumes the memory of a router are the number of BGP neighbors and the type of routing tables that they send. Filtering of more specific prefixes may reduce memory consumption while maintaining a similar reachability (in terms of address space coverage). Table 7.3 summerizes the variables that influence the eBGP table generation process.

• BGP neighbors `(number and type)`
• RIB memory consumption `(in MB)`

Table 7.3: Variables influencing eBGP.

iBGP: Table 7.4 shows that we need a set of distributions for each level of route reflection. Note that this means that the content of the distributions, (such as number of prefixes, AS path length, usage of communities, etc.), needs to be generated from traces collected at the particular RR or at a RR-client. Unfortunately, such data is currently not publicly available; therefore our script does not provide defaults for this. This means that before using iBGP, the test engineer has to collect traces from RRs in their own network and run the data characterization algorithms to derive the distributions.

• Multiple hierarchical levels of route reflection

Table 7.4: Variables influencing iBGP.

[9] VPN customers carry their own routing tables which are separated from the Internet and other VPN customers. To be able to assure that the same prefix can occur in multiple VPNs, and still is treated separately, it gets tagged with an additional 8 byte route distinguisher.

[10] Note that a peer that announces 40,000 or more prefixes over a peering session is quite rare (e.g., only tier-1's). After all this is $\frac{1}{4}$ of the total address space and who has $\frac{1}{4}$ of the whole Internet as customers?

VPNs: Table 7.5 illustrates how we treat VPNs. Again, what matters for our purposes is the memory load of the router. Thus we consider the fact that each VRFs table consumes memory and needs to be stored separately.

Each VPN customer in our model is treated as a differently composed set of distributions, this assures that each VPN customer can reflect its unique characteristics. Of course, distributions can be aggregated into classes of distributions (e.g., "small-sized VPN customers", "customer carrying a full Internet table inside its VPN").

• type of VPN neighbor.

Table 7.5: Variables influencing VPNs.

Note that we treat IPv6 prefixes in the same way: the table characteristics are determined by another set of distributions. Yet the generation process of prefixes for IPv6 follows the same mechanism (of course with a 128-bit prefix).

7.2.3 Update generation metrics

In this section we describe how we impose the load on the DUT with BGP updates that are consistent with the kind of instability events observed in the Internet. For this we use the notion of the "instability creator" (see Section 4.1 on page 49 for more details). Each instability creator affects multiple prefixes on a subset of the BGP sessions.

For our workload generator we respect the fact that not all instabilities propagate to all peers on which the prefix is announced. Rather an instability only propagates within a certain *"sphere"* of the AS topology. Furthermore, BGP signal propagation through the topology takes some time, for example due to the MRAI timer or route flap damping. This is addressed by the concept of a *"phase shift"*. Finally, BGP may show multiple updates for one prefix on one peering session that was triggered by the same instability event. Such an *"update cluster"* generates potentially multiple updates for each prefix on each peering session within the *sphere*. See Table 7.6 for a summary.

• an instability creator produces *sphere* and *phases shifts* (Section 7.2.3.1)
• an instability creator produces a set of update clusters (Section 7.2.3.2)

Table 7.6: Variables controlling instability creators.

7.2.3.1 Sphere and phase shift

In general the propagation of BGP updates through the Internet follows a directed acyclic graph. Hereby one has to keep in mind that BGP updates start at the "origin of the instability" and will only be propagated if the best route gets changed. Due to the denseness of the current Internet topology this has an impact on those ASes that will observe the effects of this instability event. We call those ASes that are affected by a certain instability event, to lie within the *sphere* of the event.

For our workload model this means that we sub-select some fraction of the prefixes and induce updates on these prefixes within a subset of the peering sessions. We may have to bias

this sub-selection so that a possible specification regarding the impact of the updates on the DUT can be satisfied.

Furthermore, we have to consider that the propagation through the *sphere* of the instability can be surprisingly long. Recall the effects of the MRAI timer from Section 2.1.1 (page 14), Section 4.7 (page 4.7), and Section 6.1.2 (page 6.1.2). This timer limits the update rate for announcements to each eBGP peer to a typical value of 30 seconds. At each route reflector inside an AS the propagation of updates are delayed by 5 seconds. Additionally route flap damping can suppress propagation for an extremely long time. Table 7.7 summerizes the relationships between different peering session on the DUT for the same instability event.

- the *sphere* of a cluster propagation is determined by a distribution
- for a given prefix the arrival time *phase shift* on any two peers is determined by a distribution

Table 7.7: Variables influencing update spread.

7.2.3.2 Cluster generation

BGP may send several update messages for a single triggering event [11]. Such a train of BGP updates triggered by one event is called an *update burst*, see Section 4.4 on page 54. Various timers, such as MRAI and route flap damping, cause clearly identifiable patterns within such an update burst. The goal is to create synthetical update bursts that reveal a behavior similar to the behavior of update bursts in the real Internet. This include that our workload model has to account for the fact that such a burst can have a heavy-tailed number of updates, and that update inter-arrival times follows a distribution (see Chapter 4).

Furthermore, we have to assure that additional characteristics are met, like ratios of announcements of withdraws, type of changes. Thus we decided to account for the following variables shown in Table 7.8.

- a cluster is a set of updates
- cluster intra-arrival-time follows a distribution
- the number of intra-cluster updates follows a distribution
- announcement & withdraw ratio
- last update in cluster is a reachable / unreachable prefix

Table 7.8: Variables influencing update bursts.

7.2.3.3 FIB changes

So far we have constructed a workload model that is consistent with the key factors that can be observed in real-world BGP traffic. The remaining metric that needs to be address is the “impact” of a BGP update on the DUT. A BGP update may cause (a) a FIB change and additional updates to its neighbors, (b) a FIB change, but no additional updates, (c) no FIB change, but additional updates, (d) just an update of the RIB, but do not result in FIB changes nor in additional updates, or (e) it might be filtered beforehand. Switch-over-times

are actually crucial metrics in the performance of a router. Because this metric is difficult to derive from publicly available traces, we only consider this metric if the user requested a specific rate. Otherwise it will be the outcome of the other distributions above. If a specific rate is requested, we create, during the construction of the update bursts, the changes in a particular way that depends on how many FIB changes, and/or update propagation events the user is looking for. If not enough FIB changes can be created in the current time block, then more events are added. See Table 7.9 for a summary of the specifiable parameters that influence the FIB changes. Note the currently version of the algorithm does not allow the user to specify a percentage of the FIB entries that is supposed to change. This is future-work.

• number of FIB changes per time unit • *% of the FIB entries that change*

Table 7.9: Variables influencing FIB changes.

7.2.4 Summary

In this section we identified a set of "knobs" that allows a test engineer to specify the characteristics of the generated BGP workload. We designed the variables and metrics around the characteristics of observable routing traffic in the Internet, so that the generated workloads are useful for testing today's and tomorrow's router architectures, and also respects the wishes of a test engineer.

7.3 Algorithm

In this Section we describe an algorithm that can test multiple devices under test and runs multiple consecutive tests automatically (e.g., regression tests). Figure 7.2 (see page 94) gives a rough overview over the algorithm. The algorithm is designed in a modular way, so that it is possible to add new variables as new router architectures may require other test metrics.

A prototype implementation of this algorithm is called **syn.pl** and is written in PERL. The prototype itself consists of a calibration phase and a test phase.

The calibration phase assumes, as indicated in the previous section, that DUTs have to be configured properly. If a DUT is not already configured, we offer an auto-configuration mechanism. This logs into the router and sets-up the appropriate BGP sessions automatically. This feature is very valuable, especially for tests with several hundreds of sessions. Furthermore, before staring the test, a calibration may be necessary to determine the range of appropriate parameters. Our script starts with identifying the initial settings and the executes a calibration phase for the equipment. The main purpose is to construct appropriate routing tables (see Section 7.3.2).

Then, the tests are performed – one after the other (see Section 7.3.3). This part relies on the information gathered by the calibration phase and consists again of three parts: (1) The sources of the updates. BGP updates need to be generated based on the distributions. Additionally, it is possible to merge the synthetically generated update stream with a recorded trace from disk. That is the reason that we allow multiple independent input sources (see

Section 7.3.3.1). (2) the sources have to be mixed together and the traces for the appropriate devices under tests are modified (e.g., adjusting next-hop, test- and network-equipment IPs to the corresponding session parameters). Furthermore, a trace can be scaled or otherwise modified (see Section 7.3.3.2). (3) Then the traces are ready to be sent to the DUTs. The test equipment that is conducting the test consists of several units (e.g., Agilent Router Tester, load generating PCs, etc.) or the trace may be written to disk for later playback. Our prototype currently pre-generates the full trace, transfer it to the replay equipment and conduct the test – but the ultimate goal is to create the updates on the fly while performing the test (see Section 7.3.3.3).

