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

Interdomain Routing Policies in the Internet: Inference and Analysis

Massimo Rimondini

Interdomain Routing Policies in the Internet: Inference and Analysis

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

Roma Tre University
Dept. of Informatics and Automation

March 2007

Committee:
Prof. Giuseppe Di Battista (Università degli Studi Roma Tre)

Reviewers:
Prof. Timothy G. Griffin (University of Cambridge)
Prof. Fulvio Risso (Politecnico di Torino)

To my parents, who made it possible
To my advisor, who made me capable of doing it
To my friends, who made it pleasant

Contents

Introduction **3**

I Background and Objectives **5**

1 Interdomain Routing with BGP **7**
1.1 BGP and Internet Routing 7
1.2 Policy Routing with BGP 15
1.3 Collecting BGP Routing Information 16

2 Motivations and Objectives **17**
2.1 The Need for Topological Information 17
2.2 Analysis of Routing Policies 18
2.3 How to Experiment Routing by Emulation 20

II Interdomain Topology Discovery Methods **21**

Introduction **23**

3 Topology Discovery by Active Probing **27**
3.1 Background 28
3.2 Related Work 29
3.3 Probing Primitives 31
3.3.1 Withdrawal Observation 33
3.3.2 AS-set Stuffing 33
3.3.3 Effectiveness and Limitations 33
3.4 Discovery Techniques 35
3.4.1 Obtaining a Topology 35
3.4.2 Operational and Ethical Impact 38
3.5 Experimental Results 40
3.5.1 Experimental Setup 40
3.5.2 Topology Discovery 41

3.5.3 Impact of Route-flap Dampening . . . 43
3.5.4 Comparison with the Full AS Graph . . . 45
3.6 Conclusions . . . 46

4 Topology Discovery based on Registry Information 47
4.1 Introduction and Related Work . . . 48
4.2 Background . . . 50
4.3 A Methodology and a Service to Extract Peerings from the IRR 51
4.3.1 Reference Data Set . . . 53
4.3.2 Integrating Information from Different Registries . . . 55
4.3.3 Extracting Peering Information from the IRR . . . 56
4.3.4 Classifying the Peerings . . . 60
4.4 Comparison with the State of the Art . . . 60
4.5 Conclusions . . . 62

III Inference and Analysis of Routing Policies 65

Introduction 67

5 Inference of Commercial Relationships between Autonomous Systems 69
5.1 Background . . . 70
5.2 Problem Statement . . . 71
5.3 Inference Algorithms . . . 73
5.4 A Methodology to Evaluate the Quality of Inference Algorithms 74
5.4.1 Measuring Differences between Inference Results . . . 75
5.4.2 Extensively Evaluating Inference Algorithms . . . 76
5.5 A Software Suite to Evaluate Inference Algorithms . . . 78
5.6 Experimental Results . . . 80
5.6.1 Data Sets . . . 80
5.6.2 Independence of Inference Results from Routing Changes 82
5.6.3 Independence of Inference Results from the Algorithm . 85
5.7 Conclusions . . . 87

6 Policy-Based Interdomain Traffic Engineering 91
6.1 Background . . . 92
6.2 Related Work . . . 93
6.3 Traffic Engineering by Using Prepending . . . 94
6.3.1 Computing Prepending by Integer Linear Programming 97
6.3.2 Computing Prepending by Computational Geometry . . 101
6.4 Remarks about Computational Complexity . . . 104
6.5 Applicability Considerations . . . 106

6.6 Conclusions 107

7 Interplay of Routing Policies at different Autonomous Systems 109
7.1 Checking the Feasibility of AS-paths 110
7.2 Revealing the Preference Associated to AS-paths 112
7.3 Instabilities Caused by Routing Policies 116
7.3.1 The Stable Paths Problem Model 118
7.3.2 An Alternative Model to Investigate BGP Routing Instabilities 123
7.4 Conclusions 131

IV Experimenting Routing by Emulation 133

8 Emulation of Computer Networks with Netkit 135
8.1 An Overview of Emulation Environments 137
8.2 The Architecture of Netkit 143
8.2.1 User-Mode Linux: a Kernel in the Kernel 147
8.2.2 Networking Support in Netkit 153
8.2.3 A Filesystem of Networking Tools 156
8.2.4 User Interface 160
8.3 Setting up a Virtual Lab 166
8.3.1 Defining the Topology 168
8.3.2 Implementing the Topology 169
8.3.3 Setting Network Addresses and Startup Time Services . 171
8.3.4 Configuring Services 172
8.3.5 Tuning Lab Startup 174
8.3.6 Testing the Lab 175
8.4 Managing a Virtual Lab 177
8.5 A Case Study: Multihoming 179
8.6 Conclusions 189

V Conclusions and Bibliography 193

Conclusions 195

Open Problems 199

Bibliography 203

Introduction

COMMUNICATION systems play nowadays a crucial role in human interactions, as they enable fast interchange of information for both business and entertainment purposes. The Internet is undoubtedly one of the largest specimen of this kind of system, providing interconnection and services to billions of users around the globe. It is its towering size and overwhelming complexity that arose the interest of lots of people trying to model, understand, and explain the laws of such a giant network.

One of the challenges that mostly attracted the attention of computer scientists is the design and operation of algorithms and protocols that make the Internet self-adjusting with respect to topological changes. Being so huge a system, it is impracticable to think of manually updating it every time a host joins or leaves the network. Instead, routing protocols have been conceived and deployed to automatically propagate the necessary reachability information.

Among the currently running routing protocols, BGP is responsible for ensuring connectivity between large administrative domains, essentially corresponding to Internet Service Providers, on a large scale. Devices running BGP do not have to care about the internal structure of ISPs, as they are in charge of routing traffic outside of their boundaries. Roughly speaking, BGP may be thought of as having a similar function to that of phone lines interconnecting a country to another. This is why BGP is often addressed as an interdomain routing protocol.

A careful understanding of the mechanics of BGP is essential to debug routing issues and to deploy consistent configurations. However, before digging into the dynamics of any routing protocol, it is essential to know the scenario it is being run on. Namely, the topology. Getting an interdomain topology is typically possible by accessing services around the world that monitor and collect BGP routing messages. However, this often impedes the disclosure of a substantial portion of the Internet, as the points of view may be limited. Also, some links may become active, hence visible, only under particular conditions, which include faults or traffic shifts. Part II of this thesis proposes methods and algorithms to overcome these limits and get a more comprehensive map of the administrative domains and their interconnections. Chapter 3 describes a method to exploit suitably wrapped up BGP messages to probe for links and domains that are not observable from collected routing data. Chapter 4 presents an alternative approach to the discovery of interdomain topologies, which is based on the extraction of information from archives that track the allocation of network resources over the Internet, known as Routing Registries.

Once the topology on which BGP operates is known, the interest shifts to the outcome of applying filters and manipulations on the routing messages being issued. By configuring policies that have nothing to do with the pursuance of optimal

routing, BGP allows to limit the spread of routing information and, to a certain extent, to purposefully alter it. Incorrect usage of policies or simply unexpected interactions among them may lead to abnormal or malfunctioning routing scenarios. Unfortunately, policies are usually restricted information, whereas they are likely to carry clues about the internal arrangement of an ISP. Part III contains an extensive analysis of the usage and effects of routing policies. Chapter 5 presents a survey and an evaluation of algorithms for inferring the commercial agreements between ISPs which, being closely related to business matters, is a piece of information that is hard to track down elsewhere. Chapter 6 introduces a theoretical framework to determine the optimal configuration policies to be deployed in order to achieve traffic engineering requirements, such as optimal bandwidth usage or cost sharing. Chapter 7 describes some methods to investigate the interactions among policies utilized at different ISPs. Probing techniques for attempting to reveal the policies running on live routers are presented, together with a model that allows to characterize and study persistent routing oscillations.

Solely introducing models for studying routing phenomena without having a chance to experiment them may bring about the risk of producing a bunch of poor founded results that is an end in itself. Part IV describes the emulation environment Netkit, which enables practicing networking scenarios as it would happen on real devices. Chapter 8 includes an introduction to the architecture of the environment and some proposed experiments that better substantiate the effectiveness of an emulation-based approach for supporting research activity.

Part I opens the thesis with an overview of BGP in Chapter 1 and some motivational arguments that justify this work in Chapter 2. Part V concludes with some final argumentations and proposals for future research, and includes an extensive bibliography.

Part I

Background and Objectives

Chapter 1

Interdomain Routing with BGP

> Does the road wind uphill all the way?
> Yes, to the very end.
> Will the day's journey take the whole long day?
> From morn to night, my friend.
>
> Uphill
> Christina Georgina Rossetti

The Internet is a huge network of hundreds of millions of nodes that exchange information with each other. Because of the dynamic nature and giant size of the network, routing of data within it takes place on two hierarchical levels. At a lower level, the devices controlled by an Internet Service Provider have a complete view of the ISP's own network and are responsible for the routing of Internet traffic within its boundaries (*intradomain routing*). At a higher level, routing between different ISPs (*interdomain routing*) takes place by means of the Border Gateway Protocol (BGP).

Since this thesis focuses on the study of interdomain routing, this Chapter provides an introduction to the fundamental concepts of BGP and to the principles of its operation.

1.1 BGP and Internet Routing

The Internet is partitioned into tens of thousands of administrative domains known as *Autonomous Systems* (ASes). Each AS is identified by an integer

number and typically corresponds to an organization, often an Internet Service Provider (ISP). The devices that make up a single AS are usually distributed over a wide geographic area (one or more countries). They are typically configured to use a single protocol (*Interior Gateway Protocol*) for internal routing and to implement coherent routing policies. Different ASes cooperate with each other to ensure global reachability of all destinations on the Internet.

The *border routers* of different ASes exchange routing information by means of the *Border Gateway Protocol* (BGP) [252, 100, 203], a *de-facto* standard for interdomain routing since the early nineties. A *peering session* between two BGP routers, called *peerers* or *peers*, is a TCP connection used to send BGP messages. Observe that peerings can involve routers at arbitrary distance (*multi-hop peering*). Since the configuration of BGP peerings affects the propagation of routing information, many studies focusing on BGP are primarily concerned with the identification of established peerings, a piece of information that is often referred to as the AS-level topology of the Internet. Most of these studies rely on the simplifying assumption that each AS is made up of a single router. This is usually reasonable and is supported by the fact that configurations on the border routers are expected to be consistent, even if there are exceptions [247]. For this reason, by extension we say that two ASes are *peers* or *neighbors* if there is at least one BGP peering session involving a router of one AS and a router of the other AS.

BGP peerings may also be configured between different routers of the same AS. In this case BGP works in the same way (except that some messages are slightly modified), but we speak of *internal BGP* or *iBGP* as opposed to *eBGP*.

BGP operates by propagating information on the reachability of contiguous blocks of IP addresses known as *prefixes*. The AS that owns a certain network, called *origin*, is responsible for first sending BGP messages concerning the corresponding prefix. Reachability information for that prefix is then selectively propagated from AS to AS depending on the configuration of BGP routers [58, 59, 121]. Traffic addressed to a certain prefix flows through the ASes that have propagated the corresponding announcement. Because of this mechanism of operation, in which a router sends to its neighbors information about the paths to be used to reach different destinations, BGP is classified as path vector protocol.

Figure 1.1 shows an example of BGP message. Nodes represent ASes and links represent BGP peerings. We assume that each AS is made up of a single BGP router. AS1 owns prefix `193.204.0.0/15` and first propagates information about it. Each BGP message consists, at least, of the prefix and the AS-level path to be used to reach it. For example, AS1 sends a BGP message

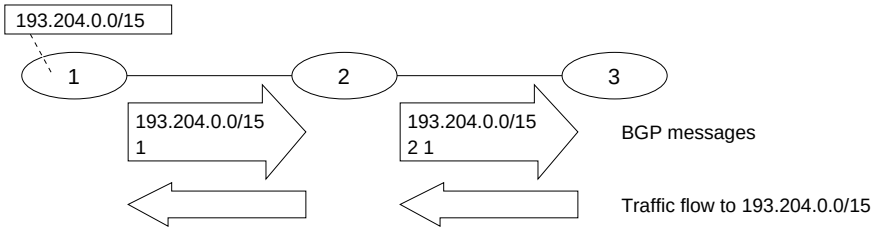


Figure 1.1: *An example of BGP message. Nodes represent ASes and links represent BGP peerings. Each AS is supposed to be made up of a single BGP router. AS1 owns prefix* `193.204.0.0/15`.

stating that `193.204.0.0/15` can be reached through himself. AS2 propagates this message after prepending its own AS number to the path: the resulting path (2 1) tells AS3 the ASes that must be traversed in order to reach `193.204.0.0/15`. Observe that traffic flows the opposite way with respect to BGP messages.

Routing information is advertised by using BGP messages called *updates*, which can be of two different types: *announcements* advertise the reachability of a prefix, while *withdrawals* communicate that a previously announced prefix has become unreachable. Figure 1.1 shows some examples of announcements. Information about the prefixes affected by a BGP update are contained in the *Network Layer Reachability Information* (NLRI) field of the update. The sequence of ASes a BGP announcement has passed through is stored in the *AS-path* field of the announcement itself. When an AS propagates an announcement, it prepends its own number to the AS-path contained in the announcement, so that the AS-path ultimately describes the sequence of ASes to be traversed in order to reach the destination prefix. A pair consisting of a prefix and the AS-path to be used to reach it is commonly referred to as a *route*. By extension, a route is an entry of the routing table that retains all the information of the BGP announcement it derives from.

A BGP message also contains other fields called *attributes*. An attribute may be:

- **Well-known** or **optional**, depending on the expected support by BGP implementations: all BGP implementations must recognize well-known attributes, while it is not required that they support all optional attributes.

- **Mandatory** or **discretionary**, depending on its presence in BGP messages: mandatory attributes must be included in every update message that contains NLRI, while discretionary attributes may or may not be sent in update messages.

- **Transitive** or **non-transitive**, depending on the fact that routers retain the attribute for propagation to other BGP routers or not.

The following is a list of the BGP attributes as documented in RFCs 4271 [252] and 1997 [164]:

- Origin (well-known, mandatory, transitive): it determines whether the prefix originated inside an AS, has been learned via the older EGP interdomain routing protocol, or is known by some other means. In most cases, the value of this attribute is 0, meaning that NLRI is interior to the originating AS.

- AS-path (well-known, mandatory, transitive): it indicates the sequence of ASes to be traversed in order to reach the prefix. More precisely, an AS-path is a sequence of segments each of which may be an AS-sequence or an AS-set. An AS-sequence is an ordered set of ASes, and contains the ASes that have been traversed by the announcement. AS-sets are unordered sets of ASes that have been introduced in order to support route aggregation. An AS-set usually contains the set of ASes of the routes from which the aggregate was formed. In practice, the vast majority of the AS-paths only contains a single AS-sequence.

- Next Hop (well-known, mandatory, transitive): defines the IP address of the router that should be used as the next hop to reach the prefix.

- Multi Exit Discriminator (MED) (optional, discretionary, non-transitive): may be used to influence a router in the choice among multiple exit points from an AS. The value of the MED is called *metric* and is only comparable between routes learned from the same neighboring AS.

- Local Preference (well-known, usage depends on the type of announcement, non-transitive): associates with a route a level of preference. If a router has to choose between different available alternatives to reach a prefix, it selects the one that has been assigned highest preference. BGP announcements directed to internal peers must be equipped with a local preference value, so that it can be distributed to all the border

routers of an AS; BGP announcements going to external peers must not include this attribute, as it contains information that is only useful for route selection at the local AS.

- Atomic Aggregate (well-known, discretionary, transitive): the presence of this attribute in a BGP update means that a router has aggregated some routes into the one contained in the update itself.

- Aggregator (optional, discretionary, transitive): this attribute contains the AS number and IP address of the router that has performed route aggregation.

- Community (optional, discretionary, transitive): the main purpose of the community attribute is to tag a group of destinations that share some common property. In general, the values of the community attribute may be assigned arbitrary meanings by network administrators, and a router may undertake arbitrary actions upon receiving an announcement tagged with a certain community. Nevertheless, some community values are well-known and can be used to limit the spread of routing information by telling routers not to advertise routes that they have learned via BGP announcements.

BGP routers store information about the prefixes they are aware of inside a table of routes called *Routing Information Base* (RIB). Information inside the RIB is updated according to the following process.

BGP speaking routers it store the BGP announcements they receive inside a set of routes called `Adj-RIBs-In`. Entries in the `Adj-RIBs-In` are then manipulated according to a set of import policies. This includes both discarding undesired routes and altering the BGP attributes of those that have not been filtered out. In this phase, routes that contain the router's own AS number in the AS-path are discarded because they would introduce cycles.

The leftover routes undergo a decision process [30] which aims at selecting, for each prefix, the route that is considered the best alternative to reach it. The decision process takes into account the values of several attributes of BGP announcements and is detailed in Figure 1.2. Observe that the selection steps are such that routes are chosen deterministically (even if this is not always true – see, e.g., [135]). Nevertheless, not all the implementations of BGP support all these steps: for example, Quagga [233] stops at step 6. That is, not all the implementations adopt deterministic selection processes, even if most of the commercial ones do.

Among the available routes to reach a prefix...

0. ...take into consideration only those for which the next hop is resolvable via an entry of the local router's forwarding table that has not been learned via BGP; among these routes...
1. ...select the route with highest weight (Cisco proprietary); in case of equal or unavailable weights...
2. ...select the route with highest local preference; in case of equal local preferences...
3. ...select the route that is configured to be locally originated by the router; if none is available...
4. ...select the route with shortest AS-path (AS-sets are considered of length 1, no matter how many ASes are in the set); if multiple routes with equally long AS-paths are available...
5. ...select the route with lowest origin type (IGP < EGP < INCOMPLETE); being equal the origin type...
6. ...select the route with lowest Multi Exit Discriminator; in case of equal MEDs...
7. ...prefer routes learned via eBGP over those learned via iBGP; in case multiple eBGP routes are available...
8. ...select the route with lowest IGP metric to the BGP next hop; in case of equal metrics...
9. ...select the route that has been announced by the router with lowest router ID.

Figure 1.2: *The BGP decision process used to pick the best route among the different alternatives available to reach a prefix.*

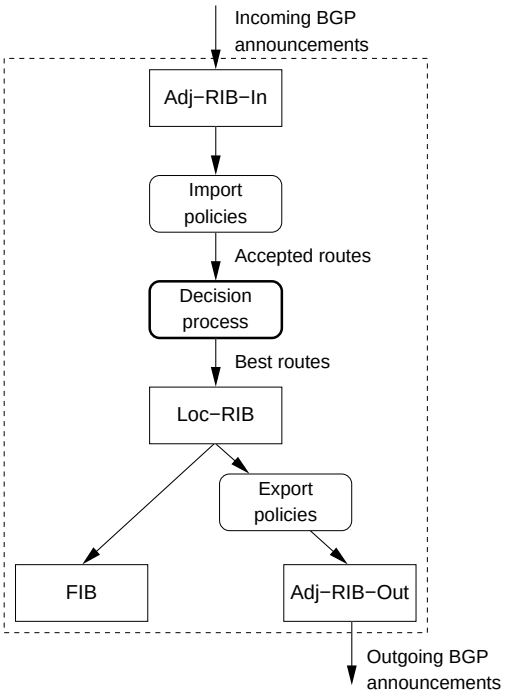


Figure 1.3: *Processing steps of BGP announcements at a BGP speaking router.*

The routes selected as best are stored in a set called `Loc-RIB`. The entries from the `Loc-RIB` are injected in the *Forwarding Information Base* (FIB) of the router and are actually used to forward packets. At the same time the entries in the `Loc-RIB` are processed according to export policies, which may discard some routes or alter the BGP attributes of the leftover ones. Entries that have not been filtered by export policies are stored in a set `Adj-RIBs-Out`. Routes in the `Adj-RIBs-Out` are then actually advertised by the router to its BGP neighbors. Figure 1.3 graphically describes this mechanism for processing announcements.

Notice that routers do not necessarily implement `Adj-RIBs-In`, `Loc-RIB`, and `Adj-RIBs-Out` as three separate data structures: this subdivision is only introduced to simplify the conceptual model.

Also consider that this model describes the processing of a cumulated set

of BGP announcements. In the real world, announcements come one after the other and must be processed separately. What happens is that, whenever a router receives a BGP announcement concerning a prefix, it updates its own RIB and runs the BGP decision process to refresh the best alternative to be used to reach that prefix. If this leads to a change of the route selected as best, the router sends a BGP announcement containing the new best route to all its neighbors. Therefore, routers always avoid advertising their full routing table, but rather announce only those entries for which the best choice has changed. Of course, new locally originated routes that are manually added to the configuration of the router are immediately announced. Withdrawals are only sent when a route becomes unavailable and there are no other alternatives in the routing table to reach its prefix. For this reason, network failures often lead to a potentially lengthy process [40] in which each router selects and announces alternative paths before eventually giving up and sending a withdrawal.

It may be the case that a BGP router receives announcements about two different prefixes such that one is contained in the other. This happens when one prefix is a smaller, more specific address block contained in a larger, less specific aggregate. If the router has to forward a packet to a destination for which two (or more) such alternatives are available, it uses the one with the most specific (i.e., longest) netmask. For example, if both `193.204.128.0/17` and `193.204.0.0/15` are contained in the router's RIB, a packet to `193.204.128.14` is forwarded according to the path of the most specific entry `193.204.128.0/17`, while a packet to `193.204.0.27` is forwarded according to the path specified in the only available entry `193.204.0.0/15`.

In order to reduce the amount of routing traffic and to save the processing power required to run the BGP decision process, routers implement a timer that limits the rate at which they send updates. The *MinRouteAdvertisementInterval* (MRAI) timer determines the minimum amount of time that must elapse between updates concerning a particular set of destinations sent by a BGP speaker to a peer. This rate limiting procedure applies on a per destination basis, although the value of the timer is set on a per BGP peer basis.

Moreover, in order to avoid routing instabilities, routers also adopt a *flap dampening* mechanism. A *flap* occurs when a prefix is repeatedly announced and withdrawn within a short span of time. Since it takes time and several BGP update messages to propagate these variations over the whole Internet, flapping routes have a harmful effect on router performance and on interdomain routing stability and must be suppressed. Dampening is implemented by assigning each route a penalty value that is increased every time the route flaps. If the penalty exceeds a threshold, the route is neither accepted from nor announced to other

peers any more. That is, the route becomes unavailable for a certain interval of time. During this interval the penalty is gradually decreased until the route becomes again usable.

1.2 Policy Routing with BGP

BGP allows to apply configuration settings that have nothing to do with the optimization of routing performance. For example, it is possible to express political or economical constraints by means of BGP configuration statements. Of course, it is also possible to take advantage of BGP configurations to implement some kinds of optimization on network traffic. The actions undertaken in processing BGP routing messages are called *policies*. Notice that there can be separate policies for handling received and sent BGP updates (import and export policies).

There are several possible usages of routing policies. For example, an ISP may configure appropriate policies that implement the commercial agreements it has established with its providers. On the other hand, the routers of an AS may implement policies that are aimed at achieving traffic engineering requirements. Other kinds of policies can be such as to ensure resilience in case of network faults.