All this needs to be controllable by the user. XML-configuration files provide the necessary interface between the user and the script as presented in the next section (see Section 7.3.1).

7.3.1 XML-Configuration language

The goal is to decompose the inputs into logical configuration files so that a clear separation of the entities in the test-lab can be achieved. Our system is driven by five databases. The first three take stock of the testbed's inventory, which consists of routing & test equipment and connecting cables. Accordingly, *network-equipment.xml*, captures the hardware setup of the devices under test (DUTs) in the test-lab (e.g., routers/interfaces, access mechanisms, etc.). *test-equipment.xml* specifies the available test components (e.g., Agilent Router Testers, PCs), and *topology.xml* provides a description of how the test equipment is connected with the devices under test (e.g., physical cables, VPNs). Furthermore, *distributions.xml* provide a mechanism to specify the characteristics of the synthetically generated workload that is supposed to be used to stress the DUTs. While *test.xml* gives a description of the tests to be performed. *test.xml* references the other databases. Here is a short overview:

Content	*Sample filename*	*Description*
Devices under test	network-equipment.xml	Setup of the DUTs in the test-lab.
Test equipment	test-equipment.xml	Setup of the test-hardware.
Link-level topology	topology.xml	Connections between interfaces.
Distributions	distributions.xml	Math. or empirical functions.
Main configuration	test.xml	Definition of test parameters.

The choice of splitting it into five databases is motivated by the rate of change to their content. First consider the physical hardware. New devices are rarely added to the test-lab, their setup and interfaces will not change very often, but the details of the available routers are essential for the tool to be able to perform the tests. Therefore providing a description of each device under test has to be done – but has to be done only once for each device (*network-equipment.xml*). The same holds for the test equipment: once bought, setup, and configured, it should be ready to use (*test-equipment.xml*). The interconnecting cables between the DUT interfaces may actually vary much more frequent for example by recabeling, or changing the VPN setup of intermediate switches, or simply by putting an interface within another logical subnet. For this reason we "outsource" this part (*topology.xml*).

Next, the workload that is supposed to be generated may be derived from public sources such as [48,49] – or may be extracted from proprietary data collected at a certain peering point that reflects a "typical" situation which is supposed to be reconstructed in the test scenario. Multiple distributions may be collected and exchanged for variables depending on the question that is under investigation in a certain test situation (*distributions.xml*).

And finally, the specification of how to perform the actual test, is the key element the test engineer will work with (*test.xml*). Our configuration "language" is designed to make this part very flexible and easy to adjust.

Now, we briefly illustrate the makeup of our configuration language with the following example: We assume that the testbed already consists of one properly set up router as DUT, a Cisco GSR 12008 [54], and an Agilent Router Tester 900 equipped with an N2X XS Test Card [111]. Furthermore, assume that the test engineer plans to conduct a single, rather simple test, that uses only BGP traffic, and no data traffic. There are supposed to be two BGP sessions on the DUT: one synthetically generated session, referred to as `full-feed` with a user-determined number of 180,000 prefixes; and one session[11], referred to as `customer`, that replays a pre-recorded BGP data stream (from disk located in `data/rrc00/...`).

A *test.xml* configuration for our example looks like this:

```
<test>
  <test-run>
    <mytest1>
      <dut use="c12008">
        <configuration-reload use="config-3"/>
        <link use="c12008 FastEthernet1/0-rt 101/4">
          <bgp>
            <full-feed>
              <number-of-sessions> 1 </number-of-sessions>
              <number-of-prefixes> 180000 </number-of-prefixes>
              <as-path-length use="my-modified-distribution"/>
            </full-feed>
            <customer>
              <data-set>
                <rib-preload>
                    data/rrc00/bview.20050301.0000.gz
                </rib-preload>
                <bgp-update>
                    data/rrc00/updates.20050301.*.gz
                </bgp-update>
              </data-set>
            </customer>
          </bgp>
        </link>
      </dut>
    </mytest1>
  </test-run>
</test>
```

Here, `<test>` is the top level structure in this database, `<test-run>` is the tag that allows the test engineer to specify the tests. Within `<test-run>` multiple tests can be defined, in the example above only a single test is defined, called `<mytest1>`. Everything encapsulated within this tag describes a single test run and is executed in parallel. The workload generation can include consecutive tests. `<dut>` groups all "instructions" together that apply to a specific device under test. Of course, different workloads can be applied to different routers. To indicate which router is the target, the XML-attribute "*use*" is used. Here, *use="c12008"* refers to

[11] If more than one session is recorded in the trace, then all sessions can be played back or a sub-set can be selected. Commands for this are not shown in the example.

a definition of a *dut* that was "labeled" with *c12008* in the XML-files. Our notation for defining such a reference is: `<dut name="c12008">`. Our tools interpretes such entities with a *name*-attribute as macro definitions. They can be referenced across all XML files. One property of such a macro is that all sub-elements of that object are "inherited". To continue in our example, the specification of *dut* may contain various mechanisms to reinitialize the router, including an inherited default. But the user can overwrite this default, redefine or disable certain tags. For example the instruction `<configuration-reload use="config-3"/>` initializes (reboots) the router with "config-3". Note, that either the reload mechanism has to be described inline (e.g., if *tftp* should be used, where to find the configuration file, etc.) or the mechanism has to be pre-configured. The latter is the case for our example.

The next tag in our example is: `<link use="c12008 FastEthernet1/0-rt 101/4">`. This is again a reference, but this time to the topology-section. It specifies the interface of the DUT and also indicates to **syn.pl** which parts of the test equipments has to perform the test.

Finally, the `<bgp>` tag configures the protocol specific details for the test. Other protocols, such as OSPF, may be added here in future work. In the case of BGP, a single router typically carries multiple BGP sessions. In the example above one session is called `<full-feed>`, the other is called `<customer>`. As our algorithm generates the updates for each session differently, we need such a sub-structure. Note that the actual tag names are of minor importance. Instead of creating "useless" grouping tags we allow the engineer to give user-defined tag-names. This may help in debugging or structuring the configuration. The concept of such user-defined tag-names was already introduced on the `test-run`-level: `<mytest1>` can be renamed to any other tag-name that captures the propose of the test in the mind of the test engineer. In fact the concept of *user-defined levels* that group sets of pre-defined tags together is a key ingredient to our framework. Thus at any level, the user may introduce tag-names as long as they do not conflict with pre-defined tag-names or with the state-machine of the parser. With such a mechanism it is possible to group certain tags together that support the user.

This is especially useful in conjunction with the following advanced feature that allows flexible specification of parameters: In the above example each BGP session and all its parameters had to be explicitly configured. Yet in the end the tool needs to know which part of the test equipment is supposed to stress which interface of a DUT. Here again, it is desirable to be "a bit more unspecific" and let the software take appropriate actions to achieve the requested workload. Therefore, it is possible to specify tags at higher levels. To illustrate this, consider the example above, where the user changed the characteristics of the session `<full-feed>` by specifying `<as-path-length use="my-modified-distribution"/>`. This causes the AS path length to be generated according to the distribution specified in "my-modified-distribution". While this allows us to be very flexible, it has the disadvantage if many sessions are supposed to be generated according to a given set of distributions. Therefore it is desirable to specify tags at higher levels, including the *dut* or even the *test-run* level. Thus, if tags are specified at a different level, it will be applied to **all** appropriate configurations. In the above example this means, if the user moves the tag `<as-path-length use="my-modified-distribution"/>` to the `<dut>`-level, that the AS path length distribution will be applied to both sessions, `<full-feed>` **and** `<customer>`. Note that in combination with user-defined levels this provides a powerful specification mechanism for the test engineer.

Next let us take a look at the other XML configuration files. We start with the corresponding *network-equipment.xml*. Note that we will not discuss all commands here. The goal is to

highlight some of the capabilities. For example *method* specifies how to access the router by using the URL format to define protocol, user, address and port. It is assumed that management interfaces are properly pre-configured to support the access method.

```
<network-equipment>
  <dut name="c12008">
    <access name="telnet">
      <method> telnet://10.4.3.1/ </method>
      <passwd> ocsic </passwd>
    </access>
    <interface>
      <name> FastEthernet1/0 </name>
    </interface>
    ...
  </dut>
</network-equipment>
```

In the same testbed we also have a Router Tester with one module comprising four fast Ethernet interfaces. Note that *host* below can be used to *ssh* to a PC for performing the test, or like in the example it specifies the Router Tester controller. The corresponding *test-equipment.xml* is:

```
<test-equipment>
  <tester name="router tester 1">
    <host> rt </host>
    <interface>
      <name> 101/4 </name>
    </interface>
    ...
  </tester>
</test-equipment>
```

Topology.xml documents the testbed cable connections and assigns the infrastructure addresses. Following the above example, we specify a cable from port 4 of our Router Tester's module 101 with the GSR's FastEhernet1/0 interface. A *link* tag defines two directly connected end points, one belonging to the router, the other to the router tester. The router end is tagged *dut* and the router tester end, *tester*. Both tags reference *name*s defined at different places.

```
<topology>
    <link name="c12008 FastEthernet1/0-rt 101/4">
        <dut use="c12008">
          <interface use="FastEthernet1/0">
            <addr> 10.1.3.1/24 </addr>
          </interface>
        </dut>
        <tester use="router tester 1">
          <interface use="101/4">
            <addr> 10.1.3.2/24 </addr>
          </interface>
        </tester>
    </link>
</topology>
```

In summary this approach facilitates the specification of regression tests and let a user easily explore the impact of various parameter combinations on the DUTs. At the same time it allows a very detailed specification of of the tests. Actually, this is crucial for a test-bed configuration language: to be able to reuse a complex specification and just change a few parameter values.

7.3.2 Initial settings

DUT memory consumption for a given protocol depends, at a minimum, on its underlying operating system and version. Accordingly, a RIB that indeed uses the user specified amount of memory cannot be constructed without a previous calibration step.

7.3.2.1 Calibrating the RIB

In the first step of the calibration phase we have to generate a RIB. This has to be done independent of a possible user specification. Note that the user can pre-load a RIB from a file and may ask the tool for some modifications (e.g., add prefixes accordingly to the configuration specifications).

The largest contributers to memory use are: number of BGP sessions and number of prefixes. As the number of BGP sessions has to be specified by the user for each test, the algorithm can only modify the number of prefixes and their parameters. The number of prefixes is used to catapult us in the overall region in which the memory consumption resides. Fine-tuning is done with the BGP attributes including AS path length, number of communities, etc..

The calibration consists of a limited number of download steps of the generated routing tables to the DUT. We measure the corresponding memory usage via SNMP or via the CLI.

There are several issues with the calibration approach: Router management processes run at regular times (such as garbage collection tasks), but not necessarily in synchronization with our calibration software. For example imaging a calibration with a large routing table, the table is generated, downloaded to the router and considered to be too large. The session is teared down, and a smaller routing table is generated and downloaded to the router. To estimate the memory consumption of the routing table the memory usage of the router is measured before and after the table is downloaded. But what if some “old” memory chunks are freed during the downloading of the new RIB? This can mislead the calibration phase. This particular problem exists in two ways: First one calibration step may influence the next calibration step and second two calibrations with the same parameters can result in different memory consumptions.

7.3.2.2 Details of RIB construction

Next we discuss how the RIB itself is created by the algorithm. First, if the user specified a RIB pre-load, this file is used as a basis. Then prefixes are added in several steps. We first identify the prefixes contained in the RIB and also determine the attributes that are consistent across all peering sessions, e.g., origin AS (Section 7.3.2.2). Next to select which prefixes of

- Minimum and maximum number of prefixes to generate (determined by various configured parameters, such as neighbor-type, etc.).
- Diversity of attributes ("equality" of BGP attributes across prefixes):
  - This parameter determines the maximum number of prefixes that can be packed in one BGP update. Note that a small number makes the attributes more divers, because many different attributes will be generated. A large number will generate a lot of prefixes with the same set of attributes.
  - This parameter determines for how many BGP update packets the AS-path and the community string should not change. This is a typical behavior in the Internet that many prefixes have the same (or very similar) attributes. Router code optimizes for this and is testable with this option. Note that a high value in this parameter will reduce memory consumption.
- Overlap to *globally advertised address space*[12]. (this will influence the number of prefixes per session.)
- AS-path length[13]
- Communities[13]

Table 7.10: Parameters used to generated RIB

the RIB are announced over which peering sessions (Section 7.3.2.2) and finally to determine their specific attributes, e.g., the full AS-path, community strings, etc. (Section 7.3.2.2).

Currently the RIB generation process depends on the variables summarized in Table 7.10. Note that those variables are used to construct the synthetical part of the RIB. If the user requested a certain memory consumption those parameters will be altered until the specified memory consumption is reached.

Now that we have discussed the basic ingredients for the RIB construction we take a closer look at details of the prefix generation (Section 7.3.2.2), the distribution of prefixes to sessions (Section 7.3.2.2) and the generation of BGP attributes (Section 7.3.2.2).

Prefix selection

This part discusses how we add synthetically generated prefixes to a pre-loaded RIB from a file. Note that if no RIB files are pre-loaded, then the RIB is empty. Thus, the goal is to generate a set of n IP prefixes out of an IP range whose prefix length and prefix nesting distribution is consistent with some distribution, determined via characterization of BGP tables or specified otherwise.

This is done by first computing how many prefixes for each prefix length at which nesting level are required. For this we first loop over all nesting levels and in each nesting level over all prefix lengths (starting at /8's and going to /32). We get an value from our distribution and multiply it with the number of prefixes that are supposed to be generated.

Next we have to understand how much address space those prefixes cover. This is especially important for more specific prefixes, because our algorithm first tries to fit all prefixes into the same super-prefix. Therefore we first try to find a prefix on the corresponding nesting

[12] Note that the overlap percentage between any two sessions on the DUT cannot be configured.

[13] *AS path length* and *communities* refers to a numerical value that is being added to the picked value from the distribution.

level (current nesting level - 1) that could carry all prefixes that should be generated. For example, if we are going to generate 3 more specifics of a prefix length of /24 we need at least a /22 to be able to fit those prefixes on the corresponding nesting level. Then we look for the smallest super-network that is able to cover this set of prefixes. If it is not possible to find such a super-net that can carry all more specifics that should be generated with the current length, then we try to split the prefixes on multiple super-nets. If even this fails, we disrespect the nesting (because the number of prefixes requested by the user is more important than the nesting structure of the prefixes).

A limitation of this approach is that to some degree the prefix length/nesting distribution depends on the number of prefixes. This is because IP address distribution policies by the regional route registries change over time. For example, formerly /8 networks were assigned according to the classful address space. In the time of the "dot-com" boom traffic engineering via more specifics was heavily used [89]. Nowadays very restrictive policies are used. This interplay is captured in a snapshot in the distribution. If a user chooses a higher number of prefixes, we scale the distribution. That means for different types of networks a distributions may have to be regenerated based on newer or more accurate data.

Finally, we choose prefixes. This means we need to generate a 32-bit or 128-bit "number" (IP address) that is not already "in used". Note that a prefix can be "in use" because it was already generated or this address space is already covered by the pre-load RIB. There are two possibilities: Either we pick a random number or we choose consecutive prefixes. The advantage of choosing consecutive prefixes is that the algorithm does only have to check for conflicts with the pre-loaded RIB and therefore the prefix generation is faster.

Prefix to peering sessions distribution

Next the prefixes are assigned to the sessions. First the pre-loaded RIB files are applied to "their sessions" (each RIB file is associated with a session via user specification). Then, we use the overlap distribution. Recall, that overlapping tells us the percentage of prefixes from the generated set of prefixes that is supposed to appear on that particular session. Consider the example "upstream": A "provider" is expected to announce a full routing table to the router. As no full routing table contains all prefixes we select a subset of all prefixes at random. We pay attention that we select **all** prefixes of depth 0,[14] and only "filter" more specifics (e.g., only $90-95\%$ of depth 1, $80-90\%$ of depth 2, etc.). In addition, if a prefix is nested at depth x and the corresponding prefix at depth $x-1$ was not selected this prefix is also excluded. This corresponds to the following interpretation: A provider is able to propagate routes to all of the reachable address space but some of the more specific routes are not announced to everyone.

As an additional consideration we need to make sure that a certain fraction of the prefixes is reachable via a certain fraction of the interfaces. This is necessary to assure the requested number of FIB changes. To induce a FIB change we must use a prefix that is announced via multiple BGP sessions that are located on different interfaces of the DUT. Therefore we prefer to distribute prefixes to those sessions that are on different interfaces of the DUT.

[14] Note this does not have to be true all the time in real Internet. Consider an aggregated prefix on one session and several deaggregated prefixes on another session. Both provide the same address space coverage but with a "different" set of prefixes.

Note that in a future version of this algorithm we plan to allow that a certain percentage of the FIB can be changed (either caused by an instability event or by user request). To realize this we have to ensure that this percentage of the FIB actually can be changed. Thus prefixes for this fraction of the FIB must be available on different interfaces of the DUT.

Attribute selection: per prefix and per session

In the next step each prefix needs to receive an AS path as well as other attributes. In the first release we mainly consider AS path and communities. Communities provide a nice way to increase memory consumption on the routers. AS path are used to group prefixes and to induce instability events. Accordingly we group prefixes that effect the same subsets of peering sessions together. Otherwise the AS path length is chosen according to an appropriate distribution and filled with random AS numbers. While a table is constructed we build statistics across the AS path and sub-select a number of candidate ASes to be used for the instability events.