In general, routing policies are implemented by leveraging the manipulation of the attributes of BGP announcements, typically the local preference, the AS-path, the Multi Exit Discriminator, or the community. One of the most commonly implemented routing policies is the *AS-path prepending*, a technique that deliberately "inflates" the length of an AS-path. An ISP is said to apply prepending whenever it propagates a BGP announcement by inserting its AS identifier more than once into the AS-path. For example, in the AS-path 1 2 2 2 3 AS2 has announced a prefix after prepending its own AS number three times. Since it affects the length of an AS-path, which is one of the factors that influences the BGP decision process, prepending is an effective way of attempting to drive the selection of routing paths performed at other ASes. Notice that this technique does not alter routing information and exploits a transitive BGP attribute, thus affecting routing choices even at distant ASes.

Other policies may simply make use of filters to throw away routes based on the AS they are announced from, on the prefixes they contain, or on the values of the community attribute. For example, a router can be configured not to accept announcements concerning `193.204.0.0/15` from AS2 because traffic to `193.204.0.0/15` should be routed through a different AS.

1.3 Collecting BGP Routing Information

Because of the complex interactions between policies deployed on routers belonging to different ASes, BGP routing poses interesting challenges which are the subject for several studies. In order to support investigations on the behavior of BGP, different services for collecting routing data have been made available.

Looking glasses are standard BGP routers that are configured to be worldwide accessible and that allow to perform different kinds of lookups. For example, a looking glass can be queried to get its routing table, have details about a certain route, obtain some statistics about the status of its peerings, or get information about route flap dampening penalties. Looking glasses can typically be accessed either via telnet or by using a web form. A comprehensive list of available looking glasses can be found at [188, 25].

A *route collector* is a router that continuously receives BGP updates from a usually large number of neighboring routers called *collector peers* but never propagates any routing information nor sends any kind of traffic. The view of the network provided by a route collector is typically better the more collector peers it has. The most important projects that provide route collectors are the RIPE Routing Information Service (RIS) [173] and the University of Oregon Route Views [198]. At the time of writing, the RIPE RIS provides 14 route collectors distributed in various locations all over the world and each having a number of peers that ranges from 4 to 141. Route Views provides 8 route collectors spread in various locations each having between 1 and 51 collector peers. Both the RIS and Route Views permanently store and make available the BGP updates received by the route collectors as well as their BGP routing tables, which are dumped on a regular basis. Both the RIS and Route Views also make their route collectors accessible as looking glasses, in order to be able to query for information about the current state of BGP routing.

In order to further support the study of BGP dynamics, some routers are configured to act as *beacons*. A *beacon prefix* is an unused block of network addresses which has a well-defined and publicly known schedule for announcement and withdrawal. By extension, a *BGP beacon* [258] is a router that is configured to periodically announce and withdraw one or more beacon prefixes according to the schedule. A few well-known beacons are listed at [172, 254].

Chapter 2

Motivations and Objectives

> And so Tom awoke and we rose in the dark
> And got with our bags and our brushes to work.
> Though the morning was cold, Tom was happy and warm
> So if all do their duty, they need not fear harm.
>
> The Chimney Sweeper
> William Blake

By looking at the subject of this work, a clear question might instinctively arise: why BGP, among the plenty of routing protocols?

It is well known by computer scientists studying networking topics that BGP, in its current version (4), is prone to the establishment of anomalous routing conditions, which may affect the reachability of some portions of a network. Besides this, BGP sometimes exhibits strange or unforeseen behaviors that are still difficult to capture into a general enough model. This Chapter briefly illustrates these and some other aspects that nourished the interest in analyzing BGP-related routing issues in depth, and it provides motivations justifying the studies presented in this thesis.

2.1 The Need for Topological Information

In order to be able to investigate on the behavior of a routing protocol, it is essential to know the topology of the network it operates on. This is typically difficult for different reasons: networks continuously evolve in topology, even

because a single user's PC has switched on or off its wireless card; moreover, even small local networks usually extend over a rather wide area (one or more buildings). This makes it unfeasible to think of gathering topological information by "on-the-field" inquiry. On the other hand, remotely querying network devices for their routing tables may not be feasible too, because access to those devices is very likely to be restricted. Therefore, information is often obtained by observing the messages exchanged by routing protocols or by sending probes aimed at discovering live hosts.

Unfortunately, even such an approach may lead to incomplete information. In the case of BGP, there are several route collectors and looking glasses which continuously receive and collect BGP updates. Yet, the propagation of these updates is limited because only the best routes are advertised and because routing policies impose constraints on the updates that can actually be sent. This prevents the collectors from observing all the BGP updates. Unfortunately, there are no available techniques to perform BGP-oriented active probings aimed at obtaining an AS-level topology, and even getting information out of static archives like the Internet Routing Registry database [216, 143, 60] is not an easy task because of the presence of out of date or inconsistent chunks of data.

Part II of this thesis presents two novel techniques to get AS-level topological information. Chapter 3 describes how standard BGP messages can be used to discover BGP peerings that are not observed during the normal operation of the Internet. Chapter 4 illustrates a methodology and a tool to accurately extract BGP peerings from the Internet Routing Registry, by taking into account and resolving inconsistencies and by digging for the most recent available information; the methodology we propose can also help in assessing the health status of the IRR.

2.2 Analysis of Routing Policies

One of the aspects that has arisen most interest while studying interdomain routing is the effect of deploying different and potentially conflicting sets of routing policies at different ASes. Part III of this thesis addresses the problem of getting to know the policies and of studying the outcome of the interplay between routing policies deployed at different ASes.

An interesting challenge that is dealt with in Chapter 5 is the discovery of commercial relationships between ASes. This knowledge would provide a deeper insight into the laws governing routing processes, and would constitute

a useful guideline for choosing connection strategies and device configurations. Unfortunately, information about the established commercial agreements is kept reserved by the operators, as it is considered critical for the economical strategies of an ISP. This brings about the need for methodologies to infer commercial relationships based on solely analyzing routing information. While some inference algorithms have already been introduced, it is still unclear to what extent the results they produce are realistic and of practical interest. Chapter 5 describes a methodology for assessing the quality of inferred relationships and describes the application of this methodology for the evaluation of state of the art inference algorithms.

Routing policies are often used as a means to perform traffic engineering. For example, a multihomed ISP may apply opportune filtering of outgoing BGP announcements in order to constrain incoming traffic to distribute on different upstream links. Operators sometimes perform traffic engineering on a trial-and-error basis, which may require several attempts that alter network conditions and potentially affect routing stability. Chapter 6 of this thesis describes theoretical approaches to determine the configuration parameters to be used to achieve different traffic engineering requirements. The proposed approaches include an integer linear programming formulation and a methodology that exploits a computational geometry model. The two approaches can be used to compute the optimal amount of prepending to be used in outgoing BGP announcements in order to achieve requirements ranging from optimal bandwidth allocation to fair distribution of upstream link costs.

Besides studying the effect of routing policies, it is also very interesting to determine how routing is influenced by the interplay of policies deployed at different ASes. Chapter 7 deals with this aspect. In particular, this thesis introduces a technique for determining whether an arbitrarily chosen AS-path that is not commonly observed on the Internet is feasible for sending traffic to a certain destination. A variant of this technique can then be used to determine the level of preference that a BGP router associates with two equally long AS-paths. Interactions between conflicting routing policies may also lead to unexpected outcomes such as routing oscillations. Chapter 7 also presents a model and some properties to determine the characteristics of BGP configurations that are likely to trigger routing instabilities. While this issue has already been extensively studied in the past, our model extends existing approaches by taking into account all the possible timings of exchanged BGP messages.

2.3 How to Experiment Routing by Emulation

One of the most common needs both for network administrators and for computer scientists studying networking is to experiment with network configurations before actually deploying them on devices. This helps in preventing trouble from happening on a running network and supports the validation of theoretical models. Performing experiments for these purposes is not feasible on a live network, as some tests may involve actions that disrupt connectivity or affect currently running services. On the other hand, no other devices may be available to perform the tests on a separate network set up for the purpose.

A possible solution to this problem is to take advantage of emulation. Part IV provides an overview of existing environments for the emulation of computer networks and describes in detail Netkit, a lightweight product based on open source software that allows to experiment networking on a single personal computer. Chapter 8 describes the architecture of Netkit and shows how it can be used to easily prepare and package complex network scenarios that can be easily redistributed for sharing with other people. Netkit is also equipped with a set of documented and ready to use virtual network scenarios that allow to immediately experiment with specific case studies. Netkit fully installs and runs in user space without the need of administrative privileges and it provides users with a familiar environment consisting of Debian based virtual Linux boxes as well as widely known routing software and networking tools. Chapter 8 also presents a sample application of Netkit to the study of a multihoming configuration, and shows how the environment can be effectively used to point out issues related to the interplay between BGP and IGP protocols.

Part II

Interdomain Topology Discovery Methods

Introduction

One of the most common ways to model the interaction of different components of a real world system is to represent the system itself as a network. The term *network* here is to be intended as a data structure that is able to capture an abstraction of the system components (usually corresponding to the nodes of the network) and of the information flows they exchange (usually assumed to traverse the network links). The data structure that best fits such representation requirements is the *graph*.

Modelling a system with a graph fits well for various kinds of analysis, depending on the specific context being considered. However, there is a piece of information that is common to all contexts and that should be accurately settled in order to get consistent results: the topology.

Unfortunately, getting topological information that likely reproduces the characteristics of the system being studied may not be easy. This is especially true in the case of the Internet, because of its huge size and continuously changing topology. To make things even more challenging, the Internet carries production traffic that is critical for many applications, which hinders shutting it down to take an offline snapshot at rest and brings about the need for effective non-intrusive exploration techniques.

There are several methods that have been proposed in the literature to find out as most as possible about the topology of the Internet. Depending on the kind of information they get topological data from, they can be classified into:

- **routing protocol based methods**: they reconstruct the topology by looking at information conveyed by routing protocols, as routing messages are known to purposefully spread topological information;
- **traceroute based methods**: they rely on the usage of traceroute-like tools in order to query the network for the paths to reach a meaningful set of destinations;

- **registry based methods**: they collect data from well known archives of routing information (Routing Registries [216, 142]), which contents are supposed to be reasonably aligned with currently adopted routing policies.

Topology discovery methods can also be classified depending on the strategy adopted for collecting network information:

- **passive methods**: they rely on the observation of routing information exchanged during the normal operation of the network; they are completely unobtrusive;
- **active methods**: they deliberately inject suitable information into the network in order to generate routing protocol message exchanges or to solicit responses by network devices.

Last, regardless of the specific techniques being adopted, topology discovery methods can be further classified according to the granularity of the topology they look for:

- **interdomain topology discovery methods**: they observe BGP routing messages and attempt to build a topological map representing the connectivity of the Autonomous Systems;
- **intradomain topology discovery methods**: they try to reveal a fine-grained map of the connectivity of routing devices and end systems; they are not necessarily limited to the topology of a single Autonomous Systems.

Many of the most significant attempts to figure out the structure of the Internet relied on active traceroute based intradomain discovery approaches. Among them, it is worth mentioning the CAIDA Skitter project [26, 15, 106, 103, 16, 105, 104] started in 1998 and the Rocketfuel project [200, 147, 165, 148], dating back to 2002. The goal of Skitter is to obtain connectivity and performance information on a large scale, while Rocketfuel is more aimed at mapping the topology of single Internet Service Providers.

Other past works focus on the behavior of the interdomain routing protocol BGP, which is still a very interesting matter for research. As a preliminary step, they usually present passive routing protocol based approaches to get an AS-level topology. The data sources that are mostly exploited for this purpose are the University of Oregon's Route Views [198] and RIPE NCC's Routing Information Service [173] collection projects. Works in this area [247, 12, 250, 76] attempt to achieve the highest level of accuracy in the topologies by overcoming the limitations

imposed by BGP routing policies on the spread of routing information (see, e.g., [73, 123] and by taking advantage of multiple data collectors in order to get a more complete view of the Internet (see, e.g., [158, 188]).

There are very few works in the literature that attempt to extract topological information solely from the Routing Registries [216]. The reluctance in using this data source is usually ascribed to the scarce overlapping between archived information and real routing behavior. Therefore, authors have been typically using Registry data only to complement topologies obtained with other techniques and to provide some preliminary estimation of the quality of data in the Registries (see, e.g., [163, 12]). An exception to this trend is the work by Siganos et al. [61], which digs into data from the Internet Routing Registry and describes a methodology to extract BGP peerings and routing policies from it.

This part describes two methodologies for discovering interdomain topologies which further improve the approaches proposed in the literature. Chapter 3 describes a technique which makes use of standard BGP announcements to reveal portions of the Internet that are not typically observed in a stable routing state. The effectiveness of the technique is then proved by experimentation on the live IPv6 and IPv4 Internet. As opposed to routing data based methods, Chapter 4 presents an improved methodology to extract BGP peering information from the Internet Routing Registry. Thanks to the accuracy put in processing Registry information, the proposed approach is shown to perform significantly better than state of the art methods. An automatic online service that extracts the peerings on a daily basis is also presented.

Chapter 3

Topology Discovery by Active Probing

What the hammer? What the chain?
In what furnace was thy brain?
What the anvil? What dread grasp
Dare its deadly terrors clasp?

The Tiger
William Blake

Due to the distributed nature of BGP, every Autonomous System only has an extremely partial view of the interdomain topology. However, it would be invaluable to network operators of an ISP to be able to know which paths might be traversed by BGP announcements for its prefixes in the presence of network faults or changes in routing policies. In fact, such knowledge would allow operators to develop more effective peering strategies, better assess the quality of their upstream connectivity, and to perform more effective traffic engineering.

This Chapter presents a technique that, by exploiting suitably crafted BGP messages, allows to discover portions of the interdomain topology of the Internet which are not typically observable in stable routing states. The probing primitives presented use standard BGP to influence how the announcements for a given prefix propagate through the Internet. An AS that originates a given prefix can use these primitives to discover which ASes and peerings can be traversed by the BGP announcements for that prefix.

3.1 Background

The BGP protocol is responsible for carrying routing information between different Autonomous Systems. The fundamental principles of its operation, its mechanisms and messages have already been introduced in Chapter 1. This Section provides some further details about the contents of BGP messages and some other preliminary definitions, which will come handy in the description of the topology discovery technique under examination.

BGP announcements for a particular prefix contain the AS-path for that prefix, which is the sequence of ASes to be used by traffic to the prefix. The BGP specification [252] states that AS-paths may be composed of an arbitrary number of AS-set or AS-sequence elements: the *AS-sequence* is an ordered list of ASes, while the *AS-set* is an unordered set of ASes. In practice, the vast majority of BGP announcements are composed of a single AS-sequence, possibly followed by an AS-set.

We say that an AS-path $A_n \dots A_2 A_1$, where A_1 is the origin AS, is *feasible* for a prefix p if the policies of each A_i permit A_i to announce p to A_{i+1} with AS-path $A_i \dots A_1$. Observe that the feasibility of an AS-path for a prefix does not imply that the AS-path is necessarily visible in the Internet: it only means that under certain circumstances it may be visible. The set of feasible AS-paths for a prefix p thus contains all the AS-paths that may be observed for p in the Internet. We note that the concept of feasible path has also been used in the literature on the stability of BGP with the name of *permitted* path (e.g. [191]). A peering between two ASes P and Q is feasible for p in the direction from P to Q if there exists at least one feasible path in which P immediately follows Q. In diagrams we shall represent a feasible peering from P to Q with a directed arc from P to Q. For example, in Figure 3.1(a) the directed arc between 10566 and 6175 indicates that the policies of AS 10566 permit it to announce the prefix `2001:a30::/32` to AS 6175.

We name *routing state* for a prefix p at a given time the set of best routes to p of each router in the Internet at that time. We empirically say that a routing state for p is *stable* if we have observed no BGP updates for p for a sufficiently large time interval. To obtain (partial) information about the evolution of the Internet routing state, we may use projects such as the RIPE NCC Routing Information Service (RIS) [173] and the University of Oregon's Route Views Project [198]. They deploy *route collectors* in specific points of the Internet to record BGP updates from routers in a number of ASes which we name *collector-peers*. Thus, the best routes of each collector-peer at any given time are known. The RIBs of the collectors and the updates

they receive are periodically dumped, permanently stored and made publicly available over the Web. Routing information can also be obtained by querying *looking glasses* [188], which are publicly accessible routers that may be queried for RIBs or other BGP information. Since looking glasses do not store any information over time, they can effectively be used only to get information in a stable routing state.

3.2 Related Work

There are several contributions in the literature which describe methodologies to infer the AS-level topology of the Internet. As already stated in the introduction of this Chapter, some of them rely on sending probe packets in order to discover the network topology, while others rely on the passive observation of BGP routing data.

The *skitter* [26, 15] project makes use of a distributed set of measurement points known as monitors which use probe packets to discover router level topologies and collect roundtrip time measurements. The monitors, about 20 at the time of writing [27], periodically send ICMP probe packets to a list of hundreds of thousands of destinations. The results are merged to obtain a router-level topology and, by mapping IP addresses to ASes based on information gathered from the Oregon Route Views [198] BGP routing table dumps, an AS-level topology is produced. While this approach ensures that the collected AS-paths closely match actual network layer routing, it has some drawbacks. For example, routers may aggregate different prefixes into a larger less specific prefix, in which case the AS that first announced the prefix cannot be determined. Moreover, some prefixes may be announced from multiple origins (for example, when anycast [23] is used). Both these situations can impair IP-to-AS mapping and bring about distortions in the AS-level topology obtained.

Further refinements to the technique used in skitter have been proposed by Mao et al. In [256] the authors adopt several methods to improve the quality of the mapping from IP addresses to ASes: they perform data cleanup in order to remove the effect of route changes, apply a heuristic to deal with hidden hops (that is, nodes that do not respond to ICMP probes), exploit DNS [157] and whois [112] information to resolve unmapped nodes, and try to single out mismatch patterns in IP-to-AS mapping in order to improve its quality. They also describe an algorithm to detect Internet Exchange Points, infrastructures where multiple ISPs meet to exchange traffic and routing information and whose presence typically cannot be observed using BGP routing

data. Since mapping IP addresses to AS numbers is a complex task, in [255] the same authors propose further improvements to their techniques based on a collection of optimization problems which can be solved efficiently by dynamic programming, together with an iterative improvement algorithm.

Much work has taken the approach of deducing the AS-level topology from BGP routing tables, usually those collected and made publicly available by projects such as Oregon Route Views [198] and the RIS [173]. An early example is described in [16], which uses the general purpose topology visualization tool Otter. This approach has been used by a large body of work on topics as diverse as evaluating the efficiency of BGP [73] and deducing the commercial relationships between ASes [121, 115, 64].

Subsequent work integrates information from BGP data with those collected from other sources. Chang et al. [76] augment Oregon Route Views data sets with information extracted from several looking glasses spread over the Internet and with AS-level adjacencies described inside the Internet Routing Registry (IRR) [22, 216]. Dimitropoulos et al. propose a topology discovery methodology based on the accumulation of AS-path information from BGP updates collected from Route Views, showing that observation of the network over a period of time brings about a linear increase of the number of links discovered [250]. The obtained topology retains the power-law behavior [145] which is typical of topologies built on the basis of BGP table snapshots, and reveals more links between low and medium-degree nodes, which are likely to be either backup links or links used for local communication between small ISPs. A similar approach is adopted by Zhang et al. in [12], in which they accumulate topological information from BGP updates and extract data from the Internet Routing Registry. In doing this, they attempt to determine the optimal size of the time window in which BGP updates should be collected: even if longer observation periods reveal more links, some of them may become inactive in the mean time. They show that an appropriate window of observation is about 60 days long.

The primary goal of most of the works mentioned above is to collect a reasonable topology of the network. Di Battista et al. in [63, 129, 32, 33] also propose a methodology and a tool (*BGPlay*) to visualize routing changes. Based on the integration of BGP data collected from Route Views [198] and the RIPE NCC Routing Information Service [173], the tool allows to graphically observe how traffic to a certain AS is routed and how routing paths change over time.

However, although these works document extremely interesting results that are essential to our understanding of the interdomain routing system, none

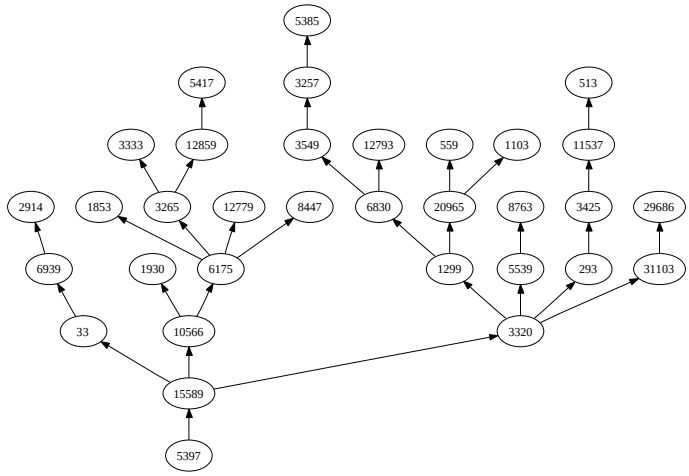
of them tackles the problem of determining whether a path can actually be traversed by a BGP announcement.

3.3 Probing Primitives

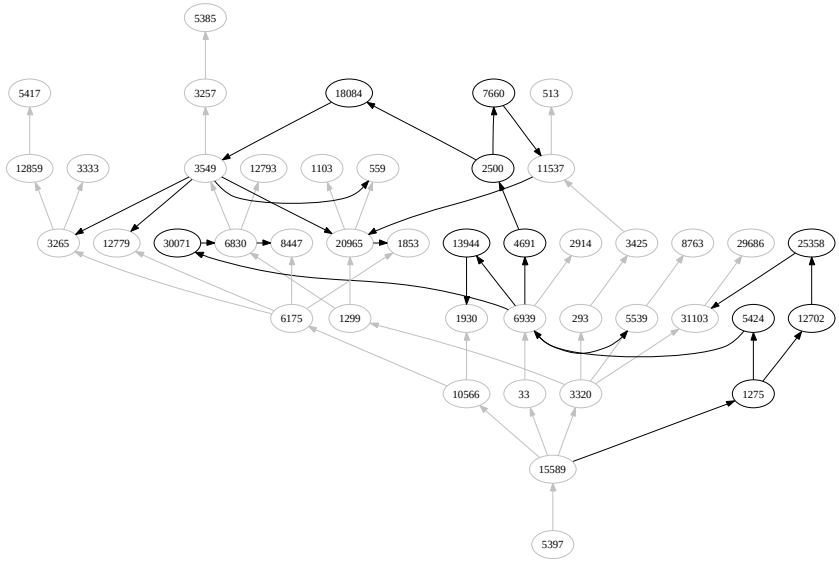
Unfortunately, despite the wealth of literature on interdomain topology, no method has been proposed to determine how BGP announcements are actually propagated. Even recent methods which make use of the passive observation of BGP dynamics [12, 250] still do not provide comprehensive information on how a specific prefix is propagated in the Internet and how it might be propagated in the event of link faults, changes in routing, or different traffic engineering strategies. This is because existing methods either attempt to discover the AS-level interconnections of the Internet, ignoring the effect of routing policies, or limit themselves to studying the paths to a particular destination at a given instant in time, and thus do not obtain any information on alternate paths that are permitted by routing policies, such as backup paths.