Below is a detailed list on how the various attributes are constructed:

- AS path (list of numbers)
  If the AS path length distribution is present on the system a path length according to that distribution is chosen. If the user throttles the memory consumption a path length multiplier is applied. The AS numbers are chosen at random including 1 and 64511 except for the ASN of the DUT as well as any ASN used by the test equipment.
- Community (list of two octet)
  If the use of communities is activated for that neighbor, a number of communities will be selected according to a distribution. The community octet itself is generated completely random.
- Origin (IGP, EGP, incomplete)
  Generated based on a distribution.
- Atomic Aggregator (NAG/AG)
  If specified it is generated based on a distribution.
- Aggregator
  This attribute does not influence the best path selection process, but needs to be stored in memory and propagated to neighbors. The IP is chosen randomly if aggregation was selected for the atomic aggregator.
- Multi-Exit-Discriminator MED (numeric value)
  MED will not be used and therefore set to 0. A user that likes to use MED can specify an optional distribution. In this case a MED value according to that distribution is chosen. The usage of MED distributions can be turned on or off on a per neighbor basis.
- Local Preference (numeric value)
  Same as MED.
- Next Hop (IP)
  It is set to the IP of the router tester. In the case of iBGP this is not correct, but IP packets must be forwarded to a valid destination. This may change in the future, if there is an integration with synthetic IGP traffic generation.

7.3.3 Equipment test phase

During the test phase we first the updates are created. The update source can be a recorded trace replayed from disk, or updates generated by our script, or both – a recorded trace "enhanced" with some synthetic updates (e.g., to achieve a certain number of FIB changes, to add additional background BGP load, etc.). For this we allow several sources that may contribute to the final update stream. The next section discusses how this works in detail (Section 7.3.3.1).

Those sources get "mixed" together in the *mixer* (Section 7.3.3.2 on page 115). The mixer can interlace multiple update files to one output stream, or it can "speed-up" a recorded trace. Furthermore, it prepares the update stream for the configured session (e.g., next-hop addresses and session specific parameters are being rewritten). Additionally the mixer collects statistics on the outgoing updates (e.g., counts the number of FIB changes that are likely to occur on the DUT). This information is provided as "feedback" to the synthetic BGP traffic generation.

Finally, the BGP updates are sent out, targeted at the device under test. This can be done via different output streams designed for different test equipment including Agilent Router Tester or a PC running Linux. Section 7.3.3.3 explains the details.

7.3.3.1 Details of update stream construction

This section discusses how to construct the synthetic update streams. We start with instability events and reconstruct update bursts. Finally we make sure to create the requested number of FIB changes.

Creation of instability events

The synthetic BGP update stream is the result of the updates of a generated set of BGP instabilities. An instability cluster consists of multiple prefixes that are supposed to "see updates". Those updates start on various BGP sessions within a relatively short time period. See Section 4.1 on page 49 for more details. Clusters of prefixes are identified during the RIB construction (or during the RIB table read). Then the update stream will be constructed on a minute by minute basis for the configured duration of the test. In each cycle we pick the number of instabilities events that are supposed to start within that minute. This value can be configured by the test engineer or is picked from a distribution. By choosing one of those cluster we have selected the maximum number of BGP peering sessions that participate in this instability event. This is important to later determine the number of possible FIB changes.

Once an instability cluster is picked, it has to be "locked". This is to assure that prefixes are only effected by one instability at a time, because if two instability clusters for the same prefix for the same time are created, the resulting instabilities would not be realistic anymore because the statistical distribution of the burst (burst duration, number of updates within a burst, inter-arrival-time distribution of updates) already capture the observed effects and dynamics.

Next the *"sphere"* of the instability is selected. This is a very important parameter, because it determines on how many peering session (and also on how many physical interfaces of

the DUT) the instability is observable. Once we have selected the number of BGP peering sessions that will participate in this instability event. We pick those sessions that will actually see the instability. Here we prefer sessions that are located on different links, so that it is possible to create the necessary FIB update rate – as requested by the user. We start with the first session[15] on each link, if all links are covered we continue with the second session on each link. Then we have to sub-select the prefixes that are part of this instability cluster. At the moment we apply the instability to all prefixes that are associated with this instability cluster.

Update burst generation

Now that we have determined which prefixes are affected by an instability cluster and on which sessions this instability is visible, we start generating the BGP updates, or in our terminology, create the *update bursts*. We consider three metrics to be relevant for an update bursts:

- number of updates in burst.
- duration of update burst.
- inter-arrival-time of updates within update burst.

From an algorithmic perspective any two of those variables may be used to derive the third metric. Nevertheless, to allow for flexibility we enable the user to provide any combination of two metrics and compute the third.

By default the number of updates in burst cluster and the duration of the cluster is used. The updates are spaced randomly within the cluster which results in an exponentially distribution of the inter-arrival times. Such a distribution of inter-arrival-times can be observed in the Internet.

Another way of constructing a update burst cluster is by using the distributions of inter-update-times and number of updates in a burst. The algorithm is: as long as the number of updates generated so far has not reached the number of updates supposed to be generated, we pick an inter-update-time from the distribution and put a new update in the stream. Note, that the update duration is the sum of the various inter-update-times.

And finally the third method is to specify the inter-update-times and the cluster duration. In this case we construct updates as long, as the duration is not reached. This approach has two problems: First, the duration distribution is only valid for bursts that have a duration – bursts with only one update are not captured by this distribution. Second, the last update may not "hit" the duration. That means that the actual duration is smaller or equal to the duration that is supposed to be generated.

FIB change generation

To be able to construct later FIB changes and determine switch-over-events, we have to keep track of which updates actually result in a modification of the DUTs best path selection process and which only change the RIB entry inside the router (this can be further sub-divided

[15] Note that the order of sessions is randomly selected to assure that each session gets the necessary amount of BGP updates.

into updates that needs to be passed on to neighbors and updates that will not be propagated). We compute the best path that is (most likely) to be picked by the DUT. Note that it is not trivial for our algorithm to detect which updates will actually result in a FIB change or propagation event on the DUT. It may seems feasible at first glance, because our algorithm has all updates targeted towards the DUT as well as the timestamps and thus can "emulate" the best path selection process. Yet updates may be delayed due to TCP problems, or updates on different interfaces may be processed in a slightly different order then that specified by the timestamp. While the result is stable in the long run, which means the router pick the "best path" that is the best path according to the configuration – this does not have to be true for flapping prefixes. If our algorithm now creates flaps to impose a required FIB change rate this could result is some "losses" of FIB changes by a clever implementation of the router software. As we do not know how the DUT is processing the intended FIB changes, we can only try to space FIB changes for one prefix "as far as possible"[16].

7.3.3.2 Stream Mixer

The *mixer* is a central part of the script. Its goal is to ease the process of adding further update sources. For example beside our generated workload, a user might add some specific probe updates to the trace. Such probe updates can be integrated in a larger setup, with IP traffic streams and/or OSPF/ISIS, to measure control plane convergence times and switch-over-times. To implement this, there may be certain conditions that must meet in order to work properly (e.g., synthetically generated prefixes must not interfere with the probe updates). To be able to assure that only minimal parts of the algorithm have to be changed to allow user-supplied update sources we ensure that every update is processed in one central place. With this it is possible for the user to check for user-specified conditions.

Beside a user might want to modify certain characteristics of an input stream. For example a recorded trace should be played back at twice the speed. This is also the appropriate place to adapt all update sources to the outgoing BGP session in the test lab. Once the outgoing update stream is in the correct order and everything is prepared so that the update trace can be send to the DUT, it is possible to estimate the number of FIB changes that will occur on the DUT but this information is important for the update stream generation itself. Global variables will assure proper communication. Furthermore, if the requirements are not meet, then additional updates are created and intermixed into the already produced update stream. This means it has to be possible to "go back in time" and reconstruct a new update mix based on a modified stream.

In summary the mixer interlaces multiple update sources, checks for certain conditions, modifies various parameters of the trace (including playback speed), prepares the outgoing update stream and collects statistics for the synthetic update generation.

7.3.3.3 Output devices

In this section we discuss those devices that are actually performing the tests, the test-equipment itself. Note that the test-equipment is typically a different set of devices than

[16] Note that this spacing has to be determined during a calibration phase by verifying that the number of actual and the number of intended FIB changes match.

the workload generating PC. This is mainly for scalability as the workload generation is very CPU and memory intensive. In a future release of our tool we plan to be able to generate the updates "on the fly", while the test-equipment performs the test. Therefore outgoing test devices can be:

- Router Tester.
- load generating PC (e.g., running BGP_Replay).
- a file stored on disk.

Choosing the option "file stored on disk" means just writing the output stream to disk for later playback via the Router Tester and/or BGP_Replay. For the time being, we have only implemented this option, to pre-generate all tables and updates and store them on disk. This means that during the calibration phase, we will generate a table, store it on disk, transfer it to the test equipment (e.g., automatically via ssh to a load generating PC or to the Router Tester controller, which in itself communicates with the modules), then the script starts the download process of the RIB. Our tool logs into the router and measures the memory consumption and decides what is next to do (e.g., repeat with differently generated RIB). Similar process with the actual test: create tracefile (for the whole test), transfer it to test equipment, start the test.

This part of the code will have to evolve in future work. The main problem we are facing here is time-synchronizing all participating test devices. Furthermore, the updates themselves have to be transfered from the workload generator to the test equipments without overwhelming them – the device that is being tested should still be the router in the middle of the test setup and not the test equipment. Future work has to investigate performance issues of the update generation, the appropriate communication between generating script and the test equipment as well as the necessary synchronization of test equipment of different types (e.g., Router Testers and PCs).

We face a further, unresolved, problem with the TCP window size handling between a PC and router. The stress on the router[17] is created when the TCP window is closed on the outbound BGP sessions of the router to the test equipment. In an repeated, prioritized cycle the router tries to send the queued updates to the test equipment, which is not possible (because of the closed TCP window). This causes the high CPU load on the router. Note that this, if controllable by the test-equipment, can also be used as DUT stressor.

7.3.4 Summary

The corner pillars of our algorithmic design are the flexible and extensible configuration language, the calibration phase, as well as the actual test phase.

In the XML specification the test engineer is able to set all parameters for the test, this includes the specification of which distributions are supposed to be used to generate the workload.

Next the parameters need to be calibrated. At the moment, this is the RIB memory consumption, and a check if the number of FIB changes can be actually achieved on the DUT. To do this we generate multiple RIBs, download them to the routers and measure the memory consumption. We repeat this until we have approximated the requested memory consumption sufficiently well. Then we send a short update stream to the DUT that induces the maximum

[17] This problem was observed at a Cisco GSR 12008 router.

number of FIB changes per time unit that is requested (and can be generated with the available prefixes). The CLI of the DUT reveals, if the intended number of FIB changes match the actual FIB recomputes on the DUT.

During the test phase the algorithm generate updates based on the identified metrics. Those synthetically generated streams can be mixed with other update sources. We compute statistics about the "impact" of an update and this influences the update generation process again. While the current implementation pre-generates the trace files for the whole specified test duration, it should be possible to generate the updates on the fly and run the update generation in an endless loop.

7.4 Summary

In this section we describe a tool that generates synthetical BGP update traces for multiple peering sessions that can be targeted to devices under test (DUTs). The tool is user-friendly, highly flexible and supports well-specified test conditions. The goal is to ease the engineer's task to instantiate a complex BGP test. Reasonable defaults are provided, so that a test engineer is not lost in the intricacies of the BGP test setup. Therefore our tool creates an initial-test setup that respects the wishes of the test engineer (e.g., number and type of peers) and at the same time is able to reflect some of the variability of the Internet (e.g., number of prefixes, AS path length, usage of communities). This means deriving via data characterization a *"normal setting"* for the BGP workload generator. Using this approach the test-setup gets easier and at the same time we bring dynamics and variability of the real Internet to the test-lab; more and more complex test setups are possible. With this our tool helps in a wide variety of equipment testing: testing software implementations, performance, scalability and data-plane convergence.

8 Conclusion

At a first glance BGP is a rather simple protocol. It forwards only reachability information from one router to the other and each router is allowed to pick its "best path" from a set of available alternatives. Yet, this simple protocol is of great interest to the community as the the experiment "Internet" has become a critical part of our communication infrastructure. Not only technical people try to optimize the goal of global reachability, but also commercial, political, social interests are biasing its de-facto standard routing protocol, BGP.

In this thesis we look at some of the complexities of BGP. We start with configuration management. Despite all of our knowledge in software, operating and distributed systems, the configuration of todays router sometimes appear to be from the stone ages of software design. In Chapter 3 we present a system that helps raise the level of abstraction from a router configuration to an AS-wide configuration. We first identify the concepts underlying a routing policy and based on this understanding we propose a system for managing AS-wide routing policies as first class entities. This eliminates the need to configure the IP routers manually to change the embedded policy. Furthermore, it provides a flexible, extensible, and scalable framework for formulating the routing policy, which respects the divisions of responsibility within an ISP. In addition, the automatic generation of the *configlets* by the *configurator* reduces the probability of errors. And by the virtue of the flexibility of the system the likelihood decreases that the system is bypassed to fix operational problems, thus clear and documented migrations from one state of the network to the next are possible and help with troubleshooting. A prototype is in production use at Deutsche Telekom. The ability to express an AS-wide routing policy as a flexible collection of enforced policies and available services independent of the current state of the network as well as the ability to easily and quickly introduce new services or adjust policies has proven to be valuable.

Beside the fact that a routing policy can now be expressed very easily on an AS-wide level, this does not solve the problem that routing policies in itself can bring the Internet to diverging states, e.g., [7]. This can happen with our system or without, actually this could happen every day and with fully legitimate policies. As a first step to be able to cope with the dynamics that arise from such a distributed system, we need a detailed understanding about the signaling properties of BGP. In Chapter 4 we review BGP dynamics and in Chapter 4 BGP beacons as well as the general behavior of BGP updates. In this context we used simple clustering technique to observe path exploration, identify stable states, and analyze diverging prefixes. While we find, that most instabilities in the Internet converge within two minutes (in the case of a withdraw within four to six minutes), there are some events that can last for several hours, days, or even weeks.

Such a knowledge can be beneficial to operators, as it may help them to detect unwanted routing conditions and start debugging problems before customers complain. Yet, troubleshooting is still difficult because of the lack of information who to call and to fix a problem. For this reason we developed a methodology to narrow down the set of ASes that potentially caused

the routing instability, solely based on the observable routing changes collected by passive monitoring the routing system. Chapter 5 presents our approach together with some initial results. Indeed, we are able to narrow down the candidate sets of possible ASes to a single AS and explain with this 93% of all prefixes that were observing updates.

While this brings us some insights on the observable protocol properties, it still does not explain why we see such behavior. Clearly, a lot of todays convergence delays are due to the MRAI timer and route flap damping, but what about overloaded routers along the path? In Chapter 6 we discuss a methodology that evaluates such questions in a test-lab by measuring BGP pass-through times. We investigate the impact of factors, such as CPU load, number of BGP peers, etc., on the propagation delays of BGP updates. We find that that BGP pass-through times are rather small with average delays less than $200ms$. Yet, if routers are operated outside their specification limits, then BGP processing delays can become quite high.

Such questions can be viewed from a more general perspective: How can we test network equipment in a test-lab so that the results are comparable to the placing this device in the operational network? How can we recreate the dynamics and the variability observable in the Internet in a small-scale test-lab. Therefore we investigate in Chapter 7 how we can reproduce a realistic BGP workload for devices under test in a small-scale test-lab. We use our knowledge from the previous chapters about the characteristics of BGP signaling properties to propose a tool that generates synthetical BGP update traces. The goal is to ease the task of the test engineer and help instantiate complex BGP tests. This is realized by assuming reasonable defaults whenever the user is "a bit vague" in the test specification. With this test-setups become easier and at the same time we bring dynamics and variability of the real Internet into the test-lab. This in turn helps in a wide variety of equipment testing: testing software implementations, performance, scalability and data-plane convergence.

The main contribution of this thesis is to study a framework of how to handle the complexity of BGP today. We investigated this

- via characterization of the dynamic behavior of BGP by observing the properties of the Internet control plane,
- via router testing by designing a tool that brings BGP dynamics into the test-lab, and
- via configuration management that defines an AS-wide routing policy and instantiate this policy in the operation network by automated router configuration.

While it is not always easy working with BGP and its limitations, we showed areas and ways to help operators, vendors to do their daily job and researchers to improve the Internet. One day there maybe replacements for BGP (e.g., [147]) – yet one should better have a very good solutions to replace BGP, the inter-domain routing protocol.

9 Future Work

In this chapter we discuss some open questions regarding how this work may evolve in the future. We start with a project, in Section 9.1, which is targeted towards a more realistic Internet-like simulation framework for more accurate traffic engineering based on insights gained in Chapters 4 to 5. Next we discuss a project, in Section 9.2, that extends our configuration management system, proposed in Chapter 3, and combines this with our router testing methodology developed in Chapter 7.

9.1 Towards more realistic Internet-like simulations

To better understand the impact of routing policies on inter-domain traffic flows, we propose to develop a methodology that allows simulating complex inter-domain topologies with conditions similar to those that are observed in reality (i.e., with more realistic filter policies). Especially traffic shifts inside ASes and their impact on other domains can only be understood by large Internet-size simulations. In addition, such questions are closely related to the limits imposed by policies in respect to how far routing updates will propagate.

Thus, to understand inter-domain traffic flows our simulation framework deals with the topology as well as with policy decision of ISPs. Sadly enough, ISPs usually do neither disclose topology nor policy information, so that path inference becomes necessary.