As an example, Figure 3.1(a) shows what the operators of AS 5397 may discover about the routing of their IPv6 prefix `2001:a30::/32` at a given time by querying the publicly available RIPE NCC RIS service [173]. An arrow from an AS A to an AS B means that an announcement of the specified prefix was made by A to B. The graph shows that AS 5397's upstream provider (AS 15589) propagates the announcement to AS 10566, AS 3320 and AS 33, but does not show, for example, that AS 15589 is propagating the announcement to AS 1275 as well. Figure 3.1(b) shows the additional information that AS 5397 may obtain by sending out a single BGP announcement using the techniques presented in this Chapter. We show in the following that a more thorough application of our techniques can yield almost three times the number of ASes and more than seven times the peerings visible using a standard RIS query.

In the following Sections we present our primitives, which are based on sending BGP updates for a prefix p from an AS Z and observing the effects using route collectors or looking glasses. While using these primitives, connectivity to p is disrupted; however, p may be a test prefix which does not carry production traffic. Since the vast majority of routing policies do not discriminate between different prefixes originating in the same AS based only on the prefix itself, the results obtained using p will hold for the other prefixes originated by Z as well.



(a)



(b)

Figure 3.1: *(a) What an operator of AS 5397 may discover about the routing of its prefix* `2001:a30::/32` *on Dec 30 2004 at 02:44:00 UTC using standard routing information services. (b) Additional topology discovered by sending one BGP announcement including ASes 33, 3320 and 10566 in an AS-set.*

3.3.1 Withdrawal Observation

The first probing primitive, which we name *withdrawal observation*, consists in sending a withdrawal for p and observing all the paths that become visible during the BGP convergence process. As first noted in [40], the withdrawal of a prefix causes a potentially lengthy (usually lasting several minutes) convergence process in which BGP explores alternate paths before concluding that the prefix is unreachable. Therefore, observation of the BGP updates during a withdrawal allows us to record the alternate paths which appear during convergence and are not visible in a stable routing state. We note that the idea of observing BGP updates to discover alternate paths is not new (see, for example, [250]); however, while the approach in existing work is that of passive observation, our primitive involves the purposeful generation of withdrawals in order to observe alternate paths.

3.3.2 AS-set Stuffing

The second primitive, which we name *AS-set stuffing*, consists in announcing p with an AS-path of $Z\{A_1, A_2, \ldots, A_n\}$, where $\{A_1, A_2, \ldots, A_n\}$ is an AS-set. Since a BGP router discards any announcement whose AS-path contains its own AS number, the ASes A_i will discard the announcement and will not propagate it to any other ASes. This effectively eliminates the ASes A_i from the Internet as far as the propagation of p is concerned; we name these ASes *prohibited* ASes. The observation of the resulting routing state and, possibly, of the convergence process, allows us to determine alternate feasible paths for p that do not contain the prohibited ASes. The use of an *AS-set* ensures that the length of the *AS-path*, which is one of the most important metrics used by BGP routers in the process of selecting a best route, does not depend on the number of prohibited ASes.

3.3.3 Effectiveness and Limitations

Despite the fact that withdrawals usually spark a significant number of BGP updates, which is likely to reveal peerings that are not visible in stable routing states, what can be seen using withdrawal observation is very hard to anticipate, since the peerings discovered depend entirely on the unpredictable order of BGP announcements. Consequently, a limitation of withdrawal observation is that it is not possible to perform targeted explorations, which impairs the use of this primitive for some applications discussed later on in this Chapter.

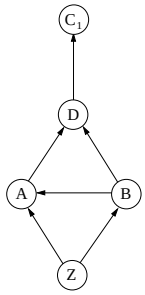


Figure 3.2: *A topology where AS-set stuffing cannot guarantee the discovery of the feasible peering between B and A.*

As for AS-set stuffing, it allows to do more than simply observe alternate paths: it allows us to alter the interdomain routing for p in a stable state. This has two important advantages. The first is that observing alternate paths does not require the collection of BGP updates, but may be performed using any commonly available looking glass, thus greatly increasing the number of observation points that may be employed. The second is that once the routing state is altered, other tools may be used to probe network connectivity and performance, thus allowing "what-if" analyses on Internet performance to be made.

There are intrinsic limitations to what we may observe using AS-set stuffing or withdrawal observation. Consider Figure 3.2. In the following we assume that Z is the origin AS; C_1 (and, in the following figures, C_2, C_3, and C_4) indicates a collector; an edge directed from x to y represents a peering that is feasible in that direction. In the case of Figure 3.2, the use of AS-set stuffing does not guarantee that it is possible to observe the peering between B and A, although it is feasible: we cannot force the observation of the peering in a stable routing state, because the only probes that can be performed are to prohibit A and/or B, but in every case the peering will not be visible since one of its endpoints is prohibited. However, a path such as C_1DABZ may be visible during BGP convergence, depending on the unpredictable order of updates propagated in the network.

3.4 Discovery Techniques

In this Section, we present several applications of the primitives introduced in Section 3.3.1 and 3.3.2, again considering the case of an AS Z that originates a prefix p. In the following, we use the concept of *feasibility graph*, which represents feasible peerings and how they are topologically related, and is thus a valuable starting point for further deductions. Given a set $\mathcal{S}$ of feasible paths for a prefix p, we name *feasibility graph* the directed graph whose nodes are the ASes and whose arcs are the peerings that appear in the paths of $\mathcal{S}$. Observe that, because of routing policies, not every path in the feasibility graph is feasible; however, since every arc in the feasibility graph is feasible, an arbitrary path $\mathcal{P}$ on the feasibility graph may well be feasible even if it is not in $\mathcal{S}$. We name *level* of a node X the length of the shortest directed path from Z to X.

3.4.1 Obtaining a Topology

There are several possible methods to obtain a feasibility graph. The simplest way is to query the route collectors for p, but the extent of such a graph is limited, as can be seen in Figure 3.1(a). A more effective approach is to use withdrawal observation. Although simple, this requires the collection of BGP updates, and thus does not permit to use looking glasses as observation points. Furthermore, the peerings discovered depend on the unpredictable order of BGP updates generated during convergence.

Another approach makes use of AS-set stuffing. By prohibiting a set of ASes, we may record one or more alternate paths. So, to obtain the most complete picture possible, the originating AS Z could in theory send 2^n announcements, including in each one an AS-set prohibiting one of the 2^n subsets of the n ASes in the Internet. Such a brute force approach is infeasible both because the number of ASes that may be included in an AS-set is limited and because of the long exploration times it would require. Therefore, we adopt the following strategy: begin with the directed AS graph seen by the route collectors at a certain instant and proceed level by level, starting from level one. For each level, prohibit all the known ASes in the level. At this point, either there will be no feasible paths to the collectors, or the announcements will propagate through new, previously unknown, nodes at the same level. Each new node and arc found is added to the feasibility graph. If new nodes in the same level have been found, insert them into the prohibited set; otherwise, empty the set of prohibited ASes and proceed to the next level. As an example, Figure 3.1(b)

shows the new nodes and arcs discovered starting from the situation in Figure 3.1(a) by announcing the AS-set $\{33, 3320, 10566\}$, which corresponds to all the known nodes at level two in the initial graph. After every BGP update, we wait a period of time to allow the network to converge and to limit the effects of route flap dampening [257]. To observe paths that are not visible in stable states, we examine all the updates received for p during the convergence period.

A more formal description of the algorithm is shown in Figure 3.3. We denote with `F` the feasibility graph, which is initially empty and is incrementally constructed during the execution of the algorithm, and with `F[lev]` the nodes of level `lev` in `F`. We use the same notation for a temporary graph `G`. We denote with `p` the prefix announced by Z. We name this algorithm the *level-by-level* exploration algorithm.

We note that this algorithm, in addition to the intrinsic constraints of AS-set stuffing already discussed in Section 3.3.3, suffers from further limitations. For example, in Fig 3.4, the algorithm does not guarantee that all the arcs will be discovered. In fact, nodes E and D will perform a choice on the paths $ZAEC_2$, $ZBEC_2$ (for E) and $ZADC_3$, $ZBDC_3$ (for D) based on their routing policies, and will therefore make some of the arcs $A \to D$, $B \to E$, $A \to E$, and $B \to D$ invisible from the collectors. The algorithm then excludes the nodes A and B on level 1 (it knows them both thanks to collectors C_1 and C_4), thus never revealing the invisible arcs.

Another possible algorithm based on the same primitive is as follows: once all nodes in a level l have been found, process each node in l in turn. For each one, prohibit all the other nodes in l, then progressively prohibit all visible nodes in level $l+1$ until no new nodes in level $l+1$ are found. Then empty the set of the prohibited nodes and advance to the next node in l. This approach overcomes the limitations of level-by-level exploration, but it requires many more updates and therefore much longer exploration times.

Finally, all these algorithms observe the network using all the route collectors simultaneously. However, in certain topologies, using only one collector at a time and merging the results at the end would discover more peerings. For example, in Figure 3.4, exploring the topology separately using level-by-level exploration, first using only C_2 and then only C_3, would also reveal the arcs $B \to E$ and $A \to D$, while exploring the topology using both collectors simultaneously might only discover the arcs $A \to E$ and $B \to D$.

```
# Obtain initial graph
F = query_route_collectors(p)

# Explore one level at a time
lev = 1
while F[lev] not empty:

    # Progressively prohibit ASes until no
    # new nodes are found in level lev
    newnodes = F[lev]
    while newnodes not empty:

        # Announce AS-set containing all
        # known nodes at level lev
        announce_as_set(F[lev])

        wait_for_bgp_convergence()

        # Query route collectors, merge new
        # nodes and arcs found into F
        previous = F[lev]
        G = query_route_collectors(p)
        F = merge_graphs(F, G)
        newnodes = F[lev] - previous

    lev = lev + 1
```

Figure 3.3: *Level-by-level exploration algorithm*

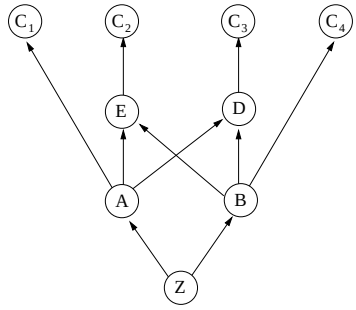


Figure 3.4: *Limitations of the level-by-level exploration algorithm: a topology in which not all the feasible peerings can be discovered.*

3.4.2 Operational and Ethical Impact

In this Section, we discuss the technical and operational factors that may limit the applicability of our techniques – including their impact on the interdomain routing infrastructure – and show how they are unlikely to be an issue.

Usage of Large AS-sets

To discover whether our BGP updates might have an operational impact on the Internet, we examined historical BGP data to determine the incidence of AS-sets and their sizes in the normal operation of the Internet. The results show that AS-sets, although rare, are constantly present in today's Internet. For example, data from the RIS route collector RRC03 for the month of February 2005 showed that every RIB dump contained between 500 and 700 entries which originated in an AS-set. The AS-sets observed are usually not very large: the largest AS-set observed during this period contained 15 ASes. However, AS-sets of unusual length have been observed before. For example, an AS-set containing 123 ASes was observed in January 2001 [71], and one containing 124 ASes was observed in June 2002 [96]. We know of no reports of adverse affects on the network caused by these announcements.

Impact on Network Devices

To determine how routers treat large AS-set, and so see whether our AS-set stuffing primitive can effectively be used in topology discovery in the real world, we also performed tests on equipment from popular router manufacturers. The BGP specification limits the length of an AS-set to 255 ASes, but the limits posed by commercial router BGP implementations are lower. This was not an issue in any of the topologies we tested, and should not pose a problem for all but very largest ISPs with hundreds of peers. Our tests on network equipment from various vendors showed that they correctly handle long AS-sets. Versions 12.2 and 12.0(17)S of Cisco IOS introduced the `bgp maxas-limit` command, which causes the router not to use or propagate announcements whose AS-path contains more than the configured number of ASes. The default value is 75; however, irrespective of the value of the setting, tests on certain releases of IOS showed that the routers would reset the BGP session if they received an AS-path more than 512 bytes long, i.e., where the total number of ASes (including both the ASes in the AS-sequence and in the AS-set) is greater than 254. We also tested with a Juniper M7i router running JUNOS 7.0R1.5, which

had no problems receiving and propagating AS-sets containing up to 255 ASes, the maximum length permitted by the BGP packet format.

As the example AS-sets used in our experiments show (see Section 3.5), such limits do not pose practical problems to our techniques in the topologies we tested, since the number of nodes involved is generally much lower. Further, we must take into account the fact that more ASes are added to the path as it is propagated in the Internet, and therefore our techniques must "leave room" for propagation, although we note that AS-paths longer than 15 ASes are rare in today's Internet [56].

As regards possible impact on router memory, we expect a large AS-set to consume about 200 extra bytes per prefix. This is negligible compared to the several megabytes of memory used by a full BGP table; also, since prefixes used for AS-set stuffing suffer connectivity problems, we expect its use to be restricted to test prefixes and limited to temporary experiments.

Route Flap Dampening

Another constraint to consider is that posed by route flap dampening, which limits the propagation of frequent updates for the same prefix. The exact effects of route flap dampening depend on the topology, but [257] suggests that the maximum length of time a route can be suppressed by dampening in today's Internet is approximately one hour. Therefore, dampening can be avoided by rate-limiting BGP probes to less than one every hour. Even at this rate, all the experiments we performed, including level-by-level explorations of nodes up to four levels away, were completed in a few hours. Rate-limiting the updates also has the effect of limiting the load placed on routers by the explorations, although this is negligible in the face of the order of the ∼15,000 updates per hour seen by Tier-1 routers.

Ethical Issues

One possible concern regarding our techniques is that they might cause operational problems: an update with a large number of ASes in the AS-set might cause confusion as to which AS actually originated the update, thus hampering debugging; and the presence of an AS number in an AS-path might suggest that that AS was involved in the update even though in fact it was not. However, we note that the conventional use of AS-sets for route aggregation already suffers from both these problems. For example, an AS-path of 1 $\{2, 3, 4\}$ implies that one of AS2, AS3, or AS4 originated the routing information that

generated the update, but does not specify which one. Also, since every BGP announcement is tied to a particular prefix, the origin of the announcement can easily be traced by querying public Internet registries. Finally, the BGP announcements used can be tagged using BGP community attributes [164] to indicate that they make use of AS-set stuffing, although community values are not always propagated by the ASes they traverse.

Another possible concern is that our use of AS-sets is not the use intended by the BGP specification. This is true, but we note that it frequently happens that protocols designed for a certain purpose are used in ways that the original designers did not foresee. Examples are Network Address Translation [153], which uses the TCP and UDP port fields to distinguish between hosts using private addresses, IP-in-IP tunneling [201], where a layer 3 protocol is carried by another layer 3 protocol instead of by a data-link layer protocol, and the use of duplicated TCP acknowledgements to implement fast retransmit and fast recovery congestion control [133].

Finally, it could be argued that placing the AS number of another AS in a BGP update should require permission from the AS in question. There is no technical reason for this; rather, such an action might be perceived as an improper use of an asset of another organization. We believe that this is a matter of policy to be discussed in the appropriate fora. Our opinion is that, since the techniques cause no technical problems, it is only a matter of determining their cost-benefit ratio. Since our techniques cannot reasonably be used if the originating AS needs to obtain permission from all the ASes that are close to it in the Internet topology, we believe that the question is simply one of deciding whether the techniques are useful for the Internet community or not.

3.5 Experimental Results

In this Section we describe the experimental setup and discuss the effectiveness of the prefix propagation discovery strategies and the limitations posed by route flap dampening. Finally, we evaluate our discovery techniques against more conventional methods.

3.5.1 Experimental Setup

Due to the innovative nature of our techniques, we first tested them in the IPv6 Internet, in order to limit the extent of any possible problems they might cause:

the IPv6 Internet is smaller than the IPv4 Internet, employs fewer legacy devices, and supports fewer mission-critical services. The IPv6 experiments were performed between November 2004 and April 2005. Once we were confident of the reliability and effectiveness of our techniques, we performed more limited testing on the IPv4 Internet in order to verify how the different topologies and operational practices in use affect our techniques. These experiments, with prior warning to various high-profile operational mailing lists [127, 128, 126], were conducted between June 2005 and July 2005.

All BGP updates were generated using custom software developed by the authors [31], and the effect of each BGP update sent was observed by means of the RIS database, which provides BGP data collected in real-time. IPv6 BGP announcements originated in AS 5397 using the prefix `2001:a30::/32`, while IPv4 BGP announcements originated in AS 12654 using the prefixes `84.205.73.0/24` and `84.205.89.0/24`. These IPv4 prefixes are reserved for BGP experiments and are announced by the RIS route collectors, which at the time of performing the experiments were present in 13 locations around the world. In order to approximate the situation of a small to medium-sized ISP, each prefix was announced by one route collector at a time.

3.5.2 Topology Discovery

To evaluate the effectiveness of our topology discovery strategies, we compared the feasibility graphs they generated to graphs obtained from the route collectors in stable routing states. The results are in Table 3.1.

As can be seen, our techniques observe between 20% and 200% more ASes and between 110% and 620% more feasible peerings than when active probing is not used. There is a difference in effectiveness between IPv6 and IPv4, which we believe to be due to the much more restrictive routing policies employed in the IPv4 Internet.

Method	IPv6		IPv4	
	ASes	Peerings	ASes	Peerings
Stable state	32	31	24	23
Withdrawal	94 (2.9×)	211 (6.8×)	28 (1.2×)	49 (2.1×)
Level-by-level	97 (3.0×)	222 (7.2×)	29 (1.2×)	55 (2.4×)

Table 3.1: *ASes and feasible peerings found by a standard RIS query, by withdrawal observation, and by level-by-level exploration.*

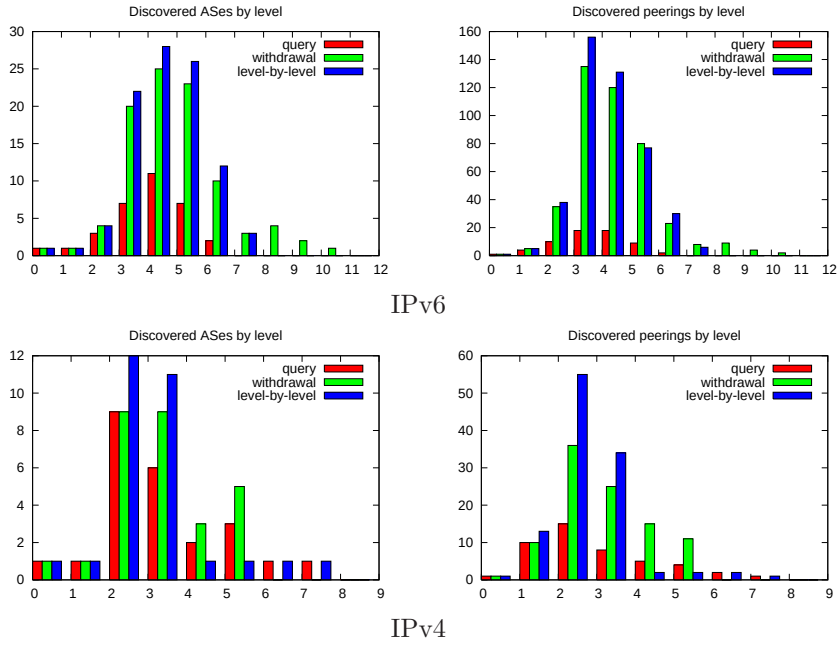


Figure 3.5: *ASes and peerings found by a standard RIS query, by withdrawal observation, and by level-by-level exploration, sorted by level, in IPv6 and IPv4. Every peering is counted twice, once at the level of one endpoint and once at the level of the other endpoint.*

The results obtained using AS-set stuffing are slightly better than those produced by withdrawal observation: although they are similar in terms of the number of nodes and peerings discovered, the topologies discovered are different. The graphs in Figure 3.5 show the level at which the new ASes and peerings are discovered. As can be seen from the graph, the topology produced by level-by-level exploration is more concentrated in the lower levels of the feasibility graph. Since the ASes that were discovered by the two methods are mostly the same, this means that certain ASes were discovered at a lower level by level-by-level exploration than by withdrawal observation. Therefore, by definition of level of a node, the topologies produced by level-by-level exploration are more accurate.

As an example of how AS-set stuffing may be used to discover the peers of

an ISP's upstream provider, Figure 3.6 shows a stable state feasibility graph for prefix `84.205.73.0/24` announced from RRC11 and observed at 14:49:59 UTC on July 5 2005. The graph shows that AS 13030 is the only upstream of the origin AS (AS 12654) and has 9 visible peers. A single announcement with an AS-path of 12654 {8210, 20932, 286, 13237, 702, 2497, 9044, 1239, 3320} at 14:55:35 UTC allowed the discovery of two previously unknown peers of AS 13030 which propagate the announcements for the prefix (in black). It is worth mentioning that a previously performed withdrawal observation had not discovered either of these ASes as peers of AS 13030.

3.5.3 Impact of Route-flap Dampening

As described in Section 3.4.2, effective use of our discovery strategies requires that the interval between updates be appropriately determined to avoid the effects of route-flap dampening. To provide a worst-case estimation of the effects on dampening of a single BGP update, we sent out withdrawals at varying time intervals. A single withdrawal typically generates a very large number of updates due to the path exploration process, thus greatly increasing the likelihood of dampening.

The first withdrawal was sent in a stable routing state, many hours after the last previously observed BGP update for the prefix. The second withdrawal was sent after approximately two hours; the third was sent approximately one hour later, and subsequent withdrawals were sent approximately every half hour. After each withdrawal, we observed the BGP convergence process in a 15-minute interval and measured the number of new ASes and peerings discovered with respect to the previous stable state. The prefix was then reannounced in preparation for the next withdrawal.

The IPv6 results in Figure 3.7 show that the number of new ASes and peerings found substantially decreases if the withdrawal is performed less than one hour after the previous withdrawal. Furthermore, approximately one hour after the first withdrawal, additional BGP activity for the prefix was observed, possibly due to paths that were dampened during the withdrawal being unsuppressed and reused. This suggests that the maximum time that a route is suppressed in the IPv6 Internet is approximately one hour. These results are in agreement with the results in [257].

Our IPv4 results, however, suggest that dampening continues to affect routing for longer than one hour: all subsequent withdrawal observations discovered about 35-40 new peerings and about 5 new nodes, compared with 43 new peerings and 17 new nodes for the first withdrawal. The results suggest that an

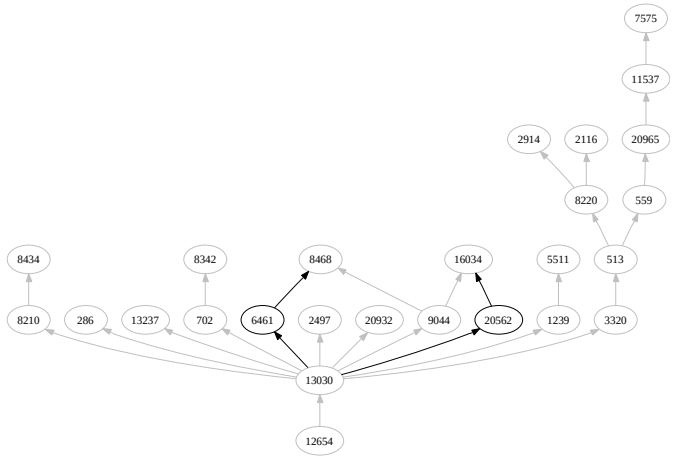
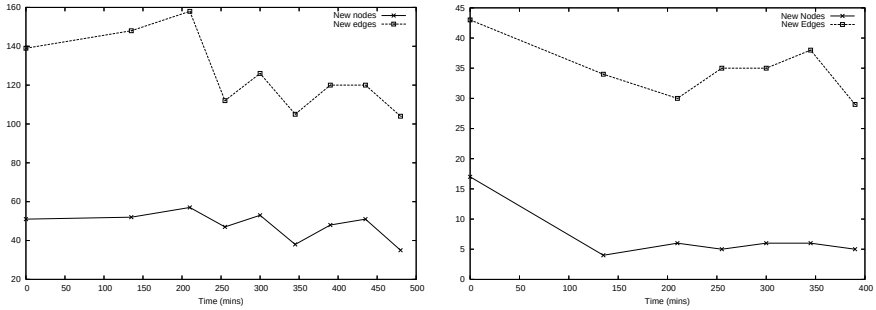


Figure 3.6: *Example of use of our topology discovery techniques: feasibility graph obtained by making a single announcement with AS-path 12654 {8210, 20932, 286, 13237, 702, 2497, 9044, 1239, 3320}. The two newly-discovered peers of AS 13030 are in black.*



IPv6 IPv4

Figure 3.7: *Effect of route flap dampening in IPv6 and IPv4. The data points in the graph correspond to withdrawal observations. The x coordinate of each data point represents the instant in which the withdrawal was sent, and the y coordinates represents the number of new ASes and peerings that were discovered during the convergence process, with respect to the preceding announcement.*

Date	Protocol	I	W	I only	W only
2005/02/23 09:54	IPv6	312	158 (51%)	175	21 (13%)
2005/02/25 10:03	IPv6	334	168 (50%)	189	23 (14%)
2005/02/27 15:18	IPv6	302	154 (51%)	174	26 (17%)
2005/07/05 00:00	IPv4	241	61 (25%)	181	1 (2%)

Table 3.2: *Comparison between the arcs in the graph W generated by withdrawal observation and those in the graph I induced by W in the global AS-graph C.*

appropriate interval between updates should be over two hours if the maximum effectiveness of our topology discovery is to be attained.

3.5.4 Comparison with the Full AS Graph

We also compared the results of our discovery strategies with more conventional interdomain topology discovery techniques. At a given time, we simultaneously obtained a feasibility graph W using withdrawal observation and a full AS graph C for all the prefixes announced on the Internet obtained from RIS data. We then compared W with the graph I induced by the nodes of W in C.

The results in Table 3.2 show that the graphs generated by withdrawal observation only have about 50% of the arcs of the induced graphs for IPv6 and 25% for IPv4. This shows that there is a substantial difference between existing topology discovery methods and our active probing discovery methods. The topology captured by the former is much richer; however, the topology discovered by our techniques consists of only those ASes and peerings that may actually be traversed by BGP announcements from p (and thus traffic flows to p) and is thus much more valuable from an ISP's point of view.

Finally, we note that between 13% and 17% of the arcs in the IPv6 graphs obtained by withdrawal observation were not visible in the graphs induced in the full AS graph. The IPv4 figure is only 2%. We suspect that this is due to the much greater redundancy of the IPv6 network, in which the widespread use of tunnels makes it easy to create a much denser connectivity mesh.

3.6 Conclusions

This Chapter presents methodologies for discovering how a prefix is propagated in the Internet. Our techniques allow the operators of an ISP to gain a greater understanding of how its BGP announcements are propagated than by using existing techniques. For example, it is possible for an ISP to use these techniques in order to know which paths might be traversed by BGP announcements for its prefixes in the presence of network faults or changes in routing policies. Extensive experimental results show the effectiveness of our methods both in the IPv6 and the IPv4 Internet. This has led us to believe that our techniques can be very useful for operators, while they still have a negligible impact on routing infrastructure.

The introduction of novel network probing techniques for discovering topologies also opens some interesting issues:

- The development of effective and computationally inexpensive topology discovery algorithms using our primitives would improve the ability to discover larger portions of the Internet with less announcements.
- The formal study of which peerings/ASes can be discovered by our techniques would further support the formulation of efficient discovery algorithms in terms of number of revealed arcs per probe and would give a better estimate of the limits of our methodologies.
- Combination AS-set stuffing with network measurements (e.g. RTT probes) would help in evaluating how network performance would be affected by alternate routing configurations.

Chapter 4

Topology Discovery based on Registry Information

Continuous as the stars that shine
And twinkle on the milky way,
They stretched in never-ending line
Along the margin of a bay:
Ten thousand saw I at a glance,
Tossing their heads in sprightly dance.

Daffodils
William Wordsworth

Most of the existing topology discovery techniques rely on the utilization of information collected during the operation of a routing protocol in a live network. As regards interdomain topologies, the best known sources of routing data are the Oregon Route Views project [198] and the RIPE NCC Routing Information Service [173], which provide BGP routing table dumps and BGP update messages collected by different monitors over time. The typical approach is to extract AS-level adjacencies from the AS-paths contained in BGP routing data. While processing BGP information in this way yields topologies that are aligned with actual routing policies, its drawback is that some adjacencies are not revealed by BGP data. This happens because some BGP peerings are only used (and, therefore, appear in AS-paths) in particular conditions, such as link failures or load imbalance.

An effective approach to augment the discovered topology is to combine information coming from different data sources. This includes both the usage of different observation points [188], and the integration of different kinds of data. For example, Zhang et al. [12], Mahadevan et al. [163], and Chang et al. [76], propose methods to construct topologies based on the combination of collected BGP data [198, 173], queries to looking glasses [188], and gathering of peering information from the Internet Routing Registry (IRR) [216, 143, 142]. There is only one relevant work [60, 61] which attempts to exclusively focus on IRR data.

This Chapter proposes an alternative approach to the discovery of topologies that only makes use of data from the IRR. The accuracy of information registered inside the IRR has been considered questionable in the past, but the methodologies introduced in this Chapter also try to address the problem of getting significant data out of the Registry. The presented approach purges IRR information and accurately processes it in order to extract BGP peering information from RPSL [22] descriptions of interdomain policies. The Chapter also describes an online service [39] that has been developed in order to automatically perform this extraction task on a daily basis. The service also allows to get several kinds of statistical information about the health status of the Registries. Taking advantage of the service also enables the understanding of how information inside the IRR varies over time.

4.1 Introduction and Related Work

The Internet Routing Registry (IRR) [216, 143] is a large distributed repository of information, containing the routing policies of many of the networks that compose the Internet. The IRR was born about ten years ago with the main purpose to promote stability, consistency, and security of the global Internet routing. It consists of several Registries that are maintained on a voluntary basis. The routing policies are expressed in the Routing Policy Specification Language (RPSL) [22, 44, 111], a replacement of the older RIPE-81 [196] and RIPE-181 [195, 184]. The IRR can be used by operators to look up peering agreements, to study optimal policies, and to (possibly automatically) configure routers.

There is a wide discussion about the current role of the IRR [60]. Some people consider it outdated and almost useless. Others have put in evidence its importance to understand the Internet routing and that it contains unique and significant information. Anyway, it is undeniable that the IRR keeps on being fed by many operators, that useful tools have been developed to deal

with the IRR (see, e.g., IRRToolSet [83]), and that several research issues on the Internet routing are, at least partially, based on the content of the IRR. However, as pointed out in [60, 61], extracting information from the IRR is far from trivial: the policies written in RPSL can be quite complex, the level of accuracy of the descriptions largely varies, and, also because of its distributed nature, the IRR contains many inconsistencies [180, 28, 42, 170].

In this Chapter we describe an online service [39], and its underlying methodology, that extracts peering information from the IRR. We believe that our service can have beneficial effects both for operators and for several research projects.

For example, the RIPE offers an IRR consistency check service (RRCC) [99, 174] that aims at detecting unregistered peerings. It verifies whether a peering that can be inferred from operational routing data is also described, in some form, into the IRR. We will show later that currently the RIPE service extracts peerings from the IRR in a way that is much less accurate than the one presented in this Chapter. Actually, the need of a better analysis of the content of the IRR is pointed out by the RIPE itself that considers this as a long term goal [174].

On the research side, Mahadevan et al. [163] presented a comparison of several characteristics of the AS-level topologies built on the basis of different data sources, including the IRR. They also proposed a metric to characterize such topologies. Zhang et al. [12] derived an AS-level topology combining IRR data with BGP routing information collected from multiple sources, such as Route Views [198], looking glasses, and route servers. They showed that the data from the RIPE Registry reveal topology information which cannot be found in other sources. Siganos et al. [61] developed a tool called Nemecis [223] that checks the correctness of IRR data and their consistency with respect to BGP routing table information. They argued that 28% of ASes have both correct and consistent policies and that RIPE is by far the most accurate Registry. Carmignani et al. [9] presented a service for the visualization of IRR data. We shall compare the level of accuracy of the methods for extracting peerings from the IRR used in the above papers with respect to those presented in this Chapter.

The methodology we propose to extract peerings from the IRR, and the service that implements it [39], are based on: a consistency manager for integrating information across different Registries, an RPSL analyzer that extracts peering specifications from RPSL objects, and a peering classifier that aims at understanding to what extent these peering specifications actually contribute to fully determine a peering. A peering graph can therefore be built with differ-

ent levels of confidence. We prove the effectiveness of our method by showing that it allows to discover many more peerings than the state of the art. As a side effect, our study also highlights how the different RPSL constructions are actually used to specify peerings.

4.2 Background

This Section presents an overview of how the Internet Routing Registry is structured and what its role is. Some fundamental concepts of the RPSL language, together with tools to manipulate it, are also introduced.

There are many publicly available Registries that describe both the allocation of Internet resources and BGP routing policies. The *Regional Internet Registries* [45] (e.g., RIPE [171], ARIN [11]) are in charge of maintaining information over wide geographic regions. The *Local Internet Registries* (e.g., VERIO, LEVEL3) describe the policies of the customers of a specific ISP. Taken together, all these Registries form the *Internet Routing Registry* (*IRR*). The main purpose of the IRR is to support a consistent global configuration of routing policies. It is also possible to automatically create BGP filters and router configurations from Registry information by using tools such as IRRToolSet [83].

The registration and maintenance of routing policies are performed on a voluntary basis by network operators, who may register the policies at one or more Registries. As a consequence, information therein may be incorrect, incomplete, or outdated. Indeed, some large ISPs and Internet Exchange Points rely on the IRR for route filtering and do not allow their customers to participate in BGP routing unless they document their routing policies in a Registry.

The routing policies stored in the IRR are described using the *Routing Policy Specification Language* (*RPSL*) [22, 44] or its more recent variant *RPSLng* [111], which introduces support to both multicast and IPv6. RPSL is an object-oriented language that defines 13 classes of objects. Routing policies are described in the `import`, `export`, and `default` attributes of `aut-num` objects. In turn, `aut-num`s may reference other objects that contribute to the specification of the policies, such as `as-set`s[1]and `peering-set`s.

Figure 4.1 shows a portion of an RPSL `aut-num` object from the RIPE Registry which describes the routing policies of AS137 (last updated 08/30/2000).

[1]RPSL's `as-set`s are descriptive data structures that group several `aut-num`s together; they must not be confused with BGP's AS-sets, which are unordered portions of an AS-path (see Section 1.1).

```
aut-num: AS137
import: from AS20965 action pref=100;
      from AS1299 action pref=100;
      accept ANY
[...]
export: to AS1299 announce AS-GARR
[...]
changed: noc@garr.it 20000830
source: RIPE
```

Figure 4.1: *A fragment of RPSL object.*

The portion of the `import` (`export`) attribute following the `from` (`to`) keyword is a very simple example of *peering specification*. The object indicates that AS137 accepts any route sent to it by AS20965 and by AS1299 and propagates to AS1299 all the routes originated by ASes belonging to the `as-set` named AS-GARR (an `as-set` is an RPSL object that specifies a set of ASes). This implies that AS137 has a peering with AS20965 and AS1299.

In our service we make use of *Peval*, a low level policy evaluation tool conceived to write router configuration generators. Peval is part of the *Internet Routing Registry Toolset* (*IRRToolSet*) [83] suite. Peval takes as input an RPSL expression and evaluates it by applying RPSL set operators (`AND`, `OR`, `NOT`) and by expanding `as-set`s, `route-set`s, and AS numbers into the corresponding sets of prefixes. Alternatively, Peval can stop the expansion at the level of ASes. We access IRR data also through the *Internet Routing Registry Daemon* (*IRRd*) [217], a freely available stand-alone IRR database server supporting both RPSL and RPSLng.

4.3 A Methodology and a Service to Extract Peerings from the IRR

The methodology we use to extract information from the IRR consists of the following main blocks, each performing specific tasks. The details of each block are provided in the following Sections.

Basic Information Analyzer: its goal is to provide preliminary information

on the Registries. For example, it computes the number of `aut-num`s and `as-set`s inside each Registry. Also, it computes the amount of overlap of information between pairs of Registries. Further, it deals with the evolution over time of Registries, measuring the number of everyday updates. Such basic information is useful for giving a correct interpretation of the results obtained by using the methodology.

Inter-Registry Consistency Manager: the IRR is, in itself, a distributed system consisting of different nodes (the Registries) that may contain inconsistent information. Starting from a set of Registries, this block constructs a purged new consistent version of the IRR. RPSL objects with the same key appearing in different Registries are compared. A choice is done relying on the timestamp of the last change and in terms of the semantics of the attributes.

RPSL Peering Specification Analyzer: this block is responsible for extracting from the IRR the peering relationships between ASes. This is done by analyzing the body of RPSL objects. The relationships extracted in this phase are *candidate peerings* for the subsequent elaboration. In this step we also evaluate the current usage of the RPSL syntax constructions for expressing peerings. The implementation of this block exploits IRRd [217] and Peval [83].

Peering Classifier: this block defines a classification of the computed candidate peerings according to their relative matchings in order to understand to what extent they contribute to fully specify a peering. The output of this step is a peering graph, that can be constructed with different levels of confidence.

The methodology has been implemented and is available as an online service at [39]. Each of the above described blocks roughly corresponds to a component of the service. The following information is automatically produced by the service, on a daily basis:

(i). General statistics on the IRR (number of objects defined in each Registry, amount of overlapping information between Registries, etc.).

(ii). A set of pairs of ASes, corresponding to peering relationships extracted from the IRR. Each pair is labeled with information about the context where it has been found, like the type of policy and the Registry.

(iii). A classification of the extracted peerings according to the way peering specifications contribute to fully specify the peering.

(iv). Other statistics about the current usage of RPSL constructions by the operators.

The service can also be queried to obtain statistical information about the evolution of Registries over time and their health status.

4.3.1 Reference Data Set

The reference Registry data we use throughout this Chapter to show the results obtained by applying our methodology has been downloaded from [171, 144] on 03/31/2006. At that time there were 68 Registries available for download, which are listed in Table 4.2. The Registries are sorted according to their size in terms of number of `aut-num` objects registered inside them (2nd column). We omit void Registries and those which the mirrors failed to collect data from.

Table 4.1 indicates the amount of overlapping between the largest Registries. For each pair of Registries (R_{i_1}, R_{i_2}) the table provides the number of `aut-num` objects that are registered both in R_{i_1} and in R_{i_2}. The main diagonal (R_i, R_i) reports the count of `aut-num`s appearing in Registry R_i only.

Figure 4.2 gives an idea of the amount of work of the operators on the IRR over time. Namely, it shows the daily percentage of size variation of the RIPE Registry (that is by far the most popular) over the period 11/14/2005–04/26/2006. The plot shows that the RIPE Registry keeps on being updated on a regular basis and that it grows of about 2% per month. Our reference date (arrow in the plot) has been selected to be one with an average number of updates.

	apnic	arin	radb	ripe	verio
apnic	2688	1	423	19	113
arin	1	463	37	7	14
radb	423	37	2037	50	45
ripe	19	7	50	11238	23
verio	113	14	45	23	310

Table 4.1: *Overlapping* `aut-num`*s between Registries.*

ripe	11468	92%	host	10	90%	reach	2	50%
apnic	3299	84%	ottix	9	33%	nestegg	2	100%
radb	2695	77%	csas	9	100%	gw	2	100%
arin	555	41%	rogers	8	100%	bendtel	2	50%
verio	498	42%	risq	8	100%	univali	1	100%
dodnic	254	11%	crc	8	62%	soundinternet	1	100%
altdb	249	63%	deru	7	0%	panix	1	0%
savvis	180	75%	sprint	6	16%	openface	1	100%
epoch	137	100%	bcnet	5	60%	koren	1	100%
level3	126	40%	vdn	4	25%	gts	1	100%
bell	74	98%	rgnet	4	100%	gt	1	100%
aoltw	53	3%	mto	4	25%	fastvibe	1	100%
jpirr	43	34%	easynet	4	100%	eicat	1	100%
sinet	28	10%	digitalrealm	4	100%	ebit	1	100%
arcstar	16	6%	look	3	100%	area151	1	100%
chtr	11	0%	retina	2	50%			

Table 4.2: `aut-num` *objects in the Registries before and after resolving inter-Registry inconsistencies.*

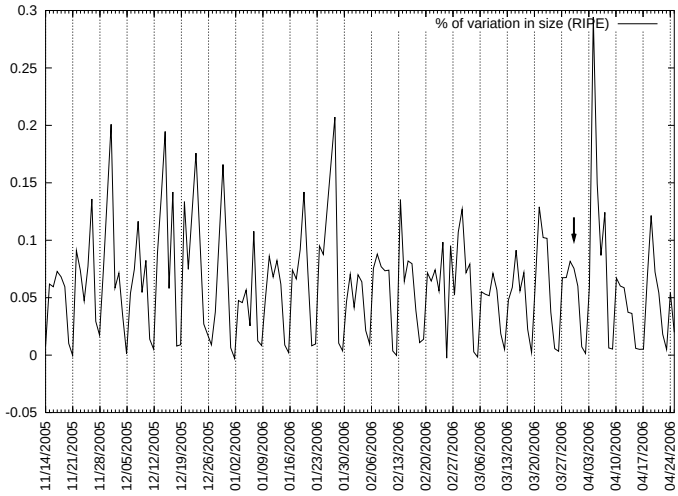


Figure 4.2: *Daily growth of the RIPE Registry.*

4.3.2 Integrating Information from Different Registries

RFC 2622 [22] considers the IRR system as a whole. However, the IRR consists of several Registries, and the same object may be defined in many of them. For example, in our data set, AS2510 is registered both in APNIC and in JPIRR. Of course, the presence of multiple definitions of the same object can lead to inconsistencies. Our Inter-Registry Consistency Manager takes care of resolving them. It takes as input a set of Registries and processes them in order to build a new repository where each RPSL object is defined only once. Whenever it detects for a certain RPSL object the presence of multiple definitions (possibly coming from different Registries), it examines all the definitions in order to determine which of them contains the most significant information. Such definition is kept in the final repository, while all the others are discarded. In what follows, a triple $(x; y; z)$ represents a number of `aut-num`s, `as-set`s, `peering-set`s, respectively. In our data set we have $(19,800; 7,798; 149)$ overall definitions. Among them, $(18,735; 7,478; 149)$ are unique. Hence, potential inconsistencies affect at most $(1,065; 300; 0)$ objects.

If an RPSL object is defined multiple times, the most informative definition is selected. We call *stub object* an `aut-num` object which misses information about BGP policies or a `set` object which misses the specification of the set members (consider that some attributes of RPSL objects are optional). Operators sometimes use stub objects as "placeholders" which can be referred to inside other parties' BGP policies. Our data set contains $(3,133; 206; 11)$ stub definitions. If we detect that an object appears in more than one Registry, we discard its stub instances. Since stub objects do not provide useful data about the existence of peerings, this does not cause any loss of information.

However, it may still be the case that several Registries contain non-stub instances of a single RPSL object. If this happens, we select the instance with the most recent update timestamp, that is contained in the `changed` attribute. After removing the stub definitions and selecting the most recent timestamp, the potential inconsistencies affect at most $(44; 77; 0)$ objects. Note that, even if the `changed` attribute is optional, in our data set there is only one definition that misses the timestamp over 2,271,446 objects in the IRR.

Yet, if there are (at least) two instances with the same most recent date, we select the definition belonging to the Registry with highest *rank*. We rank the Registries according to their size. This choice is somehow arbitrary. However, a Registry with a higher number of objects often provides more reliable information than the others. Also, as shown above, the choice impacts very few objects. Last, we have inspected the objects that have multiple definitions

with the most recent date and discovered that in most cases their definitions coincide. Of course, other rankings could be applied without impacting the general structure of the method.

The third column of Table 4.2 shows the percentage of the remaining `aut-num` objects per Registry after running the Inter-Registry Consistency Manager. It is interesting to observe that RIPE has the highest absolute number and the highest percentage among the top 5 Registries.

4.3.3 Extracting Peering Information from the IRR

In this Section we detail the procedure we apply to extract BGP peering information from RPSL data. As already stated in Section 4.2, peering specifications only appear in the `[mp-]import`, `[mp-]export`, and `[mp-]default` attributes of `aut-num` objects. Hence, `aut-num`s are the starting points of the peering extraction.

Figure 4.3 shows a fragment (25 lines, `ASX1-ASX13` represent ASes) of RPSL code that puts in evidence many of the problems encountered while discovering peerings in the IRR. Even if this example is not taken from the real life IRR, it is a patchwork of pieces of code that are quite common in RPSL objects. We now show how to extract from this fragment the peerers of `ASX5`.

By scanning this code with the RIPE RRCC [174] scripts [234], the following peerers of `ASX5` are found: `ASX1`, `ASX3`, `ASX12`, `ASX13`. They come out by examining the lines 17, 19 (`import from`), 24, 25 (`default to`). However, these peerings are neither correct nor complete. On one hand, the peering between `ASX5` and `ASX1` does not hold, since the `refine` semantics require to compute the intersection between `ASX2:AS-Z2` and ASes `ASX1`, `ASX2`. On the other hand, there are peerers of `ASX5` that have not been discovered. The peerers `ASX6` and `ASX7` can be inferred only by considering all the ASes that belong to the `peering-set` used at line 20 and defined at lines 1-5. Further, the peerers `ASX8` and `ASX10` can be inferred only by considering all the ASes that belong to the `as-set` used at line 21 and defined at lines 6-8,11-13. Finally, the peerers `ASX2` and `ASX11` are not discovered because the scripts in [234] support neither multiple peerings appearing in the same `from` expression, nor the `mp-export` attribute.

We now describe our method for extracting peerings from the RPSL code. In particular, we describe how we build a set of candidate peerings which we use later to identify peerings.