While the physical topology shows the paths that could be used across the network, it relies on the routing protocols to decide which paths are actually being used. To the best of our knowledge, today, no simulation framework fully captures both aspects: the available links in the topology (i.e., [104, 148]) and the decisions taken by the routing system (i.e., [31, 137]).

Our goal is to develop a methodology where researchers and ISPs can ask *what-if* questions in terms of:

- impact of link failures on routing dynamics, BGP update spread and traffic shifts;
- impact of changed status of a relationship (e.g., canceled peering);
- impact of changes in the connectivity of transit networks;
- impact of internal link changes/failures on the routing inside an AS;
- impact of complex policies between ASes;
- impact on the traffic of a transit AS of changing its Internet connectivity.

We now discuss how to tackle these problems, by looking at how to construct the necessary simulation setup.

9.1.1 Proposed approach

The ingredients to our large scale Internet-like simulations are: a simulator, the inter-domain topology, approximations of intra-domain topologies, and a set of filter policies.

Recent advances on simulators, such as C-BGP [74] developed at UCL in Belgium, make it feasible to work with large models of the Internet that include all ASes. Basic AS-level relationships can be inferred by common algorithms such as [31]. Yet, this is not sufficient, because internal topologies of ISPs span a large geographical region (see Section 2.1.1 on page 13, especially Figure 2.3) and filter policies[1] add an additional complexity to the best path selection process that has to be considered as a relevant factor in the simulation.

Our intent is to start with a standard AS relationship inference with data from AS paths collected by RIPE, Routeviews, and Akamai. In a next step this inference is used as a "seed" to the simulation. We model the internal topology on a PoP level of transit ISP – stub ASes are modeled with one router only. It is important to note that ASes, such as tier-1 and large tier-2, operate their networks on multiple continents. Smaller ASes operate only within a certain region (they can be geographically located by whois-queries and/or other techniques). IGP metrics, unless better known, can be assigned according to a "semi-random" function that approximates geographic distances.

Finally it is important to derive the filtering on a per eBGP session basis. In a first step the filter policies are approximated from the standard routing policies inference. We validate the simulation by comparing them with actual monitored paths. Yet, the outcome is not expected to be very accurate. Therefore a heuristic similar to the one presented in Section 5.2 is used to detect locations (AS-edges within the simulation) that hint towards a different setting of policies or undetected inter-AS links. With that we can improve the accuracy of the simulation in successive steps. The simulation is re-run and revalidated with the new policies inferred by our heuristics. Additional information from various looking glasses may help in improving the accuracy up to the desired level.

9.1.2 Benefits

Such an Internet-wide simulation technique is very valuable for operators to diagnose problems, optimize network performance, identify peering partners, provide better connectivity to the rest of the Internet (e.g., shorter average AS paths due to an improved selection of upstream providers) and see what competitors are doing as well as estimate their impact on the own AS. Researchers can study routing protocol behavior, traffic shifts with load-adaptive routing mechanisms and improve traffic engineering methodologies. Finally the accuracy might even be sufficient to detect problematic routing conditions more easily [7, 16, 23].

9.2 Configuration management

Next we turn to configuration management. The system presented in Chapter 3 enables operators to implement their business model in a straightforward, flexible fashion, facilitates automatic network management and allows an easy realization of new policies – there is also a potential risk that it becomes now **too easy** to instantiate wrong policies. There should be an automated configuration management system which first validate that the vendor-specific configuration actually matches the intended routing conditions the administrator contemplates.

[1] Policy settings of ISPs are not limited to the two simple categories "customer-provider" and "peering".

Thus, how can an ISP verify their configurations? How can tools help in checking if the intended policy actually matches the effective policy in a network (e.g., [60])? Is it possible to validate router configurations in a way that they are guaranteed to be globally sane (e.g., [61])?

9.2.1 Proposed approach

Recall from Chapter 3, that we designed a system that is able to generate the appropriate eBGP *configlets* and thus can configure routers automatically. This is done based on a set of input configuration files decompositioned by the organizational boundaries within an ISP (i.e., the policy designer creates the routing policy of the ISP; the network administrator adds customers, routers, etc.; and the person realizing and testing BGP operations is working on the *fragments* of the router code).

What is missing in this design is a validation of the generated *configlets* before they are uploaded to the routers in the operational backbone. We propose to use the above mentioned C-BGP simulator for this task. The simulator is able to read the generated *configlets* directly and thus simulate network with the policies in an AS-wide (or even Internet-wide) context.

Together with an accurate model of the internal topology of an AS, the RIB of all neighboring ASes, and maybe the approximation from Section 9.1 this results in a powerful tool, that can find local unwanted policy settings that were configured by mistake as well as potentially even detect global conflicting policies [7]. With such a methodology it is possible to forsee the routing states created by the configuration [21]. Based on this, certain criteria can be checked: e.g., identify routers that pick paths via the upstream, while a customer link is available (due to wrong communities?); or detect that customer routes disappear on peering links, because a peer announced a shorter AS path which might be picked; or the simulation does not “converge” based on conflicting policies; etc..

After the policy is checked via simulation (iBGP, eBGP and VPNs), the tools needs to make sure that the *configlets* are actually supported by the field equipment. With our methodology developed in Chapter 7 basic compliance and regression tests can be performed with a workload that corresponds to the setting where the actual router is located, which adds confidence that the router is actually capable of performing its task appropriately.

An integrated documentation system keeps track about the migration from one network configuration state to the next. Here the changes between the last and the new configuration are highlighted which help operators to retrace changes and ease debugging.

9.2.2 Benefits

The importance of the Internet requires that the protocols are correctly configured. Thus we propose to use our system that enables ISPs to specify a routing policy AS-wide (see Chapter 3) and combine this with a verification mechanism. This uses a simulation of the routing policies of the AS (or even beyond the AS) to check if the intended outcome of a policy actually matches the BGP expressions that are supposed to be uploaded to the routers. Furthermore, basic router tests (see Chapter 7) can verify that the router is capable of performing its task in the given environment with the given configuration. This may avoid some configuration mistakes and thus may contribute to the stability of the Internet.

List of Figures

List of Tables

Bibliography

[1] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)." RFC 4271.

[2] J. Moy, *OSPF: Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.

[3] D. Oran (Editor), "OSI IS-IS Intra-domain Routing Protocol," 1990. RFC 1142, ISO DP 10589.

[4] A. Feldmann, A. C. Gilbert, and W. Willinger, "Data networks as cascades: Investigating the multifractal nature of Internet WAN traffic," in *Proc. ACM SIGCOMM*, 1998.

[5] Akamai Technologies. `http://www.akamai.com`.

[6] L. Qiu, Y. R. Yang, Y. Zhang, and S. Shenker, "On selfish routing in internet-like environments.," in *Proc. ACM SIGCOMM*, pp. 151–162, 2003.

[7] T. G. Griffin and G. Huston, "BGP Wedgies," 2005. RFC 4264.

[8] R. Bush, T. Griffin, Z. M. Mao, E. Purpus, and D. Stutzbach, "Happy Packets - Initial Results," 2004. NANOG 31.

[9] A. Feldmann, N. Kammenhuber, O. Maennel, B. M. Maggs, R. D. Prisco, and R. Sundaram, "A methodology for estimating interdomain web traffic demand.," in *Proc. ACM IMC*, 2004.

[10] T. G. Griffin Interdomain routing links. `http://www.cl.cam.ac.uk/users/tgg22/interdomain/`.

[11] T. G. Griffin and B. J. Premore, "An Experimental Analysis of BGP Convergence Time," in *Proc. ICNP*, 2001.

[12] T. G. Griffin, "What is the Sound of One Route Flapping?," 2002. IPAM.

[13] R. Teixeira, A. Shaikh, T. G. Griffin, and J. Rexford, "Dynamics of Hot-Potato Routing in IP Networks," in *Proc. ACM SIGMETRICS*, 2004.

[14] R. Teixeira, A. Shaikh, T. G. Griffin, and G. M. Voelker, "Network sensitivity to hot-potato disruptions," in *Proc. ACM SIGCOMM*, 2004.

[15] R. Teixeira, N. G. Duffield, J. Rexford, and M. Roughan, "Traffic matrix reloaded: Impact of routing changes," in *Proc. Passive and Active Measurement Workshop (PAM)*, 2005.

[16] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "Policy Disputes in Path Vector Protocols," in *Proc. ICNP*, 1999.

[17] N. Feamster, H. Balakrishnan, and R. Johari, "Stable Policy Routing with Provider Independence," in *Proc. ACM SIGCOMM*, 2005.

[18] D. Caldwell, A. Gilbert, J. Gottlieb, A. Greenberg, G. Hjalmtysson, and J. Rexford, "The cutting EDGE of IP router configuration," in *ACM SIGCOMM HotNets Workshop*, 2003.

[19] D. Wetherall, R. Mahajan, and T. Anderson, "Understanding BGP misconfigurations," in *Proc. ACM SIGCOMM*, 2002.

[20] S. Uhlig and O. Bonaventure and B. Quoitin, "Interdomain Traffic Engineering with minimal BGP configurations," in *Proc. of the 18th International Teletraffic Congress, Berlin*, 2003.

[21] B. Quoitin and S. Uhlig, "Modeling the routing of an Autonomous System with C-BGP," 2005. (under submission).

[22] "IETF Benchmarking Methodology Working Group (bmwg)." `http://www.ietf.org/html.charters/bmwg-charter.html`.

[23] T. G. Griffin and G. Wilfong, "A Safe Path Vector Protocol," in *Proc. IEEE INFOCOM*, 2000.

[24] J. Postel, "Internet protocol," 1981. RFC 791.

[25] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Supernetting: an Address Assignment and Aggregation Strategy," 1992. RFC 1338.

[26] V. Fuller, T. Li, J. Yu, and K. Varadhan, "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy," 1993. RFC 1519.

[27] "IANA - Internet Assigned Number Authority." `http://www.iana.org/ipaddress/ip-addresses.htm`.

[28] K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, and J. Postel, "Internet Registry IP Allocation Guidelines," 1996. RFC 2050.

[29] G. Huston, "Where's the Money?," 2005. The ISP Column. `http://www.potaroo.net/ispcol/2005-01/interconn.html`.

[30] R. Govindan and A. Reddy, "An analysis of Internet inter-domain topology and route stability," in *Proc. IEEE INFOCOM*, 1997.

[31] L. Gao, "On Inferring Autonomous System Relationships in the Internet," in *Proc. IEEE Global Internet*, 2000.

[32] J. W. Stewart, *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley, 1999.

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

[34] T. Bates, R. Chandra, and E. Chen, "BGP Route Reflection - An Alternative to Full Mesh IBGP," 2000. RFC 2796.

[35] R. Teixeira and J. Rexford, "A measurement framework for pin-pointing routing changes," in *Proc. ACM SIGCOMM Network Troubleshooting Workshop*, 2004.

[36] R. Chandra, P. Traina, and T. Li, "BGP Communities Attribute," 1996. RFC 1997.

[37] L. Subramanian, V. N. Padmanabhan, and R. H. Katz, "Geographic properties of Internet routing," in *Proc. Usenix*, 2002.

[38] Cisco Systems, "How the bgp deterministic-med Command Differs from the bgp always-compare-med Command," 2005. `http://www.cisco.com/warp/public/459/bgp-med.pdf`.

[39] Cisco Systems, "BGP Cost Community," 2005. `http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/120limit/120s/120s24/s_bgpcc.pdf`.

[40] "BGP Multipath Load Sharing for Both eBGP and iBGP in an MPLS-VPN." `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t4/fteibmpl.pdf`.

[41] T. G. Griffin and G. Wilfong, "On the Correctness of IBGP Configuration," in *Proc. ACM SIGCOMM*, 2002.

[42] S. Floyd and V. Jacobson, "The synchronization of periodic routing messages," *IEEE/ACM Transactions on Networking*, 1994.

[43] C. Villamiyar, R. Chandra, and R. Govindan, "BGP route flap damping," 1998. RFC 2439.

[44] C. Panigl, J. Schmitz, P. Smith, and C. Vistoli, "RIPE Routing-WG Recommendation for Coordinated Route-flap Damping Parameters ," 2001. `http://www.ripe.net/ripe/docs/ripe-229.html`.

[45] Z. M. Mao, R. Govindan, G. Varghese, and R. Katz, "Route flap damping exacerbates Internet routing convergence," in *Proc. ACM SIGCOMM*, 2002.

[46] R. Bush, T. Griffin, and Z. M. Mao, "Route flap damping: harmful?." RIPE 43, September 2002. `http://www.ripe.net/ripe/meetings/archive/ripe-43/presentations/ripe43-routing-flap.pdf`.

[47] Juniper Networks Inc., "Out-delay." `https://www.juniper.net/techpubs/software/junos/junos57/swconfig57-routing/html/bgp-summary32.html`.

[48] RIPE's Routing Information Service. `http://www.ripe.net/ris/`.

[49] University of Oregon RouteViews project. `http://www.routeviews.org/`.

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

[51] Cisco, *Cisco IOS Configuration Fundamentals*. Cisco Press, New Riders Publishing, 1998. Documentation from the Cisco IOS reference Library.

[52] R. Enns, "NETCONF Configuration Protocol," 2004. Internet Draft (draft-ietf-netconf-prot-04.txt).

[53] "Introduction to Configuring JUNOS Internet Software." `http://www.juniper.net/training/elearning/junos_cli/`.

[54] Cisco 12000 Series Routers. `http://www.cisco.com/en/US/products/hw/routers/ps167/`.

[55] IRRToolSet. `http://www.isc.org/sw/IRRToolSet/`.

[56] C. Alaettinoglu, "Rtconfig: Router configuration generator," 1996. NANOG 6.

[57] Internet Routing Registry. `http://www.irr.net/`.

[58] RADB - Routing Assets Database. `http://www.radb.net/`.

[59] G. Siganos and M. Faloutsos, "Analyzing BGP Policies: Methodology and Tool," in *Proc. IEEE INFOCOM*, 2004.

[60] N. Feamster, "Practical verification techniques for wide-area routing," in *Proc. ACM Workshop on Hot Topics in Networks*, 2003.

[61] N. Feamster and H. Balakrishnan, "Verifying the correctness of wide-area Internet routing," Tech. Rep. MIT-LCS-TR-948, MIT, 2004.

[62] D. Meyer, J. Schmitz, C. Orange, M. Prior, and C. Alaettinoglu, "Using RPSL in practice." RFC 2650.

[63] C. Villamizar, C. Alaettinoglu, D. Meyer, and S. Murphy, "Routing Policy System Security," 1999. RFC 2725.

[64] RANCID - Really Awesome New Cisco confIg Differ. `http://www.shrubbery.net/rancid/`.

[65] Cisco IP Solution Center. `http://www.cisco.com/en/US/products/sw/netmgtsw/ps4748/index.html`.

[66] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang, "Automated provisioning of BGP customers," *IEEE Network Magazine*, 2003.

[67] A. Feldmann and J. Rexford, "IP network configuration for interdomain traffic engineering," *IEEE Network Magazine*, 2001.

[68] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," in *IEEE Symposium on Security and Privacy*, pp. 17–31, 1999.

[69] S. Agarwal, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "The Impact of BGP Dynamics on Intra-Domain Traffic," in *Proc. ACM SIGMETRICS*, 2004.

[70] S. Uhlig and O. Bonaventure, "Implications of interdomain traffic characteristics on traffic engineering," *European Transactions on Telecommunications*, 2002.

[71] N. Feamster and H. Balakrishnan, "Towards a Logic for Wide-Area Internet Routing," in *Proc. ACM SIGCOMM FDNA Workshop*, 2003.

[72] N. Feamster and H. Balakrishnan, "Detecting BGP Configuration Faults with Static Analysis," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[73] T. G. Griffin, A. D. Jaggard, and V. Ramachandran, "Design Principles of Policy Languages for Path Vector Protocols," in *Proc. ACM SIGCOMM*, 2003.

[74] S. Uhlig and B. Quoitin, "Tweak-it: BGP-based Interdomain Traffic Engineering for Transit ASes," in *Proc. of the first EuroNGI Conference on Next Generation Internet Networks*, 2005.

[75] T. G. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *Proc. ACM SIGCOMM*, 1999.

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

[77] B. Chinoy, "Dynamics of Internet routing information," in *Proc. ACM SIGCOMM*, 1993.

[78] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek, "Measuring the effects of Internet path faults on reactive routing," in *Proc. ACM SIGMETRICS*, 2003.

[79] D. Obradovic, "Model and convergence time of BGP," in *Proc. IEEE INFOCOM*, 2002.

[80] X. Zhao, M. Lad, D. Pei, L. Wang, D. Massey, S. Wu, and L. Zhang, "Understanding BGP Behavior Through A Study of DoD Prefixes." In Proc. of DISCEX III, April 2003.

[81] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "An analysis of BGP multiple origin AS (MOAS) conflicts," in *Proc. ACM IMW*, 2001.

[82] O. Maennel and A. Feldmann, "Realistic BGP traffic for test labs," in *Proc. ACM SIGCOMM*, 2002.

[83] O. Maennel and A. Feldmann, "Identifying Problematic Inter-domain Routing Issues," 2002. NANOG 24.

[84] C. Labovitz, A. Ahuja, A. Abose, and F. Jahanian, "An Experimental Study of Delayed Internet Routing Convergence," in *Proc. ACM SIGCOMM*, 2000.