For each `aut-num` object A we compute three sets $import(A)$, $export(A)$, and $default(A)$ of candidate peerers corresponding to the `[mp-]import`,

```
1.  peering-set: ASX1:PRNG-Y1
2.  peering: PRNG-Y2
3.  peering: ASX6

4.  peering-set: PRNG-Y2
5.  peering: ASX7

6.  as-set: ASX1:AS-Z1
7.  members: ASX8, ASX9
8.  mbrs-by-ref: MNTR-ASX1

9.  as-set: ASX2:AS-Z2
10. members: ASX2, ASX4

11. aut-num: ASX10
12. member-of: ASX1:AS-Z1
13. mnt-by: MNTR-ASX1

14. aut-num: ASX5
15. import: { from ASX2:AS-Z2 accept 100.0.0.0/8;
16.         } refine {
17.           from ASX1 ASX2 accept 100.1.0.0/16;
18.         } except {
19.           from ASX3 accept 100.1.1.0/24;}
20. export: to ASX1:PRNG-Y1
21.         to ASX1:AS-Z1 except ASX9
22.         announce 100.1.1.0/24
23. mp-export: to ASX11 at 2001::1 announce 2001::/48
24. default: to ASX12 action pref=10
25. default: to ASX13 100.1.1.1 at 100.1.1.2
```

Figure 4.3: *Sample fragment of RPSL code pointing out the problems encountered while extracting BGP peering information from the IRR.*

[mp-]export, and [mp-]default attributes, respectively. We describe our procedure with reference to the import attributes. The other attributes are processed in a similar way. If A defines a private AS, it is discarded (it should not be visible in the Internet).

An import attribute may contain a simple or a structured policy. A simple import policy may contain several peering specifications. In this case $import(A)$ is the union of the candidate peerers corresponding to these peering specifications. If a peering specification is a peering-set, it is recursively expanded into its members. If it involves information about routers (e.g., interfaces, inet-rtrs, rtr-sets), they are removed. We keep only AS names, as-set names, set operators, and the keyword AS-ANY. The resulting expression is evaluated by using Peval [83]. The output of Peval, consisting of a set of ASes, contributes to the set of candidate peerers.

A structured policy is a policy that has except and/or refine operators. In this case, $import(A)$ is still the union of several candidate peerers, but these candidate peerers are determined in a different way. First, we extract the two arguments of the except (refine) operator, which are simple policies. Then, we process these policies as above, thus obtaining two sets of peerers. The union (intersection) of these two sets is our set of candidate peerers. If there are multiple except/refine expressions, we process them iteratively. Consider that, as an amendment to RFC 2622 [22], RFC 4012 [111] confirms that the RPSL specification does not allow the use of nested except/refine expressions.

If an aut-num has many import attributes the above procedure is repeated for each one.

Finally, private ASes in $import(A)$ are removed.

Some technical issues are worth being pointed out. For example, a peering specification may contain the AS-ANY keyword. AS-ANY is either used "alone" (e.g. import .... from AS-ANY ....) or in a structured policy. In the first case one could argue that there is an AS that has a peering with all the other ASes, which is clearly unrealistic. Hence, in this case we discard the peering specification. Else, if AS-ANY is used inside a structured policy, we apply the above algorithm. Last, observe that also inet-rtr objects may contain information about peerings. However, we do not consider such peerings meaningful unless they appear in an [mp-]import, [mp-]export, or [mp-]default attribute of an aut-num.

Table 4.3 shows the actual usage of RPSL constructions in the specification of peerings. This gives an idea of the syntactical expressions that are most frequently used by the operators. The second column of Table 4.3 also shows

`aut-num` objects	Action	Uses Peval	Occurrences
having a `default` attribute	Supported	No	4,851
having an `mp-import`, an `mp-export`, or an `mp-default` attribute	Supported	No	220
having a `peering-set` object in (*)	Supported	No	16
having an `as-set` object in (*)	Supported	Yes	939
having AS-ANY in (*) without further specifications	Discarded	No	660
having AS-ANY in (*) within a `refine` expression	Supported	No	24
having an `and`, an `or`, a `not`, or an `except` operator in (*)	Supported	Yes	5
having a `refine` or `except` expression in (*)	Supported	No	29
registering a private AS	Discarded	No	1
Private ASes in (*)	Discarded	No	86
`inet-rtr` objects having `peer` attributes	Discarded	No	217

(*)= "an `[mp-]import`, an `[mp-]export`, or a `[mp-]default` policy"

Table 4.3: *Incidence of different RPSL constructions in the specification of peerings.*

ripe	342995	bell	974	risq	67	look	16	soundinternet	8
verio	118999	fastvibe	968	sinet	50	eicat	15	gw	8
radb	19309	level3	558	ottix	38	nestegg	14	digitalrealm	8
apnic	13979	epoch	439	jpirr	38	mto	14	univali	6
reach	9402	dodnic	389	csas	36	area151	14	gts	2
savvis	1593	gt	219	retina	22	openface	10	easynet	2
arin	1233	rogers	134	crc	22	bendtel	10	aoltw	2
altdb	1068	host	79	bcnet	18				

Table 4.4: *Peering candidates per Registry.*

whether the listed constructions are supported by our processing engines or are simply discarded because they do not provide useful information. The third column specifies whether the policy evaluator Peval [83] is invoked in resolving each construction.

Table 4.4 shows the number of peering candidates extracted from the Reg-

istries with respect to the reference data set of 03/31/2006.

4.3.4 Classifying the Peerings

Once a peering candidate has been extracted from the IRR, it is classified according to the following two categories. Let A and B be the two ASes participating in the peering candidate. $A \xrightarrow{E} B$ represents the fact that A registered an export policy allowing BGP announcements to be sent to B. In turn, $A \xrightarrow{I} B$ indicates that B registered a policy according to which B accepts incoming announcements from A. The peering candidates are also tagged with the Registries from which they have been extracted.

At this point, the peering candidates are used to determine whether there actually is a peering between two ASes. For example if, for two ASes A and B, we have found four peering candidates of type $A \xrightarrow{E} B$, $A \xrightarrow{I} B$, $A \xleftarrow{E} B$, $A \xleftarrow{I} B$, it means that both A and B have fully considered their partner in the peering. Hence, we call this peering "full peering" (A—B). Of course, there can be cases when the policies describe a peering only partially. For example, we might have only $A \xrightarrow{E} B$, $A \xrightarrow{I} B$, in which case the announcements from A to B are described in the policies, while there is no evidence of policies allowing announcements from B to A. We call this situation "half peering" ($A \overset{1/2}{—} B$).

Table 4.5 shows all the possible relationships between two ASes (combinations that can be obtained by swapping A and B are omitted). The column Peering Type associates a symbol to each possible situation. The column # of Peerings counts the peerings of each category. The column Single Registry reports the percentage of cases where all the candidate peerings contributing to the peering are in a single Registry. We detail this percentage for the case of the RIPE Registry. A "self peering" refers to an AS that registers a peering with itself.

The peering types of Table 4.5 can be used to construct Internet topologies with different levels of confidence.

4.4 Comparison with the State of the Art

In order to compare the peerings discovered with our techniques with those discovered with previous approaches we ran on the same data set we used in our experiments the piece of code that RIPE uses for peering extraction in

Policy Type				Peering Type	# of Peerings	Single Registry	RIPE Only
$A \xrightarrow{E} B$	$A \xrightarrow{I} B$	$A \xleftarrow{E} B$	$A \xleftarrow{I} B$				
√	√	√	√	$A \overset{}{\text{———}} B$	42,599	96,7%	94.6%
	√	√	√	$A \overset{3/4\neg E}{\text{———}} B$	1,373	84.6%	80.3%
√		√	√	$A \overset{3/4\neg I}{\text{———}} B$	1,013	88.8%	82.2%
√				$A \overset{1/4E}{\text{———}} B$	34,155	100%	7.7%
	√			$A \overset{1/4I}{\text{———}} B$	13,997	100%	23.7%
√	√			$A \overset{1/2}{\text{———}} B$	114	90.4%	57.9%
√		√		$A \overset{1/2E}{\text{———}} B$	19	78.9%	47.4%
√			√	$A \overset{1/2AB}{\text{———}} B$	143,342	100%	58.4%
	√		√	$A \overset{1/2I}{\text{———}} B$	51	72.5%	66.7%
Total (including Self-Peerings)					236,663		
Self-Peerings					195		

Table 4.5: *Classification of the peerings extracted from the IRR.*

the RRCC service [234]. The peerings obtained in this way can be considered analogous to our peering candidates. By using the RIPE code we obtained 295,587 RRCC peering candidates, that are much less than our overall amount of 512,758 peering candidates (see Table 4.4). By aggregating the RRCC peering candidates with the method of Section 4.3.4 we obtained 108,521 RRCC peerings. Again, much less than our 236,663 peerings (see Table 4.5). Further, there are 102 RRCC peerings that we did not find. We discovered that 100 of them involve private ASes and the remaining 2 are due to an incorrect processing of the `and` operator by the RRCC code. A comparison with [9] gave similar results.

Comparing our results with the ones presented in [163, 12, 60] is not easy. In fact, they refer to the versions of the IRR of 04/07/2004, 10/24/2004, and 06/22/2003, respectively. To the best of our knowledge, no repository is available with IRR historical data. We have a repository of such data in the interval described in Figure 4.2 but, unfortunately, this interval does not cover the above dates. The authors of [163] provide the peerings extracted from the IRR on 04/07/2004. The work in [12] is supported by a web site providing several files of peerings. It is updated on a daily basis, yet the peerings discovered in the IRR are unavailable. Also the work in [60] has a web site [223] that allows to interactively explore the peerings detected on a specific date (at the time of writing, 11/08/2005). Again, this date is not covered by our archives.

Hence, only a rough comparison is possible. The topology of [163] reports 56,949 peerings while [12] reports the discovery of 70,222 peerings. Both refer to the RIPE Registry only. Paper [60] reports 127,498 peerings referred to the entire IRR. All these figures are very far from our results.

4.5 Conclusions

We think that the data contained into the IRR are a unique source of valuable information and therefore the results presented in this Chapter should be considered as a starting point and a necessary premise for future research on the topic. The availability of an effective peering extraction technique allows to retrieve topological information from the IRR with greater accuracy and opens, at least, the following perspectives.

Comparing the topological data extracted from the IRR with live routing data would give some hints about the overlapping of information between the two data sources and would help in assessing the health status of the IRR.

It would also be interesting to study how BGP announcements would spread over the Internet according to the policies registered in the IRR. This would give a better estimate of the consistency of IRR data against actual routing and would bring about the opportunity to perform specific actions on the IRR to improve its consistency.

On a longer term, one could even take advantage of routing policies documented in the IRR for emulating the entire (or a significant portion of) Internet. Virtual routers could be configured according to the policies described in RPSL objects. This could be of great help in understanding the behavior of the Internet and in forecasting, preventing, or debugging abnormal or unsafe routing scenarios.

Part III

Inference and Analysis of Routing Policies

Introduction

One of the aspects that has attracted most interest in the analysis of interdomain routing phenomena is the study of routing policies. The reason is that routing policies drive many of the behaviors that can be observed on the Internet. In the particular case of BGP, policies can be specified independently of routing optimization requirements, and this "freedom" of choice can lead to unexpected interactions among policies specified at different sites. Therefore, one of the reasons that foster investigations on routing policies is the need to prevent and, should they happen, to debug routing anomalies [193, 98, 68, 114, 75, 97, 101, 74, 191, 125, 124, 194].

However, there are also other reasons that make it interesting to understand the effects of routing policies. For example, policies are used to implement the commercial relationships [121] between different Autonomous Systems and, especially on the part of the ISPs, knowing such relationships would improve the awareness in setting up new commercial agreements. Also, routing policies are an effective means to achieve traffic engineering requirements: operators can deploy ad hoc configurations that influence the way traffic flows access or leave the Autonomous System they operate.

Unfortunately, routing policies are not publicly available heritage. Since they often implement ISP strategic choices, they are kept reserved in order not to leak internal sensible information. This brings about the need to develop algorithms that are capable of inferring them based on publicly available data. Several solutions have been proposed in the literature to address the problem of inferring the commercial relationships between Autonomous Systems [248, 187, 115, 122, 260, 121]. Chapter 5 presents a brief survey of these algorithms and describes a methodology for the comparative evaluation of their results, which helps in validating and determining the quality and reliability of the inferred relationships. The methodology has been implemented as a publicly available suite of software tools [237],

and is used to perform extensive analyses on the results produced by the inference algorithms.

Among the known interdomain traffic engineering strategies [21, 152, 19, 151, 132], one that has proved to be effective for influencing inbound traffic flows is based on the prepending technique [175]. Chapter 6 describes a theoretical framework which helps in determining the optimal amount of prepending to be used for pursuing different kinds of objectives, from cost sharing to well-balanced bandwidth allocation.

Chapter 7, which concludes this part, presents possible approaches to the study of the interaction of interdomain routing policies configured at different sites. In particular, it presents techniques to determine whether policies allow traffic to a certain prefix to traverse arbitrarily chosen AS-paths, and to establish how the BGP selection mechanism assigns a preference to two distinct, equally long, AS-paths. Issues related to bad interactions of routing policies leading to the occurrence of routing instabilities are also discussed in this Chapter. In particular, a model is proposed to capture different kinds of stable configurations.

Chapter 5

Inference of Commercial Relationships between Autonomous Systems

> But I was one-and-twenty,
> No use to talk to me.
>
> A Shropshire Lad, poem XIII
> Alfred Edward Housman

The study of interdomain routing policies definitely helps in better understanding routing dynamics. This does not only mean focusing on abnormal scenarios (such as loss of reachability, occurrence of routing oscillations, etc.) with the purpose of debugging them, but also gaining deeper insight in the mechanisms that regulate the large scale behavior of the Internet.

There are several people that would gain a benefit from knowing currently running routing policies. Researchers can take advantage of this knowledge to address and correct flaws of routing protocols [150]. Operators have an additional piece of information to debug unstable configurations [98, 193, 97, 191, 125, 194, 124, 190, 192]. Internet Service Providers can better choose their upstream providers [246, 58, 59, 160].

Unfortunately, routing policies are kept reserved by network operators because they are considered as integral part of the economic strategies of an ISP. For this reason, ad hoc inference algorithms have been developed to deduce them from publicly available routing information. This Chapter concentrates on algorithms for the inference of the commercial relationships between Autonomous Systems [248, 187, 115, 122, 260, 121]. The Chapter presents a brief survey of these algorithms

and describes a methodology to assess the quality and reliability of their results by comparative analysis. A publicly available suite of software tools [237] that implements the methodology is also described. The methodology and the tools are exploited to perform extensive analyses on the results produced by the inference algorithms.

5.1 Background

The administrative authorities controlling each Autonomous System need to subscribe contracts for obtaining connectivity to the rest of the Internet. These contracts are commonly (and a bit roughly) referred to as *commercial relationships*, and can be of different kinds [58, 59, 57].

The BGP protocol allows to impose limits on the spread of routing information by means of the configuration elements known as *policies*. In practice, commercial relationships are implemented using specific sets of policies. What follows is a description of the most common commercial relationships [122, 121], together with the policies that usually implement them:

Customer-provider: AS_c is said to be a customer of AS_p if AS_c pays AS_p for obtaining the connectivity to the rest of Internet. The policies that are used to export routing information are usually the following:

- AS_c exports to AS_p its own prefixes and the ones of its customers; it does not export prefixes coming from its peers or providers;
- AS_p exports to AS_c its own prefixes and those of its other customers, peers, and providers.

Peer-peer: Two ASes are said to be peers if they mutually agree to exchange traffic between their customers, quite often free of charge. The policies that are used to export routing information are usually the following:

- each of the two ASes exports to the other its own prefixes and the ones of its customers; it does not export prefixes coming from its peers or providers.

Understanding the commercial relationships between ASes is useful for several reasons. New Internet Service Providers (ISPs) can exploit such knowledge in order to infer the relevance of the other ASes in the Internet and hence choose better which ASes should be preferred for establishing commercial relationships. Network administrators can obtain useful hints from such

knowledge, because it can help them in avoiding configurations which induce BGP instabilities [124, 125, 191]. People studying the Internet evolution can exploit the knowledge of the commercial relationships to better understand the laws that control the growth of the network.

Explicitly asking the ISPs for the relationships they establish each other is practically impossible for several reasons: the number of ASes which we may need to query is incredibly large, such organizations are usually not willing to reveal information that are sensible for their core business, and it is hard even to collect the needed contact information. Hence, other procedures must be introduced, that do not involve the direct contact with the AS organization.

Several algorithms [122, 115, 64, 187] have been proposed in the literature to infer the commercial relationships between ASes, based on the observation "from the outside" of their routing behavior. This Section presents a brief survey of these algorithms.

Inference algorithms usually take as input a list of AS-paths and produce as output a relationship assignment. The list of AS-paths can be obtained from one or more *telnet looking glass* servers [188], which are routers whose BGP routing tables can be looked at from a remote location. Since routing tables contain the AS-paths used to reach various destinations, these paths can be merged into an *AS graph*. The vertices of this graph are the ASes, and the edges correspond to adjacencies in the paths. In turn, an adjacency (AS_1, AS_2) is evidence of a BGP session between a router of AS_1 and a router of AS_2.

5.2 Problem Statement

Given an AS graph G, an inference algorithm produces a *relationship assignment* on it, that is a labelling of each edge of G with the relationship occurring between its terminal nodes. We will also refer to the labelled graph as an *oriented* graph. The relationship assignment corresponds to a partial orientation of the AS graph G: it can be assumed that an undirected edge corresponds to a peering relationship, while a directed edge is oriented from customer to provider. Therefore, the terms *orientation* and *relationship assignment* will be used as synonyms in the following.

In [122] has been first observed that, if the commercial relationships are actually implemented using the policies described in Section 5.1, then all the AS-paths should have no *valleys*. Given an oriented AS graph G, an AS-path $p = AS_1, AS_2, \ldots, AS_n$ is *valley-free* (or *valid*) if either one of the following conditions holds:

- p is a sequence $AS_1, \ldots, AS_i, 1 \leq i \leq n$ of customer-provider edges, followed by a sequence $AS_i, \ldots, AS_n$ of provider-customer edges;
- p is a sequence $AS_1, \ldots, AS_i, 1 \leq i < n$ of customer-provider edges, followed by the peer-peer edge AS_i, AS_{i+1}, followed by a sequence $AS_{i+1}, \ldots, AS_n$ of provider-customer edges.

In other words, p is *valley-free* with respect to a given orientation if provider-customer edges are always followed by provider-customer edges and peer-peer edges are always followed by provider-customer edges.

All the algorithms which are described here are based on the fundamental assumption that realistic routing policies would lead to AS-paths all satisfying the valley-free property.

Figure 5.1 shows some examples of invalid paths. Consider, for example, path (a), which traverses the provider-customer edge (Pr_1, C) and then the customer-provider edge (C, Pr_2). Such a configuration implies that customer C exports prefixes coming from Pr_2 to Pr_1 or vice versa. This would result in customer C offering transit service between Pr_1 and Pr_2, which is unrealistic. A similar argument holds for paths (b), (c) and (d).

The valley-free property inspired the formulation of a combinatorial problem, which has been first introduced in [115] as the *Type of Relationship* (*ToR*) problem: given an AS graph G and a set of AS-paths P, find an orientation (relationship assignment) of some of the edges of G which minimizes the number of invalid paths in P.

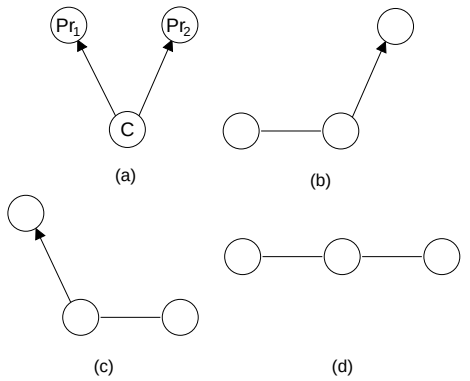


Figure 5.1: *Examples of invalid (non* valley-free*) paths.*

5.3 Inference Algorithms

It has been proved [64, 187] that ToR is NP-complete. Therefore, inference algorithms are either based on heuristics [122, 115] or on less constrained versions of the same problem [64, 187].

The first heuristic has been proposed by Lixin Gao [122]. This algorithm starts by computing the degree (number of adjacent ASes) of each AS in the AS graph and uses it to infer transit relationships. Customer-provider relationships are then assigned according to the inferred transit relationships. A refined version of this algorithm allows to also assign peer-peer relationships.

A second algorithm has been introduced by Agarwal et al. [115]. This algorithm considers routing data obtained from various looking glasses, which the authors call *vantage points*. For each vantage point, the algorithm associates a rank to each AS. Then, if two adjacent ASes are found to have a different rank, the one with lower rank is considered as a customer and the other one as a provider. Note that, given an edge e, each vantage point could assign a different relationship to e, from its specific point of view. The algorithm actually assigns to e the relationship which is proposed by the highest number of vantage points.

One of the most recent algorithms has been proposed by Di Battista et al. in [64]. This algorithm is based on a reduction of the ToR problem to the well known problem of satisfiability of propositional formulae (SAT). Essentially, the relationship assignment is computed by defining an instance of the SAT problem and solving it. Note that the algorithm looks for a solution of the ToR problem with no invalid paths, which reduces the computational complexity but at the same time deviates from the original formulation of the ToR problem. However, starting from the input set P, a maximal subset of valid paths can then be computed by using some heuristics. A similar approach has independently been proposed by Erlebach et al. in [187].

The most recent approach has been proposed by Dimitropoulos et al. [248, 249, 251], who reformulate the ToR problem as a multiobjective optimization problem. The goal is to extend the combinatorial optimization approach based on the minimization of invalid paths with information about the AS degree. A tunable parameter allows to steer the inference of the relationships to pursue one objective or the other. This is a way to conciliate Di Battista [64] and Erlebach's [187] approach with Gao's [122], and to overcome some of the shortcomings of the two.

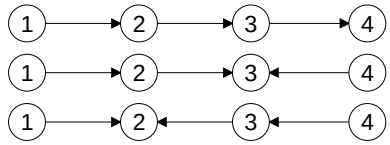


Figure 5.2: *Examples of orientations (relationship assignments) which make the AS-path* 1234 *valid.*

5.4 A Methodology to Evaluate the Quality of Inference Algorithms

An important issue that is left open by the algorithms presented in Section 5.3 is to understand whether their approach yields results that are of practical interest.

One possibility could be to evaluate the proposed techniques against their ability of enforcing the valley-free property on the AS graph, yet this may not be enough. For example, suppose the input to our inference algorithm is just one AS-path. Figure 5.2 shows that many solutions exist that make this path valley-free but, presumably, only one of them is correct.

Another approach could be to validate the results of the inference against relationships known from reality. In [122] the authors pursue this approach. However, such a validation process cannot scale to the entire Internet, and can therefore be only performed in the small.

Further investigation is therefore needed in order to validate the inference results. A first step towards this is to identify features that a reasonably good inference should have:

Stability: Internet is constantly evolving under the pressure of social and economic forces, but this evolution is slow and never affected by many changes on the short period. On the contrary, the technical aspects of the network superimpose to the above slow evolution the changes brought about by routing algorithms, mainly when technical failures happens. This evolution usually suffers of bursts of many events on the short period. The results of an inference algorithm should ideally be mostly affected by the first kind of changes.