[85] C. Labovitz, "Scalability of the Internet backbone routing infrastructure," in *PhD Thesis, University of Michigan*, 1999.

[86] G. Huston, "IPv4 Address Space Report," 2005. `http://bgp.potaroo.net/ipv4/`.

[87] S. Donelan, "What Worked and What Didn't: 9/11," 2001. NANOG 23.

[88] C. Labovitz and A. Ahuja, "Shining Light on Dark Internet Address Space," 2001. NANOG 23.

[89] G. Huston, "Allocations vs announcements," 2004. Internet Society Publications: ISP Column. `http://www.potaroo.net/papers/isoc/2004-05-02/alloc.html`.

[90] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "BGP routing stability of popular destinations," in *Proc. ACM IMW*, 2002.

[91] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang, "Observation and analysis of BGP behavior under stress," in *Proc. ACM IMW*, 2002.

[92] M. Caesar, L. Subramanian, and R. H. Katz, "Route cause analysis of Internet routing dynamics," tech. rep., UCB/CSD-04-1302, 2003.

[93] D.-F. Chang, R. Govindan, and J. Heidemann, "The Temporal and Toplogical Characteristics of BGP Path Changes," in *Proc. ICNP*, 2003.

[94] M. Lad, A. Nanavati, D. Massey, and L. Zhang, "An algorithmic approach to identifying link failures," in *10th Pacific Rim Dependable Computing Symposium*, 2004.

[95] J. Wu, Z. M. Mao, J. Rexford, and J. Wang, "Finding a needle in a haystack: Pinpointing significant BGP routing changes in an IP network," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[96] RIS Routing Beacons. `http://www.ripe.net/projects/ris/docs/beacon.html`.

[97] C. Labovitz, R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Trans. Networking*, 1998.

[98] C. Labovitz, R. Malan, and F. Jahanian, "Origins of Internet routing instability," in *Proc. IEEE INFOCOM*, 1999.

[99] C. Labovitz, A. Ahuja, and F. Jahanian, "Experimental study of Internet stability and wide-area network failures," in *Proc. International Symposium on Fault-Tolerant Computing*, 1999.

[100] C. Labovitz, R. Wattenhofer, S. Venkatachary, and A. Ahuja, "The impact of Internet policy and topology on delayed routing convergence," in *Proc. IEEE INFOCOM*, 2001.

[101] Z. M. Mao, R. Bush, T. G. Griffin, and M. Roughan, "BGP Beacons," in *Proc. ACM IMC*, 2003.

[102] S. Bürkle, "BGP convergence analysis," Diplomarbeit, Saarland University, 2003.

[103] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proc. IEEE INFOCOM*, Computer Science Department, University of Southern California, 2000.

[104] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, 2002.

[105] D. G. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan, "Topology Inference from BGP Routing Dynamics," in *Proc. ACM IMW*, 2002.

[106] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *Proc. ACM IMW*, 2002.

[107] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz, "Characterizing the Internet hierarchy from multiple vantage points," in *Proc. IEEE INFOCOM*, 2002.

[108] G. D. Battista, M. Patrignani, and M. Pizzonia, "Computing the Types of the Relationships between Autonomous Systems," in *Proc. IEEE INFOCOM*, 2003.

[109] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. E. Anderson, "The End-to-End Effects of Internet Path Selection," in *Proc. ACM SIGCOMM*, 1999.

[110] H. Tangmunarunkit, R. Govindan, S. Shenker, and D. Estrin, "The impact of Internet policy on Internet paths," in *Proc. IEEE INFOCOM*, 2001.

[111] Agilent N2X Multi-Services test solution. `http://advanced.comms.agilent.com/n2x/`.

[112] IXIA BGP Routing Protocol Emulation Software, 2002. `http://www.ixiacom.com/`.

[113] AX/4000 Spirent Communications, 2005. `http://www.spirentcom.com`.

[114] Arsin Corporation, "RIG, A BGP Routing Instability Generator," 2001. `http://www.arsin.com/archive/archive26.htm`.

[115] H. Berkowitz, A. Retana, S. Hares, and P. Krishnaswamy, "Benchmarking methodology for basic BGP convergence," 2002. Internet Draft (draft-ietf-bmwg-bgpbas-01.txt).

[116] H. Berkowitz, A. Retana, S. Hares, P. Krishnaswamy, and M. Lepp, "Terminology for Benchmarking BGP Device Convergence in the Plane," 2003. Internet Draft (draft-ietf-bmwg-conterm-05.txt).

[117] A. Shaikh, L. Kalampoukas, R. Dube, and A. Varma, "Routing stability in congested networks: Experimentation and analysis," in *Proc. ACM SIGCOMM*, 2000.

[118] A. Shaikh, R. Dube, and A. Varma, "Avoiding instability during graceful shutdown of OSPF," in *Proc. IEEE INFOCOM*, 2002.

[119] D.-F. Chang, R. Govindan, and J. Heidemann, "An Empirical Study of Router Response to Large BGP Routing Table Load," tech. rep., USC/ISI, 2001.

[120] D. A. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, and A. Greenberg, "Routing design in operational networks: A look from the inside," in *Proc. ACM SIGCOMM*, 2004.

[121] B. Raghavan and A. C. Snoeren, "A system for authenticated policy-compliant routing," in *Proc. ACM SIGCOMM*, 2004.

[122] G. Huston, "NOPEER Community for Border Gateway Protocol (BGP) Route Scope Control," 2004. RFC 3765.

[123] O. Bonaventure and B. Quoitin, "Common utilizations of the BGP community attribute," 2003. Internet Draft (draft-bq-bgp-communities-00.txt).

[124] D. McPerson, V. Gill, D. Walton, and A. Retana, "Border Gateway Protocol (BGP) Persistent Route Oscillation Condition," 2002. RFC 3345.

[125] North American Network Operators Group. `http://www.nanog.org/`.

[126] RIPE Meetings. `http://www.ripe.net/meetings/`.

[127] Asia Pacific Regional Internet Conference on Operational Technologies. `http://www.apricot.net/`.

[128] R. Govindan, C. Alaettinoglu, G. Eddy, D. Kessens, S. Kumar, and W.-S. Lee, "An architecture for stable, analyzable Internet routing," *IEEE Network Magazine*, 1999.

[129] G. Huston, "Interconnection, Peering, and Settlements," in *Internet Protocol Journal*, 1999.

[130] D. Turk, "Configuring BGP to Block Denial-of-Service Attacks," 2004. RFC 3882.

[131] NetML (an XML specification language to configure and express network devices). `http://giga.dia.uniroma3.it/~ivan/NetML/`.

[132] T. G. Griffin, "Routing policy languages must be designed and standardized." WIRED – Workshop on Internet Routing Evolution and Design, 2003.

[133] K. Claffy, H.-W. Braun, and G. Polyzos, "A Parametrizable methodology for Internet traffic flow profiling," in *IEEE Journal on Selected Areas in Communications*, 1995.

[134] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," tech. rep., USC/ISI-96-631, 1996.

[135] A. Basu, C.-H. L. Ong, A. Rasala, F. B. Shepherd, and G. Wilfong, "Route Oscillations in I-BGP with Route Reflection," in *Proc. ACM SIGCOMM*, 2002.

[136] H. Uijterwaal, "Routing Beacons," 2002. IEPG meeting `http://www.ripe.net/ris/Talks/0211_IEPG/sld009.html`.

[137] Z. M. Mao, L. Qiu, J. Wang, and Y. Zhang, "On AS Level Path Inference," in *Proc. ACM SIGMETRICS*, 2005.

[138] G. Battista, M. Patrignani, and M. Pizzonia, "Computing the Types of the Relationships Between Autonomous Systems," in *Proc. IEEE INFOCOM*, 2003.

[139] S. Agarwal, C.-N. Chuah, S. Bhattacharyya, and C. Diot, "Impact of BGP dynamics on router CPU utilization," in *Proc. Passive and Active Measurement Workshop (PAM)*, 2004.

[140] "ENDACE measurement systems." `http://www.endace.com/`.

[141] S. Bradner and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices," 1999. RFC 2544.

[142] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *Proc. ACM SIGCOMM*, 1999.

[143] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet topology," in *IEEE Communication Magazine*, 1997.

[144] L. Gao and J. Rexford, "Stable Internet routing without global coordination," in *Proc. ACM SIGMETRICS*, 2001.

[145] G. Huston, "IPv4 ALL Allocation Report," 2004. `http://bgp.potaroo.net/ipv4/stats/allocated-all.html`.

[146] M. Kühne, P. Rendek, S. Wilmot, and L. Vegoda, "IPv4 Address Allocation and Assignment Policies for the RIPE NCC Service Region," 2003. RIPE-288.

[147] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica, "HLP: A Next-generation Interdomain Routing Protocol," in *Proc. ACM SIGCOMM*, 2005.

[148] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," in *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems – MASCOTS*, 2001.