Independence from the algorithm: Computing the types of the relationships requires using a specific inference algorithm. Thus, the obtained relationships might be bound to choices which are specific to the that

algorithm. This is obviously undesirable, since only one choice of the relationships can be considered correct. The ideal inference result should be algorithm independent, in the sense that it is the same (or, at least, very similar) for all "good" inference algorithms.

This constitutes a first possible set of features to test the validity of the different inference procedures known in the literature. The aim is to exploit them in order to determine whether considering the valley-free property alone, which most algorithms are based on, is sufficient to produce an acceptable assignment.

5.4.1 Measuring Differences between Inference Results

In accordance with the features defined in Section 5.4, which are supposed to characterize good quality algorithms for inferring the commercial relationships, the methodology is based on two kinds of analysis: a *stability analysis*, which compares the inference results obtained from a single algorithm run on inputs taken at different time instants, and an *algorithm independence analysis*, that compares the inference results obtained using different algorithms on the same input.

Given an AS graph $G = (V, E)$ obtained as described in Section 5.1, let $R : E \rightarrow V \cup \{peering, unknown\}$ be a function describing the relationship assigned by an inference algorithm to each edge $e = (AS_1, AS_2) \in E$, in the following way:

$$R(e) = \begin{cases} AS_2 & \text{if } AS_2 \text{ is a provider of } AS_1 \\ AS_1 & \text{if } AS_1 \text{ is a provider of } AS_2 \\ peering & \text{if } AS_1 \text{ and } AS_2 \text{ are peers} \\ unknown & \text{if no relationship is known.} \end{cases} \quad (5.1)$$

We now introduce a set of measures which can be used to compare two distinct relationship assignments $R_A(\cdot)$ and $R_B(\cdot)$, defined on the same graph $G = (V, E)$. In particular, the differences between the assignments are estimated in terms of the cardinality of the following sets of edges, each isolating a particular kind of difference:

$$Both = \{e \in E | R_A(e) \neq unknown \ \wedge R_B(e) \neq unknown\} \quad (5.2)$$

$$OnlyInA = \{e \in E | R_A(e) \neq unknown \ \wedge R_B(e) = unknown\} \quad (5.3)$$

$$OnlyInB = \{e \in E | R_A(e) = unknown \ \wedge R_B(e) \neq unknown\} \quad (5.4)$$

The set *Both* contains edges which have successfully been assigned a relationship by both the inferences. *OnlyInA* and *OnlyInB* contain edges for which the assignment has successfully been inferred in only one case.

The set *Both* is further partitioned into the following two subsets:

$$Consistent = \{e \in Both \mid R_A(e) = R_B(e)\} \quad (5.5)$$

$$Inconsistent = \{e \in Both \mid R_A(e) \neq R_B(e)\} \quad (5.6)$$

In turn, the set *Inconsistent* can be further partitioned in order to identify the kind of difference occurring between the two assignments:

$$Opposite = \{e \in Inconsistent \mid R_A(e) \neq peer \ \wedge \ R_B(e) \neq peer\} \quad (5.7)$$

$$PeerInA = \{e \in Inconsistent \mid R_A(e) = peer \ \wedge \ R_B(e) \neq peer\} \quad (5.8)$$

$$PeerInB = \{e \in Inconsistent \mid R_A(e) \neq peer \ \wedge \ R_B(e) = peer\} \quad (5.9)$$

The set *Opposite*, as the name itself suggests, contains edges that have been assigned the opposite relationship by $R_A(\cdot)$ and $R_B(\cdot)$ (i.e., they are labelled as customer-provider edges by one assignment and as provider-customer by the other).

In order to better understand how the relationship assignments $R_A(\cdot)$ and $R_B(\cdot)$ are compared, Figure 5.3 graphically depicts the sets of edges defined up to this point. Table 5.1 shows how these sets are populated according to the relationships described by $R_A(\cdot)$ and $R_B(\cdot)$: each edge $e = (AS_1, AS_2)$ is inserted in the sets listed in the cell corresponding to the values of $R_A(e)$ and $R_B(e)$.

5.4.2 Extensively Evaluating Inference Algorithms

The sets introduced in Section 5.4.1 have been defined to compare two different assignments constructed on the same graph. In this Section we define measures to characterize how the same assignment algorithm works on many graphs obtained by data snapshots distributed over time.

Consider a sequence of sets of AS-paths that are obtained by consecutively probing the network over time and suppose to run a single inference algorithm on each set. Such sequence of sets induces a sequence of AS graphs $G_1(V_1, E_1), \ldots, G_n(V_n, E_n)$.

Let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ be an AS graph such that: $\widetilde{E} = \bigcup_{i=1}^{n} E_i$, that is, $\widetilde{E}$ is the set of edges which appear in at least one of the G_i; $\widetilde{V}$ is naturally induced by $\widetilde{E}$ (note that no isolated nodes are omitted in $\widetilde{V}$ since the graphs $G_1, \ldots, G_n$

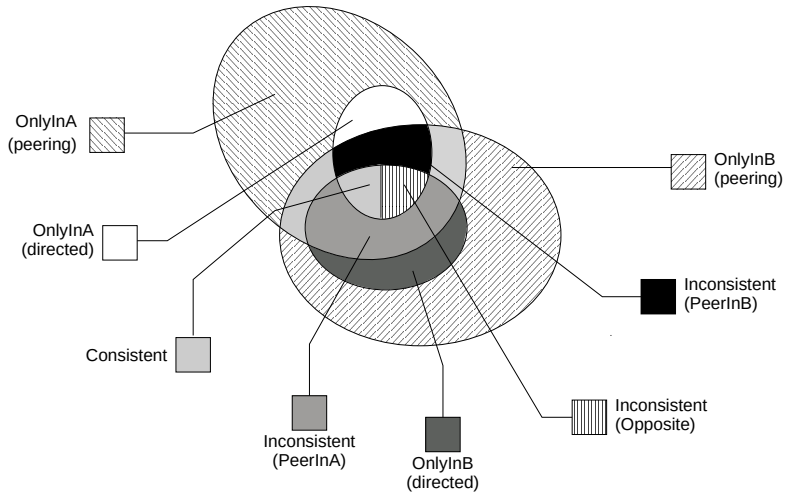


Figure 5.3: *The sets of edges we use to compare two relationship assignments $R_A(\cdot)$ and $R_B(\cdot)$. This diagram is to be intended in a universe which contains the edges of G. Outer ovals represent peering edges. Inner ovals contain directed (customer-provider or provider-customer) edges.*

$R_A((AS_1, AS_2))$ $R_B((AS_1, AS_2))$	AS_1	AS_2	*peering*	*unknown*
AS_1	*Both*, *Consistent*	*Both*, *Inconsistent*, *Opposite*	*Both*, *Inconsistent*, *PeerInA*	*OnlyInB*
AS_2	*Both*, *Inconsistent*, *Opposite*	*Both*, *Consistent*	*Both*, *Inconsistent*, *PeerInA*	*OnlyInB*
peering	*Both*, *Inconsistent*, *PeerInB*	*Both*, *Inconsistent*, *PeerInB*	*Both*, *Consistent*	*OnlyInB*
unknown	*OnlyInA*	*OnlyInA*	*OnlyInA*	—

Table 5.1: *The sets of edges we use to compare two relationship assignments $R_A(\cdot)$ and $R_B(\cdot)$. Each edge $e = (AS_1, AS_2)$ of the AS graph is inserted in the sets listed in the cell corresponding to the values of $R_A(e)$ and $R_B(e)$.*

are connected). The inference on the i-th data set results in a relationship assignment $R_i(\cdot)$ which is defined on E_i. However, to make the notation easier, in the following we assume $R_i(\cdot)$ to be defined on $\widetilde{E}$, having value *unknown*on edges in $\widetilde{E} \backslash E_i$.

Each edge $e \in E$ is labelled with values which summarize the history of the relationships assigned to it:

$$Occurrences(e) = |\{i \mid 1 \leq i \leq n \wedge e \in E_i\}| \quad (5.10)$$

$$Assignments(e) = |\{i \mid 1 \leq i \leq n \wedge R_i(e) \neq unknown\}| \quad (5.11)$$

$$Changes(e) = |\{i \mid i \leq i \leq n-1 \wedge R_i(e) \neq R_{i+1}(e)\}| \quad (5.12)$$

The value $Occurrences(e)$ corresponds to the number of graphs $G_i(i = 1, \ldots, n)$ edge e appears in. $Assignments(e)$ is the number of graphs in which edge e has been assigned a relationship. $Changes(e)$ is the number of times $R(e)$ changes its value in the sequence $R_1(e), \ldots, R_n(e)$.

5.5 A Software Suite to Evaluate Inference Algorithms

In order to apply the evaluation methodology and to compute the measures defined in Sections 5.4.1 and 5.4.2, we have developed a suite of software tools called TORQUE (Type Of Relationship QUality Evaluation) [237].

Computing the measures requires processing both the data sets provided as input to the inference algorithm and the corresponding inference results. The tools of the suite handle both kinds of information. The intended usage flow of the tools in TORQUE is shown in Figure 5.4, where data are shown as square boxes while tools are represented as rounded boxes. Arrows describe the flow of information.

Data sets come, usually, in the form of `show ip bgp` dumps, while inference algorithms work on files containing plain lists of AS-paths. For this reason we implemented an *AS-path extractor* tool. We payed special attention to implement various kinds of processing on the extracted AS-paths, like removal of prepending, AS-sets, cycling paths, duplicates, etc.

A *graph generator* tool merges the information of an AS-path list, a BGP dump, and a relationship assignment computed by an inference algorithm, producing a *labelled AS graph* where each edge is associated with the values of the function $R(\cdot)$ (see Section 5.4.1). The tool is designed to be flexible with respect to the input information used to generate the labelled graph.

The *graph comparison* tool allows to compare two or more labelled AS graphs. When exactly two labelled AS graphs are given as input, it computes

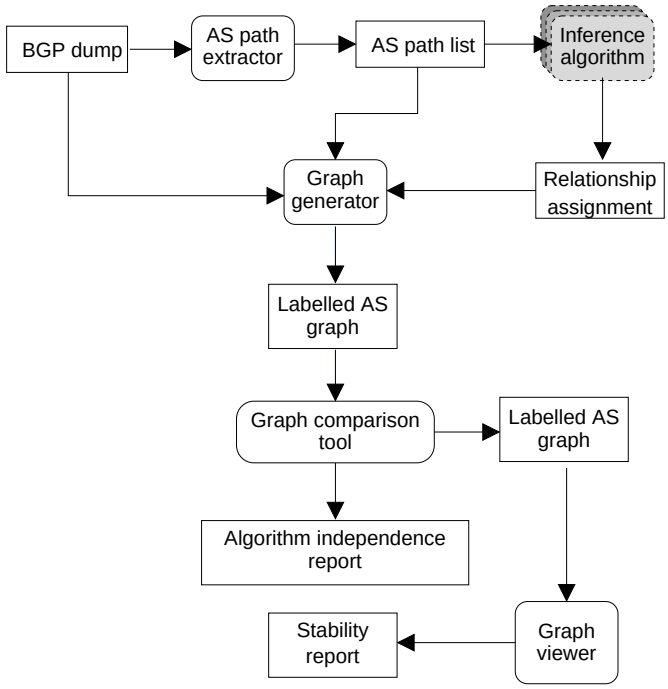


Figure 5.4: *Intended usage flow for the tools in the* TORQUE *software suite [237]. Square boxes represent data; rounded boxes represent tools; arrows describe information flows.*

and provides a report of the cardinalities of the sets *Both*, *OnlyInA*, *OnlyInB*, *Consistent*, *Inconsistent*, *Opposite*, *PeerInA*, and *PeerInB*. If more than two labelled AS graphs are given as input, it computes, for each edge, the following measures: *Occurrences*, *Assignments*, and *Changes*. In the latter case the output is another AS graph which edges are labelled with these measures.

Last, the *graph view* tool may be used to produce various kinds of reports (strongly connected components, differences in edges orientation, etc.).

Both the methodology and the tools have been conceived with one important objective in mind: to clear up the semantic of the whole processing, which we believe can be helpful in establishing a unified approach for this kind of data analysis.

5.6 Experimental Results

In this Section we present the results we obtained by applying the methodology and the tools to evaluate the inference algorithms by Di Battista et al. [64] (DPP) and by Subramanian et al. [115] (SARK). Due to the lack of an available working implementation (Gao's group has developed one [259], but it fails in processing our large data sets), this analysis leaves out Gao's [122] and Dimitropoulos' [248] algorithms. However, by using the software tools, it would be easy to extend the investigation to the missing algorithm once an implementation is available.

5.6.1 Data Sets

In the following, we use the term *snapshot* to denote both a list of AS-paths and the AS graph which it induces. The specific meaning depends on the context.

We consider snapshots provided by two different sources.

The first one is the site describing the SARK algorithm [116], which provides several snapshots, each collecting paths simultaneously taken from different BGP looking glasses. We consider the snapshots listed in Table 5.3. We address this data set using the name MVP.

The second source is the Oregon Route Views Archive [198], which provides BGP routing table dumps of a router having peering sessions with about 50 ASes at the time of the experiments. The dumps are collected every two hours. We consider the snapshots corresponding to the time intervals in Table 5.2. Figure 5.5 shows the size of the data sets RV1 and RV2. The set RV2 contains an incomplete snapshot (10/03/2001 14:00), which is due to some failure in the Route Views collection process.

Since we want to test the stability of the inference results, we chose RV1 to cover a period during which only a few ASes changed the commercial agreements, whereas RV2 has been selected to consider a time interval during which several commercial changes took place. We obtained information about com-

Snapshot name	Period		# of snapshots
	Start	End	
RV1	03/25/2003 00:00	03/31/2003 22:00	84
RV2	09/25/2001 00:00	10/08/2001 22:00	168

Table 5.2: *Snapshots taken from [198].*

Date	Used looking glasses (AS number)	AS graph		Unique AS-paths
		Nodes	Edges	
04/18/2001	1, 1740, 3549, 3582, 3967, 4197, 7018, 8220, 8709	10909	23817	511200
01/29/2002, 02/04/2002	1, 3549, 3582, 3967, 4197, 7018, 8220, 8709	12708	27555	722481
04/06/2002	1, 1838, 3549, 3582, 3967, 4197, 5511, 7018, 8220, 8709, 15290	13079	28309	942382
07/29/2002	1, 1838, 3257, 3549, 3582, 3967, 4197, 5511, 7018, 8220, 8709, 15290	13705	29073	948720
08/09/2002	1, 1838, 3257, 3549, 3582, 4197, 5511, 7018, 8220, 15290	13754	29009	894396
10/19/2002	1, 1838, 3582, 3967, 5511, 7018, 8220, 15290	14113	29422	881836
10/29/2003	1, 50, 210, 553, 852, 1838, 3257, 3549, 3582, 3741, 3967, 4197, 5388, 5511, 6395, 6539, 6893, 7018, 8220, 8709, 8843, 9328, 15290	16420	37470	1143373
11/13/2003	1, 50, 210, 553, 852, 1838, 3257, 3549, 3582, 3741, 3967, 4197, 5388, 5511, 6395, 6539, 6893, 7018, 8220, 8709, 8843, 9328, 15290	16461	37406	1157802
11/28/2003	1, 50, 210, 553, 852, 1838, 3257, 3549, 3582, 3741, 3967, 4197, 5388, 5511, 6395, 6539, 6893, 7018, 8220, 8709, 8843, 9328, 15290	16316	31419	247691
12/12/2003	1, 50, 210, 553, 852, 1838, 3257, 3549, 3582, 3741, 3967, 4197, 5388, 5511, 6395, 6539, 6893, 7018, 8220, 8709, 8843, 9328, 15290	16585	37790	1216534
12/29/2003	1, 50, 210, 553, 852, 1838, 3257, 3549, 3582, 3741, 3967, 4197, 5388, 5511, 6395, 6539, 6893, 7018, 8220, 8709, 8843, 9328, 15290	16728	38162	1164370
01/13/2004	1, 50, 210, 553, 852, 1838, 3257, 3549, 3582, 3741, 3967, 4197, 5388, 5511, 6395, 6539, 6893, 7018, 8220, 8709, 8843, 9328, 15290	16762	38205	1264677

Table 5.3: *Snapshots in the MVP data set from [116].*

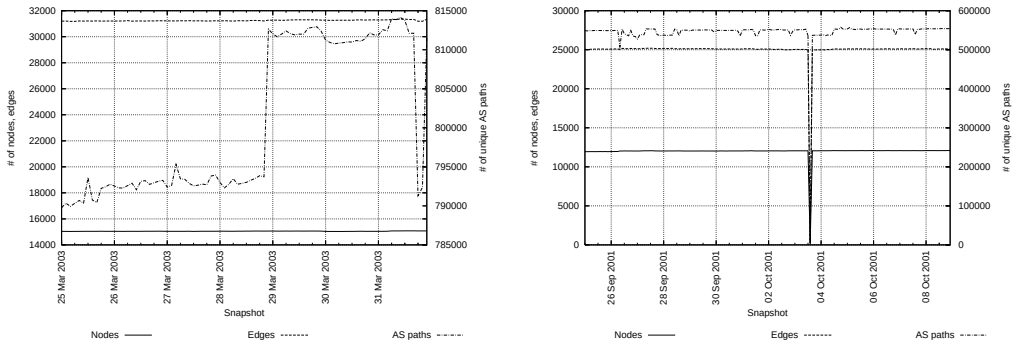


Figure 5.5: *Size of the snapshots in RV1 (left) and RV2 (right).*

mercial agreements from [215] and [218], and used them to identify the two intervals.

All the snapshots have been processed in order to extract AS-paths from BGP dumps. We always considered lists of unique AS-paths: i.e., for AS-paths occurring more than once in an input data set, only one instance is kept.

5.6.2 Independence of Inference Results from Routing Changes

Our first kind of analysis ascertains whether the results produced by inference algorithms are only affected by real commercial events and not by the (more frequent) underlying routing changes. We consider an assignment *stable* when the relationships remain almost the same over snapshot collected at different times.

The stability analysis considers all the data sets described above. In particular, we run the DPP algorithm [64] on all the sets MVP, RV1, and RV2, and we consider the results of the SARK [115] algorithm on the MVP data set. Stability is evaluated as follows.

For the MVP data we consider pairs of consecutive snapshots and compare the relationship assignments computed by the same algorithm (DPP or SARK) on such pairs (with the exception of the pair 10/19/2002, 10/29/2003). For each pair we compute the cardinalities of the sets defined in Section 5.4.1, with $R_A(\cdot)$ and $R_B(\cdot)$ being defined on the intersection of the AS graphs of the two snapshots. The results are shown in Tables 5.4 for the DPP algorithm and 5.5

	A: 04/18/2001 B: 01/29/2002, 02/04/2002		*A: 01/29/2002, 02/04/2002 B: 04/06/2002*		*A: 04/06/2002 B: 07/29/2002*		*A: 07/29/2002 B: 08/09/2002*	
OnlyInA	7		1		2		5	
OnlyInB	18		9		1		1	
Both	15919		24743		23638		28039	
Consistent	15118	(95%)	24112	(97%)	23044	(97%)	27719	(99%)
Inconsistent	801	(5%)	631	(3%)	594	(3%)	320	(1%)

	A: 08/09/2002 B: 10/19/2002		*A: 10/29/2003 B: 11/13/2003*		*A: 11/13/2003 B: 11/28/2003*		*A: 11/28/2003 B: 12/12/2003*	
OnlyInA	3		4		4		4	
OnlyInB	2		8		2		2	
Both	25458		35751		30654		31034	
Consistent	24916	(98%)	35211	(98%)	30120	(98%)	30509	(98%)
Inconsistent	542	(2%)	540	(2%)	534	(2%)	525	(2%)

	A: 12/12/2003 B: 12/29/2003		*A: 12/29/2003 B: 01/13/2004*	
OnlyInA	7		9	
OnlyInB	2		6	
Both	37038		36865	
Consistent	36653	(99%)	36268	(98%)
Inconsistent	385	(1%)	597	(2%)

Table 5.4: *Comparison of the results produced by the DPP algorithm [64] over pairs of consecutive snapshots (A, B) (data set MVP). Values represent the cardinalities of the sets in the leftmost column. Percentages are relative to the value of Both.*

for the SARK algorithm. The percent values for the rows *Consistent* and *Inconsistent* are relative to the value of *Both*; the percent values for the rows *Opposite*, $PeerInA$, and $PeerInB$ are relative to the value of *Inconsistent*. In Table 5.4 the cardinalities concerning peering relationships are omitted since the DPP algorithm does not assign them. All the comparisons show that at least 95% of the assigned relationships are consistent for consecutive data sets.

For the RV1 and RV2 data sets we consider the relationship assignments computed using the DPP algorithm and, for each data set, we evaluate the historical measures defined in Section 5.4.2. The results are shown in Figures 5.6 to 5.8. Table 5.6 reports the size of the graph $\widetilde{G}$.

Figure 5.6 shows a distribution that helps us in evaluating the goodness of

	A: 04/18/2001 B: 01/29/2002, 02/04/2002		*A: 01/29/2002, 02/04/2002 B: 04/06/2002*		*A: 04/06/2002 B: 07/29/2002*		*A: 07/29/2002 B: 08/09/2002*	
OnlyInA	86		109		96		80	
OnlyInB	77		111		87		94	
Both	15749		24472		23407		27789	
Consistent	15056	(96%)	23713	(97%)	22517	(96%)	27343	(98%)
Inconsistent	693	(5%)	759	(3%)	890	(3%)	446	(2%)
Opposite	55	(8%)	33	(4%)	55	(6%)	14	(3%)
PeerInA	350	(51%)	383	(50%)	358	(40%)	257	(57%)
PeerInB	288	(41%)	343	(46%)	477	(54%)	175	(40%)

	A: 08/09/2002 B: 10/19/2002		*A: 10/29/2003 B: 11/13/2003*		*A: 11/13/2003 B: 11/28/2003*		*A: 11/28/2003 B: 12/12/2003*	
OnlyInA	111		103		613		64	
OnlyInB	89		157		58		599	
Both	25224		34542		29099		29482	
Consistent	24425	(97%)	33812	(98%)	28748	(99%)	29146	(99%)
Inconsistent	799	(3%)	730	(2%)	351	(1%)	336	(1%)
Opposite	44	(6%)	21	(3%)	8	(2%)	2	(1%)
PeerInA	367	(46%)	376	(52%)	237	(68%)	103	(31%)
PeerInB	388	(48%)	333	(45%)	106	(30%)	231	(68%)

	A: 12/12/2003 B: 12/29/2003		*A: 12/29/2003 B: 01/13/2004*	
OnlyInA	157		111	
OnlyInB	168		177	
Both	35739		35642	
Consistent	35100	(98%)	35172	(99%)
Inconsistent	639	(2%)	470	(1%)
Opposite	10	(2%)	14	(3%)
PeerInA	346	(54%)	233	(50%)
PeerInB	283	(44%)	223	(47%)

Table 5.5: *Comparison of the results produced by the SARK algorithm [115] over pairs of consecutive snapshots (A, B) (data set MVP). Values represent the cardinalities of the sets in the leftmost column. Percentages in rows Consistent, Inconsistent are relative to the value of Both; other percentages are relative to the value of Inconsistent.*

the inference as far as edge coverage is concerned. An edge $e \in \widetilde{E}$ may appear in a number $Occurrences(e) \leq n$ of snapshots and be assigned a relationship in $Assignments(e) \leq Occurrences(e)$ of them. The Figure shows the number of edges having a certain fraction of successfully assigned relationships. There are 32446 edges for RV1 and 27235 for RV2 to which the inference assigned a relationship in all the snapshots. That is, for at least 99% of the edges of $\widetilde{G}$ the assignment always succeeds.

In order to further assess the quality of the assignment, Figure 5.7 shows the evolution over time of the fraction of the edges of each snapshot which have an assigned relationship. The values are fairly constant around 99% for every snapshot. The spike of value 100% on 10/03/2001 at 14:00 for RV2 is due to the presence of an incomplete BGP dump in the Oregon Route Views Archive (see Figure 5.5).

Figure 5.8 shows a distribution that allows to understand how stable the relationship assignment is. Each edge e of $\widetilde{G}$ may change its assignment a number $Changes(e)$ of times over the observation period. The Figure shows the distribution of edges showing a certain number of $Changes(\cdot)$. There are 31766 edges for which the assignment never changes in RV1 and 25968 in RV2. This means that more than 94% of the edges of $\widetilde{G}$ never change the assignment.

Interestingly, Figure 5.8 puts in evidence a scale free behavior [6, 5, 4] of the distribution of the values of $Changes(e)$. As far as we know, this is the first time that a scale free distribution is observed in this type of phenomenon.

5.6.3 Independence of Inference Results from the Algorithm

We now use our methodology to evaluate to what extent the inferred relationships are independent from the specific algorithm used to compute them.

For each of the snapshots listed in Table 5.3, we compare the relationships inferred by the DPP and SARK algorithms, and use the cardinalities of the sets defined in Section 5.4.1 to estimate their level of similarity. The compared assignments, $R_A(\cdot)$ and $R_B(\cdot)$, are both defined on the same graph G.

	$\lvert\widetilde{V}\rvert$	$\lvert\widetilde{E}\rvert$
RV1	15150	32534
RV2	12317	27490

Table 5.6: *Size of the graph $\widetilde{G}$.*

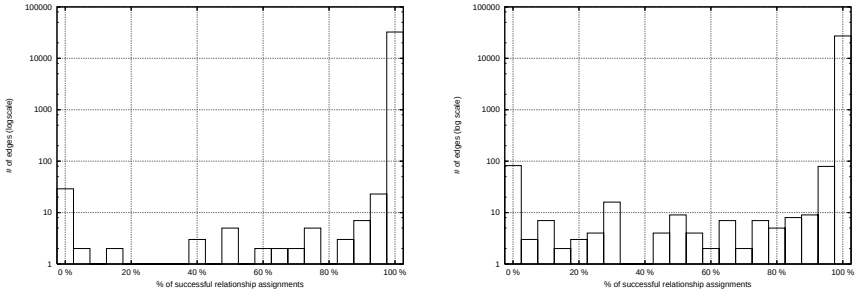


Figure 5.6: *Distribution of the fraction of the edges of $\widetilde{G}$ with an assigned relationship for the data sets RV1 (left) and RV2 (right).*

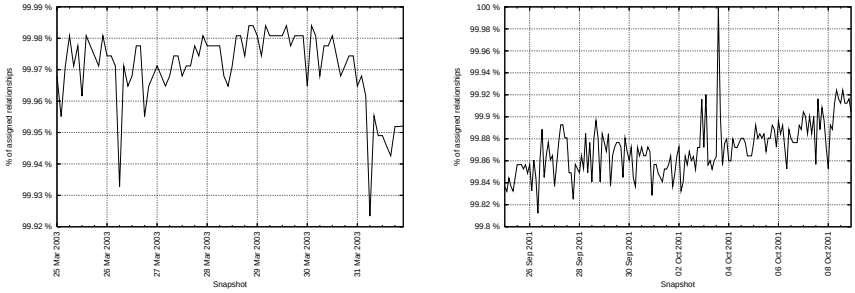


Figure 5.7: *Evolution over time of the fraction of the edges of $\widetilde{G}$ that have been assigned a relationship for the data sets RV1 (left) and RV2 (right).*

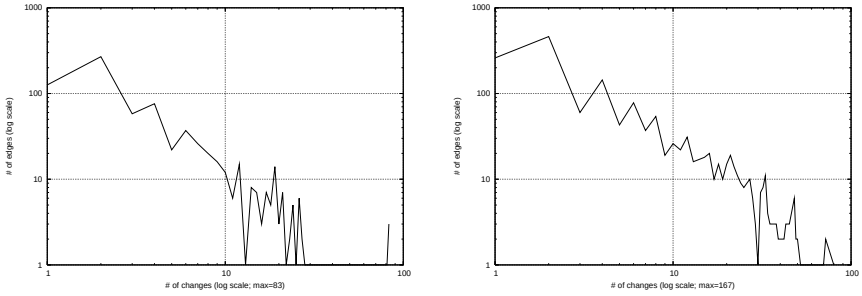


Figure 5.8: *Distribution of the number of assignment changes for the edges of $\widetilde{G}$ for the data sets RV1 (left) and RV2 (right).*

The results are shown in Table 5.7. The row *PeerInA* has been skipped because DPP does not infer peering relationships. The edges in *Consistent* are always more than 90% of those in *Both*, which means that the two algorithms almost produce the same solution. Inconsistencies are equally shared between *Opposite* and *PeerInB*.

5.7 Conclusions

In this Chapter we analyze the results produced by state of the art algorithms for the inference of the commercial relationships between Autonomous Systems. We perform two kinds of analysis: the first evaluates the degree of independence of the inferred relationships from routing changes that have nothing to do with the alteration of commercial agreements. The second kind of analysis is aimed at determining whether the results are independent from the specific algorithm being used. We describe a methodology and a software toolkit [237] to perform the analyses.

The methodology is used to evaluate the results produced by two well known inference algorithms [64, 115] over publicly available data from various looking glasses [116] and from the Oregon Route Views archive [198].

We find that the two algorithms produce highly stable results. In particular, using data from looking glasses [116], the percentage of AS pairs that have the same assigned relationship when using the same algorithm on two time adjacent snapshots is above 95% for both the algorithms. Using data from Route Views [198] and considering the algorithm by Di Battista et al. [64], this percentage slightly reduces to 94%.

The results also show that the two inference algorithms almost produce the same assignments: among the AS pairs to which both the algorithms assigned a relationship, more than 90% have the same assignment.

These conclusions led us to think that the *valley-free* approach, on which inference algorithms are based, leads to results that are of practical interest.

Dealing with the problem of evaluating the quality of the results produced by inference algorithms has also opened other interesting perspectives which we consider relevant and worth being further analyzed:

- It would be interesting to extend the presented analysis to take into account the results obtained by the algorithm by Gao [122] and Dimitropoulos et al. [248, 251, 249].

	04/18/2001	*01/29/2002, 02/04/2002*	*04/06/2002*	*07/29/2002*
OnlyInA	192	213	197	200
OnlyInB	39	21	6	6
Both	23584	27317	28106	28866
Consistent	21487 (91%)	24991 (91%)	25631 (91%)	26261 (91%)
Inconsistent	2097 (9%)	2326 (9%)	2475 (9%)	2605 (9%)
Opposite	993 (47%)	1047 (45%)	1217 (49%)	1176 (45%)
PeerInB	1104 (53%)	1279 (55%)	1258 (51%)	1429 (55%)

	08/09/2002	*10/19/2002*	*10/29/2003*	*11/13/2003*
OnlyInA	179	196	1205	1173
OnlyInB	9	14	28	25
Both	28819	29212	36235	36208
Consistent	26348 (91%)	26659 (91%)	32683 (90%)	32651 (90%)
Inconsistent	2471 (9%)	2553 (9%)	3552 (10%)	3557 (10%)
Opposite	1165 (47%)	1247 (49%)	1689 (48%)	1738 (49%)
PeerInB	1306 (53%)	1306 (51%)	1863 (52%)	1819 (51%)

	11/28/2003	*12/12/2003*	*12/29/2003*	*01/13/2004*
OnlyInA	1548	1367	1194	1118
OnlyInB	5	12	37	22
Both	29866	36409	36930	37063
Consistent	28332 (95%)	32754 (90%)	33321 (90%)	33557 (91%)
Inconsistent	1534 (5%)	3655 (10%)	3609 (10%)	3506 (9%)
Opposite	829 (54%)	1783 (49%)	1795 (50%)	1711 (49%)
PeerInB	705 (46%)	1872 (51%)	1814 (50%)	1795 (51%)

Table 5.7: *Comparison of the relationship assignments $R_A(\cdot)$ computed by the DPP [64] algorithm and $R_B(\cdot)$ computed by the SARK [115] algorithm on the data set MVP. Values represent the cardinalities of the sets in the leftmost column. Percentages in the Consistent and Inconsistent rows refer to the value of Both, while the others refer to the value of Inconsistent.*

- The space of the solutions of the relationship assignment problem is still missing a rigorous characterization. Up to now, research has focused on the identification of just one relationship assignment, yet it would be valuable to have a complete view of the degrees of freedom of the problem.
- Other papers tackled the problem of studying the Internet as a computer system by a fusion of algorithmic ideas and principles borrowed from game theory [29]. It would be interesting to examine possible contact points between game-theoretic approaches and the algorithms for inferring commercial relationships.
- Once a relationship assignment has been computed, it would be interesting to obtain a classification of the Autonomous Systems into hierarchical levels. Techniques for doing this have already been proposed by Ge et al. [260], by Subramanian et al. [115], and by Dimitropoulos et al. [251, 248, 205]. However, it would be interesting to know which is the best and most realistic way of obtaining hierarchy.

Chapter 6

Policy-Based Interdomain Traffic Engineering

She will bring, in spite of frost,
Beauties that the earth hath lost;
She will bring thee, all together,
All delights of summer weather.

Fancy
John Keats

ROUTING policies are an effective means to force traffic flows to obey constraints that have nothing to do with the optimization of routing performance and operation. For example, they can be used to make upstream links robust to failures by deploying configurations that automatically switch to alternative backup connections [125]. Appropriate routing policies are also used to ensure the correct fulfillment of commercial agreements between Internet Service Providers [122, 115, 64, 248], as also discussed in Chapter 5.

There is also another point in using routing policies: traffic engineering requirements can in some cases be achieved by taking advantage of standard BGP configuration atoms [21, 152, 19, 151, 132]. For example, due to performance and cost issues, optimizing the distribution of network traffic among upstream links is one of the main concerns for an Internet Service Provider. This Chapter presents an approach to pursue optimal inbound traffic distribution by exploiting a technique to artificially extend the length of AS-paths that is known as prepending. Even though a similar approach has already been proposed [175], this Chapter

introduces a reformulation in terms of an Integer Linear Programming and a Computational Geometry problem. The goal is to propose a theoretical framework to determine the optimal amount of prepending to be used for achieving different optimization objectives, and therefore to eliminate the need to resort to trial-and-error approaches.

6.1 Background

An Internet Service Provider (ISP) interacts with the rest of the Internet using the Border Gateway Protocol (BGP). For each of its IP prefixes P, it announces P to its neighboring ISPs. In turn, these neighbors pass the announcement of P to their neighbors, etc. At each step, every ISP receiving an instance of the announcement checks it against its policies, compares it against other instances of the same announcement received from other neighbors, selects the best according to the BGP metrics, and sends to its neighbors only the selected one.

In this propagation process the ISP originating P progressively loses control of what happens. The "control" is quite strong in the interaction with the immediate upstreams. It is regulated by a contract and enforced with several BGP features, like communities, prepending, etc. However, starting from the second-third step what happens to the announcement of P is, up to a large extent, uncontrolled by the original ISP. Since the traffic takes the opposite direction with respect to the one of the announcements, not controlling the propagation of the announcements means not being able to control incoming traffic flows.

Up to now, this topic attracted limited research interest for several reasons. First, up to a few years ago, ISPs at the lower level of the Internet hierarchy often had one upstream only, and in this case it is hard to influence the propagation of the announcements. Even for multi-homed ISPs, stable routing was often considered much more important than optimal traffic flow distribution. Last, the knowledge of the rest of the Internet by an ISP was somewhat poor.

These obstacles are now less relevant than in the past. Most ISPs are now multi-homed. Stability is always a problem but also competition is, and it is crucial to offer better services at a lower cost. Many resources are available to explore the structure of the Internet [173, 198, 33, 32, 39, 34].

Even a limited control on the propagation of the announcements could be used for:

- balancing the incoming traffic from the upstream providers, to improve performance or to shape the traffic according to the cost of the links;
- forcing a large portion of the incoming traffic to use a specific transit Autonomous System (AS) that is known to be reliable and/or with high bandwidth availability;
- improving the distribution of the internal traffic flows of an ISP.

The length of the AS-path is one of the main factors used to select the best path to reach the ISP's prefix P. Therefore, one possibility for the ISP to affect incoming traffic flows is to apply to the announcements of P an appropriate amount of AS-path prepending.

This Chapter addresses the problem of determining the optimal amount of prepending to be used in the announcements made by the ISP in order to achieve different traffic engineering requirements. The proposed approach relies on the formulation of an Integer Linear Programming problem or, as an alternative, a Computational Geometry one. We also show how efficient algorithms for computing the optimal amount of prepending can be devised based on these two formulations.

6.2 Related Work

The following brief survey on the state of the art in the field of traffic engineering puts in evidence that only a few works address the optimal prepending problem directly.

An introduction to the basic principles of traffic engineering is made by Awduche et al. in [43]. This work mainly focuses on intradomain traffic engineering, but it also presents some considerations about interdomain routing.

Feamster et al. propose in [151] some objectives and guidelines for interdomain traffic engineering using BGP. They show how data from BGP tables and from NetFlow [225] archives can be used to predict traffic flow changes, to limit the influence of neighboring ASes on the routing choices, and to reduce the overhead of routing changes.

Some BGP based techniques for traffic engineering are presented in [19] by Quoitin et al. This work describes how to control both the incoming and the outgoing traffic of an ISP, but does not present an experimental study.

Another description of BGP based techniques for traffic engineering is made by Swinnen et al. in [132]. This work also contains an experimental study of the

impact of AS-path prepending on incoming traffic volumes. In their study, they use the Javasim [219] event-driven simulator for running a BGP model over a topology built with the topology generator BRITE [13]. They show that the distribution of interdomain paths is actually affected by AS-path prepending.

Chang and Lo [175] approach the problem of finding the optimal prepending by using two kinds of measurements on the network. They collect NetFlow [225] data (passive measurements) and probe the network with *ping* packets in order to discover the upstream ISPs' routing policies with respect to the AS-path length (active measurements). Both kinds of measurements are used to predict the impact of prepending variations on network routing. They also test their methodology on a dual-homed AS. This approach, although effective, does not efficiently scale with the number of upstreams. The same paper shows that prepending based techniques favorably compare with alternative methods.

Other contributions focus on the impact of routing policies on the length of the AS-paths [73, 123].

It is also worth mentioning that several "route control" tools are available to automatically perform traffic engineering [229, 211, 202]. These solutions consist of devices and/or software tools that analyze live data flows and, if needed, adjust the network configuration according to user defined policies (application priorities, expected network behavior, performance, etc.). Some route control tools also claim to perform incoming traffic optimization by tuning BGP announcements of local prefixes [229]. In our opinion, such tools would benefit from theoretically sound formulations of incoming traffic engineering by prepending.

6.3 Traffic Engineering by Using Prepending

In this Section we define a model to describe the prepending choices of an ISP and their effects on Internet routing. This model is then used as a starting point to compute the optimal amount of prepending to be used to achieve different traffic engineering requirements.

Let A be the set of all the ASes in the Internet. Consider a specific AS t, called *target*, announcing a prefix P to its peers. We call *upstreams* these peers, denote them by U ($U \subset A$), and number them in $1, \ldots, |U|$. Each AS $a \in A - \{t\}$ receives one or more announcements about P and chooses one based on the length of the AS-path associated with them. We assume that policies allow announcements to traverse the network without constraints. Further, we assume that ASes choose the best announcement concerning P on the basis of

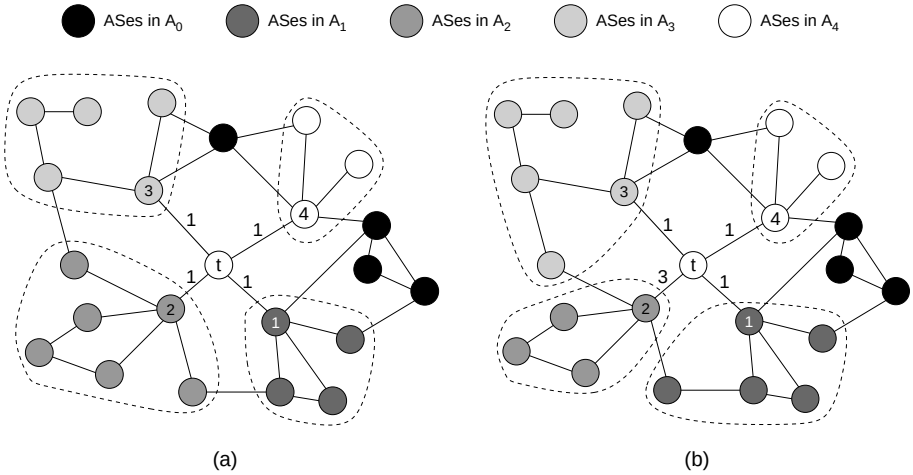


Figure 6.1: *(a) The network model used for studying the effect of AS-path prepending. t is the* target *AS originating prefix P. ASes $1, \ldots, 4$ are its upstreams. (b) The effect of prepending applied by t. Edge labels represent prepending amounts. The figure shows the sets $A_0, \ldots, A_4$ after applying prepending.*

the sole AS-path length. We refer to a simple model where attributes such as Local-Preference, MED, etc. are not used.

We define a partition of the ASes of $A - \{t\}$ into sets $A_0, A_1, \ldots, A_{|U|}$, where A_i is the set of the ASes that reach P through upstream i. Set A_0 is used to denote the ASes that have two (or more) shortest paths to reach P. More formally, if a has exactly one shortest AS-path to reach P, we say that a belongs to A_i, where $i \in U$ is the last AS occurring in the AS-path before t. If a has two or more shortest paths to reach P, each using a different upstream link of t, we say that a belongs to A_0. Essentially, A_0 contains the ASes whose choice cannot be predicted by simply looking at the length of the AS-path.

Figure 6.1(a) shows a simple network illustrating the model. Nodes represent ASes and edges represent BGP peerings. The figure describes the partition of A into the sets $A_0, \ldots, A_4$.

By using prepending, AS t can try to affect the way in which the other ASes reach P. AS t uses a prepending $w_i \geq 1$ when announcing P to AS $i, i \in U$ if it inserts its identifier in the AS-path w_i times.

Consider again the network of Figure 6.1(a). Suppose that t wants to

decrease the number of ASes that reach P through AS 2 (i.e., it wants to decrease the size of A_2). It can apply prepending $w_1 = 1$, $w_2 = 3$, $w_3 = 1$, and $w_4 = 1$. The resulting sets A_i are shown in Figure 6.1(b).

An administrator of t could use prepending for different purposes. This list shows some examples, which are used as possible objectives for the algorithms presented in this Chapter.

- Equal-Cardinality. The first possibility is to balance as much as possible the cardinalities of the sets A_i. This, to a first approximation, corresponds to balancing traffic to t coming from its upstream links.

- Equal-Load. Another possibility is to take into account the amount of traffic sent by each AS to t, so that the load on each upstream link is balanced.

- Shape-Bandwidth. Bandwidth requirements can be considered as well. The aim then becomes to compute a prepending assignment such that the load on the upstream links conforms to bandwidth availability.

- Equal-Cost. It is also possible to introduce a cost model for the upstream links, which takes into account their usage. This leads to search for the prepending assignment that ensures the best cost sharing.

- Equal-Cost-Threshold. The cost model can be refined to introduce a fixed base cost and a threshold, so that additional charges are only applied when the threshold is exceeded. ISPs often apply this kind of charging in their contracts.

Observe that a naïve approach to find the optimal prepending could consist of trying every possible combination of prepending amounts and choosing the combination that minimizes a specific objective function. This could require a great number of attempts, and for each attempt a considerable amount of time. Actually, after a fault, the network is known to converge within few minutes [41, 40]. However, many consecutive routing updates can easily trigger route flap damping [257, 24]. Experimental settings which need to repeatedly send updates at fixed rates adopt time intervals of 2 hours [258]. Furthermore, ISPs may deprecate a great amount of configuration changes on the part of their peers. This is why we consider approaches that can achieve optimality with a limited number of attempts.

6.3.1 Computing Prepending by Integer Linear Programming

We now propose an Integer Linear Programming (ILP) formulation of the problem of finding the best combination of prepending amounts to be used in the announcements of P. The ILP formulation consists of the following ingredients:

- Constants d_{ai} are an input of the problem, and the ISP t is supposed to know them (we shall see in Section 6.5 which is the practical impact of this assumption). They represent the length of the shortest AS-path from a to t's upstream i when P is announced to i only.
- w_j represents the amount of prepending through upstream j.
- Variables c_{ai} are used to model how the ASes choose to reach prefix P. Namely, c_{ai} is 1 if AS a uses upstream link i to reach P; 0 otherwise.
- We consider the following generic objective function, that will be refined later on.

$$\min f(c_{ai}) \tag{6.1}$$

There are two main types of constraints: Choice-Constraints and Tie-Constraints.

Choice-Constraints (Equation 6.2) model the choice of an AS that has only one shortest AS-path to reach P. In particular, if AS a chooses upstream link i ($c_{ai} = 1$), then the corresponding AS-path is the shortest one and there is no other path with the same length involving a different upstream link:

$$\forall a \in A - \{t\}, i \in U : c_{ai} = 1 \Rightarrow \forall j \in U - \{i\} : w_j + d_{aj} > w_i + d_{ai}$$

In order to write this implication in the form of a set of linear constraints, we introduce a constant M. We choose M large enough to ensure that constraints 6.2 are satisfied whenever $c_{ai} = 0$.

$$\forall a \in A - \{t\}, \forall i, j \in U, i \neq j : w_j + d_{aj} > w_i + d_{ai} + (c_{ai} - 1)M \tag{6.2}$$

Tie-Constraints (Equations 6.3, 6.4, and 6.5) model the case in which AS a knows at least two shortest paths to reach P through two different upstreams i and j. When this happens, e_{aij} is 1. Otherwise, e_{aij} is 0:

Obviously, the two paths through i and j have the same length:

$$\forall a \in A - \{t\}, \forall i, j \in U, i \neq j : e_{aij} = 1 \Rightarrow w_i + d_{ai} = w_j + d_{aj}$$

And their length is that of a shortest path:

$$\forall a \in A - \{t\}, \forall i, j \in U, i \neq j : e_{aij} = 1 \Rightarrow \forall k \in U : w_i + d_{ai} \leq w_k + d_{ak}$$

Again, we use a constant M that is large enough to satisfy constraints 6.3, 6.4, and 6.5 when $e_{aij} = 0$.

$$\begin{aligned} \forall a \in A - \{t\}, \forall i, j \in U, i \neq j : & \\ w_i + d_{ai} &\geq w_j + d_{aj} + (e_{aij} - 1)\, M \qquad (6.3) \\ w_j + d_{aj} &\geq w_i + d_{ai} + (e_{aij} - 1)\, M \qquad (6.4) \end{aligned}$$

$$\begin{aligned} \forall a \in A - \{t\}, \forall i, j, k \in U, i \neq j : & \\ w_i + d_{ai} + (e_{aij} - 1)\, M &\leq w_k + d_{ak} \qquad (6.5) \end{aligned}$$

Constraint 6.6 is introduced to ensure that each AS a either belongs to one A_i or belongs to A_0. Note that, if $a \in A_0$, there can be more than one e_{aij} that is set to 1. Observe that constraints 6.2, 6.3, and 6.4 prevent the two situations from happening simultaneously.

$$\forall a \in A - \{t\} : \sum_{i \in U} c_{ai} + \sum_{i,j \in U, i \neq j} e_{aij} > 0 \qquad (6.6)$$

Constraints 6.7 to 6.10 are used to define the domains of the variables.

$$\begin{aligned} \forall a \in A - \{t\}, i \in U &: c_{ai} \in \{0, 1\} \qquad (6.7) \\ \forall a \in A - \{t\}, \forall i, j \in U, i \neq j &: e_{aij} \in \{0, 1\} \qquad (6.8) \\ \forall i \in U &: w_i \in \mathbb{N}, \qquad (6.9) \\ & w_i > 0 \qquad (6.10) \end{aligned}$$

Let $n = |A|$ and $m = |U|$. The size of the problem is dominated by constraints 6.5, which give raise to $(n-1)m^2(m-1)$ inequalities.

Objective function 6.1 can be used to implement several requirements, as follows. Standard operations research techniques can be used to plug all the following objective functions in the ILP formulation.

- Equal-Cardinality.

$$f(c_{ai}) = \max_{i,j \in U} \left(\sum_{a \in A - \{t\}} c_{ai} - \sum_{a \in A - \{t\}} c_{aj} \right) \qquad (6.11)$$

- EQUAL-LOAD. Let l_a, $a \in A - \{t\}$ be the amount of traffic that AS a sends to P.

$$f(c_{ai}) = \max_{i,j \in U} \left(\sum_{a \in A - \{t\}} c_{ai} l_a - \sum_{a \in A - \{t\}} c_{aj} l_a \right) \tag{6.12}$$

- SHAPE-BANDWIDTH. Let $b_i, i \in U$ be the available bandwidth on upstream link i.

$$f(c_{ai}) = \max_{i \in U} \left| \sum_{a \in A - \{t\}} c_{ai} l_a - b_i \right| \tag{6.13}$$

  Minimizing this function corresponds to limiting traffic bursts and, at the same time, maximizing link usage.

- EQUAL-COST. Let cost_i be the cost that AS t has to pay for the upstream link i.

$$f(c_{ai}) = \max_{i,j \in U} (\text{cost}_i - \text{cost}_j) \tag{6.14}$$

  Function 6.14 can be used in conjunction with different cost models (i.e., different definitions of cost_i). For example, let unit_i be the cost for the unit of traffic that flows through upstream link i. The following constraint defines a simple linear cost model:

$$\forall i \in U : \text{cost}_i = \text{unit}_i \sum_{a \in A - \{t\}} c_{ai} l_a \tag{6.15}$$

- EQUAL-COST-THRESHOLD. A more realistic cost model that can be used with objective function 6.14 takes into account the fixed costs of link maintenance. Upstream links may have a fixed base cost base_i and traffic exceeding a threshold thresh_i may be charged unit_i per unit:

$$\forall i \in U : \text{cost}_i = \max \left\{ \sum_{a \in A - \{t\}} c_{ai} l_a - \text{thresh}_i, 0 \right\} \text{unit}_i + \text{base}_i$$

How to Break Ties

As already discussed, given a prepending assignment, an AS a can have two or more shortest AS-paths to reach P, each using a different upstream link of

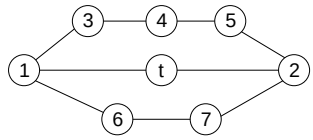


Figure 6.2: *A network for which ties can only be avoided with trivial prepending assignments (e.g., $w_1 = 5$ and $w_2 = 1$ or $w_1 = 1$ and $w_2 = 5$).*

t. In this case we say that there is a *tie* at AS a: the AS-path chosen by a to reach P cannot be predicted on the basis of the sole AS-path length. Since ties correspond to unpredictable choices, we would like not to introduce them at all.

Observe that a prepending assignment that does not give rise to ties always exists (just set $w_1 = 1$ and $w_i = n, i = 2, \ldots, m$), but it can be easily shown that such an assignment can be arbitrarily bad.

As an example, consider the network in Figure 6.2 and suppose to rule out those prepending assignments that introduce ties. In this network, there are two AS-paths between AS 1 and AS 2: one of odd length (1 3 4 5 2) and one of even length (1 6 7 2). Every prepending assignment such that $|w_1 - w_2| = 2k$, $k = 0, 1$ introduces a tie at one of the ASes in the path 1 3 4 5 2. On the other hand, every prepending assignment such that $|w_1 - w_2| = 2k + 1$, $k = 0, 1$ introduces a tie at one of the ASes in the path 1 6 7 2. Therefore, ties are only avoided by using prepending amounts such that $|w_1 - w_2| \geq 4$. In this way, all the ASes would fall into either A_1 or A_2 and, for example, the objective function Equal-Cardinality would assume high (i.e., bad) values, which is undesirable. Consider that a configuration similar to the one in Figure 6.2 is likely to appear in real world instances.

In conclusion, looking for solutions that do not introduce ties can lead to very poor prepending assignments.

This is the reason why objective functions do not consider ASes in A_0. However, once an optimal prepending assignment has been found by using the ILP, the number of ASes in A_0 provides an estimate of the quality of the assignment itself: assignments with a low number of ties should be preferred. All the ASes in A_0 may then be arbitrarily assigned to one of the $A_1, \ldots, A_m$ (i.e., the corresponding c_{ai} can be set to 1). By doing so, it is possible to explore the range of values that objective functions can assume, thus deriving bounds on the quality of the assignment.

Accommodating Multiple Prefixes in the Formulation

The model we have introduced assumes that AS t announces a single prefix P. However, an AS typically announces many prefixes to its neighbors. Both the model and the ILP formulation can be modified to consider multiple prefixes.

Since different policies can be applied for each prefix, the input values d_{ai} are depend on the specific prefix being considered. In particular, suppose that t announces p prefixes $P_1, P_2, \ldots, P_p$. We can introduce vectors $\mathbf{d}_{ai}$, such that the k-th component ($\mathbf{d}_{ai}[k]$) of vector $\mathbf{d}_{ai}$ is the length of the shortest path from a to i when t announces P_k to upstream i only. Similarly, variables w_i, c_{ai}, e_{aij} can be replaced by vectors, and the sets A_i can be organized in a vector as well. When using objective functions EQUAL-LOAD, the quantities l_a become vectors too.

Constraints 6.2 to 6.10 must be written for each prefix P_k, and they must use the values $\mathbf{w}_i[k]$, $\mathbf{d}_{ai}[k]$, $\mathbf{c}_{ai}[k]$, and $\mathbf{e}_{aij}[k]$.

Objective functions should be modified to consider all the prefixes. For example, function EQUAL-CARDINALITY (6.11) can be replaced with the following:

$$f(\mathbf{c}_{ai}) = \max_{i,j \in U} \left(\sum_{a \in A - \{t\}}^{1 \leq k \leq p} \mathbf{c}_{ai}[k] - \sum_{a \in A - \{t\}}^{1 \leq k \leq p} \mathbf{c}_{aj}[k] \right)$$

Also, the cost models should be rewritten. For example, the linear cost model 6.15 can be replaced by the following:

$$\forall i \in U : \text{cost}_i = \text{unit}_i \sum_{a \in A - \{t\}}^{1 \leq k \leq p} \mathbf{c}_{ai}[k] \mathbf{l}_a[k]$$

6.3.2 Computing Prepending by Computational Geometry

We now show how approaching the optimal prepending problem by Computational Geometry instruments can lead to efficient algorithms. For simplicity, we focus on the case in which t has 3 upstreams (i.e., $m = 3$). Our considerations can be generalized to a greater number of upstreams.

For any choice of prepending, it is possible to map each AS $a \in A - \{t\}$ to a point in a 3-dimensional Euclidean space, parametrically with respect to the amount of prepending. In particular, if we consider a coordinate system $OX_1X_2X_3$, this point has coordinates $[x_1, x_2, x_3]^T$, where $x_i = d_{ai} + w_i, i = 1, 2, 3$.

This space can be partitioned into regions so that all the points falling in the same region correspond to ASes in the same $A_i, i = 1, 2, 3$. Points belonging to A_0 fall on the border between regions. Changing the prepending amounts results in translating all the points or, equivalently, the coordinate system. As a consequence, points can shift from one region to another.

More precisely, the regions and the sets $A_i, i = 1, 2, 3$ are put in correspondence in the following way:

$$A_1 \leftrightarrow \begin{cases} x_1 < x_2 \\ x_1 < x_3 \end{cases} \qquad A_2 \leftrightarrow \begin{cases} x_2 < x_1 \\ x_2 < x_3 \end{cases} \qquad A_3 \leftrightarrow \begin{cases} x_3 < x_1 \\ x_3 < x_2 \end{cases} \tag{6.16}$$

Each pair of regions corresponding to $(A_i, A_j), i, j = 1, 2, 3, i \neq j$ is separated by a border B_{ij}:

$$B_{12} : \begin{cases} x_1 = x_2 \\ x_3 \geq x_1 \end{cases} \qquad B_{23} : \begin{cases} x_2 = x_3 \\ x_1 \geq x_2 \end{cases} \qquad B_{13} : \begin{cases} x_1 = x_3 \\ x_2 \geq x_1 \end{cases} \tag{6.17}$$

The union of B_{12}, B_{13}, and B_{23} corresponds to the set A_0. The intersection of B_{12}, B_{13}, and B_{23} defines a straight line $r : x_1 = x_2 = x_3$.

Let A_i $[A'_i]$ $i = 1, 2, 3$ be the set of ASes $a \in A - \{t\}$ that use upstream i to reach P when using prepending amounts w_i $[w'_i]$. The following property holds.

$$\forall k > 0, i = 1, 2, 3 : w'_i = w_i + k \Rightarrow A'_j = A_j, j = 1, 2, 3 \tag{6.18}$$

This property can be proved by showing that using prepending $w'_i = w_i + k$ is equivalent to translating points in the same direction as r.

As a consequence of Property 6.18, studying the effect of prepending does not require considering all the combinations of prepending amounts. For example, it is possible to keep the prepending on an upstream i fixed while only altering the others. Hence, the component x_i is fixed as well, which corresponds to projecting points on one of the coordinate planes. Consider that, if we fix a prepending amount, others may become negative while searching for optimal prepending. This can be amended by translating all the points in the same direction as r in order to move them to the first octant without altering the composition of the sets $A_i, i = 0, 1, 2, 3$.

To exploit the symmetry of our construction, we project the points on the plane $H : x_1 + x_2 + x_3 = 0$, which passes through origin O and whose normal is r. After projection, borders 6.17 become half lines. In order to easily figure out the disposition of the points and the shape of the regions after projection, we now consider a coordinate system $OX'_1X'_2$ such that axis X'_2 coincides with

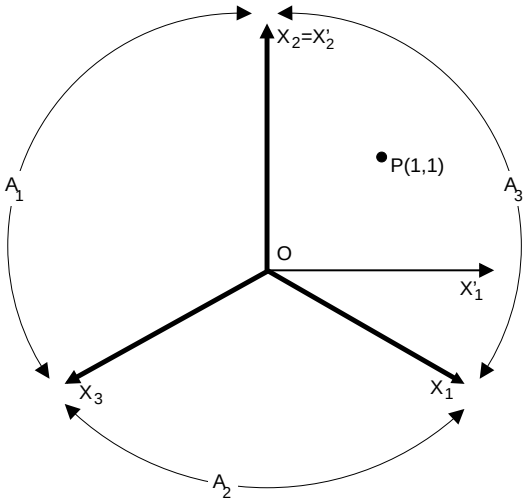


Figure 6.3: *Projection on plane $H : x_1 + x_2 + x_3 = 0$ of the axes X_1, X_2, X_3, in the coordinate system $OX'_1X'_2$. Point P is the projection of the straight line $x_1 = t$, $x_2 = t + \frac{\sqrt{6}-\sqrt{2}}{2}$, $x_3 = t - \sqrt{2}$.*

the projection of X_2 over H and axis X'_1 also belongs to H and is orthogonal to X_2.

Using standard linear algebra, it's easy to see that a point $\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$ becomes $\begin{bmatrix} x'_1 & x'_2 \end{bmatrix}^T = \begin{bmatrix} \frac{x_1-x_3}{\sqrt{2}} & \frac{-x_1+2x_2-x_3}{\sqrt{6}} \end{bmatrix}^T$ in the coordinate system $OX'_1X'_2$. Note that each point on H is the projection of a line parallel to r.

In Figure 6.3 half lines X_1, X_2, X_3 represent the projection of the positive half of the of axes X_1, X_2, X_3, which partition the plane into three regions corresponding to A_1, A_2, A_3. Points falling over X_1, X_2, or X_3 correspond to A_0. Optimal prepending is obtained by placing origin O so that one of the objective functions 6.11, 6.12, 6.13, 6.14 is minimized. Note that in 6.11, 6.12, 6.13, 6.14 the values of the c_{ai} can be easily computed on the basis of the disposition of the points on H.

Let $M = \max_{a \in A-\{t\}, i=1,2,3} d_{ai}$. The hexagon of side M lying on H and centered in O contains all the points representing the ASes. Therefore, by exploiting projection on H, we need $3M^2$ attempts to find the position of origin O that gives optimal prepending. Figure 6.4 helps in intuitively understanding

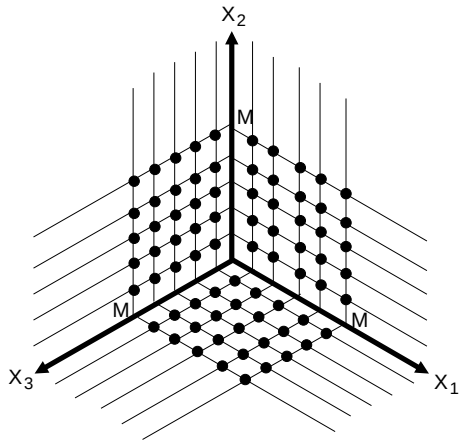


Figure 6.4: *Points represent all the possible placements of origin O, corresponding to different prepending assignments. The drawing lies on plane* $H : x_1 + x_2 + x_3 = 0$. *Finding optimal prepending requires placing origin* O *in* $3M^2$ *different points.*

this. For each prepending choice, it takes $O(n)$ time to compute the value of any of the objective functions 6.11, 6.12, 6.13, 6.14. Since, in general, M is $O(n)$, this leads to the following conclusion:

> Given a network with n ASes and a target AS announcing a prefix to 3 upstream providers, the optimal amount of prepending to be used in the announcements can be found in $O(n^3)$ time.

This holds for any of the objective functions Equal-Cardinality, Equal-Load, Shape-Bandwidth, Equal-Cost, Equal-Cost-Threshold described in Section 6.3. This result can be generalized to an arbitrary number m of upstreams, thus leading to computational complexity $O(n^m)$.

6.4 Remarks about Computational Complexity

Section 6.3.2 proposes an algorithm to compute the optimal amount of prepending in $O(n^3)$ time for the case of an ISP with 3 upstream providers. One can argue that more clever techniques can lead to better results.

Exploring the prepending space by local search techniques is hard due to local minima. For example, consider a hill climbing search, in which it is possible

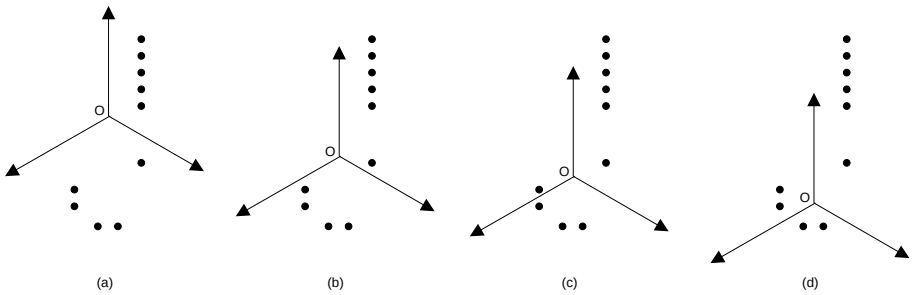


Figure 6.5: *Example of situation in which using hill climbing search is not effective for pursuing* EQUAL-CARDINALITY.

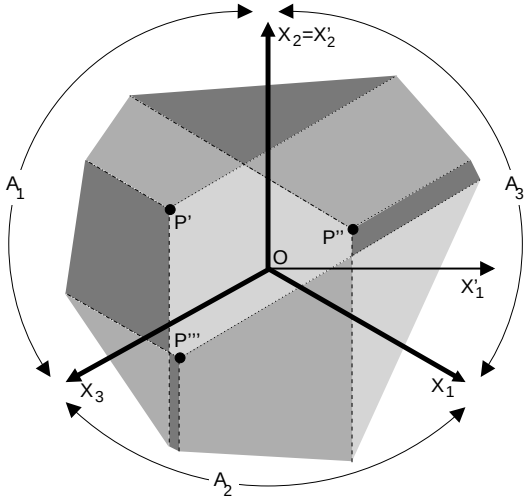


Figure 6.6: *A configuration of points (P', P'', P''') after projection on H. The drawing lies on plane H. Shaded areas represent equivalence areas.*

to move from one prepending configuration by changing only one $w_i, i = 1, 2, 3$ of one unit. Figure 6.5 shows a situation in which this kind of search is not effective for pursuing EQUAL-CARDINALITY. Figure 6.5(a) shows a local minimum with $f(c_{ai}) = 5$; by moving against hill climbing, in 6.5(b) we get $f(c_{ai}) = 6$; then $f(c_{ai}) = 5$ in 6.5(c) and a better solution with $f(c_{ai}) = 4$ in 6.5(d).

An alternative approach consists in trying to avoid all the prepending assignments that lead to the same composition of the sets $A_i, i = 0, 1, 2, 3$. For this purpose, plane H can be considered partitioned in *equivalence areas*, so that placing O in any of the points of an area leads to the same sets A_i. These areas are represented with different gray tones in Figure 6.6.

Let R be the set of all equivalence areas and $n = |A|$. Suppose to add one point at a time. Since each point introduces $(n-1)+2$ new equivalence areas, we have $|R|_n = |R|_{n-1} + n + 1$, and $|R|_1 = 3$. This leads to $|R|_n = \frac{n^2+3n+2}{2}$.

In real settings it is not unusual that a small set of ASes is responsible for a large amount of the ISP incoming traffic [132]. In these situations we can consider only those ASes that generate most of the traffic, and hence we can have $n << M$ (i.e., even with a small number of ASes we may have large distance values). In this case, exploiting equivalence areas can lead to a great speedup.

The set of all the equivalence areas is described by an arrangement of $3n$ half lines. In [72] is presented an algorithm for describing and enumerating such an arrangement in $O(n^2)$ time, and this is shown to be optimal. This result is also valid for an arbitrary dimension. To be rigorous, the algorithm works with straight lines. However, considering the arrangement with straight lines instead of half lines does not increase the complexity of the arrangement itself. In fact, suppose to replace half lines with straight lines. Then we have $|R|_n = |R|_{n-1} + 6n - 1$ and $|R|_1 = 3$, which leads to $|R|_n = 3n^2 + 2n + 1$.

6.5 Applicability Considerations

In our model the lengths of the shortest AS-paths from each AS to each upstream are supposed to be known. However, we believe that this assumption does not prevent the approach to have a practical impact.

First, observe that in order to compute shortest AS-path lengths the knowledge of the whole network is not required. On the contrary, it is possible to announce a prefix to one upstream at a time and to measure how it reaches the remote ASes. Such preliminary measurement may also be performed using a test prefix which does not carry production traffic. As already pointed out in Chapter 3, most routing policies do not discriminate between different prefixes originating in the same AS, and therefore the use of a test prefix is unlikely to affect the measurement.

Second, the amount of information needed in practical cases is far smaller than the theoretical bound. A few ASes can be responsible for a large amount of

incoming traffic. Restricting to these critical ASes, which can be automatically identified by exploiting widely adopted tools such as NetFlow [225], would provide a reasonable solution with a limited amount of input data.

Finally, the AS-paths of the announcements reaching the remote ASes may be retrieved from several sources. The Oregon Route Views Project [198] and the RIPE NCC RIS [173] collectors cumulatively offer a view of hundreds of ASes. Also, traceroute servers and BGP looking glasses [188] provide a long list of ASes for which such information is available. As a last resort, for those critical ASes not covered by these sources, an assumption of symmetric routing may be tried, and the reverse AS-paths may be considered where necessary.

Despite the above considerations, and due to behaviors that are not accounted for in our model (local preferences, "prefer customer" policies, etc.), the computed prepending amounts may still be suboptimal and local search, as described in [175], may be required to refine them.

6.6 Conclusions

This Section focuses on the problem of optimizing the distribution of the incoming traffic of an ISP. In particular, we introduce a model to compute the optimal prepending that the ISP should use in its BGP announcements. Several optimality criteria are proposed, and we show how to compute optimal prepending both by using an Integer Linear Programming formulation and by exploiting Computational Geometry techniques.

We also show that computing the optimal prepending amounts requires knowing network information that can be obtained from publicly available data sources.

The model and the approaches introduced in this Chapter still leave room for improvements.

A valuable complement to the presented analysis would come from determining bounds on the algorithmic complexity of the problem of determining optimal prepending. This would give a more accurate view of the tractability of the problem, and would possibly reveal the need of other heuristic approaches.

The techniques presented in this Chapter still miss a case study. Experimenting these techniques would help in better assessing their applicability and effectiveness.

It would also be interesting to investigate the impact of prepending choices on the stability of the Internet. Suppose that the interest for interdomain traffic engineering increases and suppose that several ISPs start performing aggressive

routing control based on prepending. That is, suppose that ISPs systematically "play" the game of influencing routing using prepending. By applying game theory techniques, it would be possible to study whether this game admits a (Nash) equilibrium. Interesting studies about the influence of selfish routing and traffic engineering on routing stability have been proposed in [119, 69, 253, 70]. However, these works use different traffic engineering objective functions and do not consider the effect of applying AS-path prepending.

Chapter 7

Interplay of Routing Policies at different Autonomous Systems

> Day after day, day after day,
> We stuck, nor breath nor motion;
> As idle as a painted ship
> Upon a painted ocean.
>
> Rime of the Ancient Mariner
> Samuel Taylor Coleridge

Administrators managing an Autonomous System usually have an interest in deploying BGP configuration policies that implement specific requirements to ensure the correct operation of the AS. These requirements may span from traffic engineering to connection robustness, and from the enforcement of commercial agreements to the sharing of load over multiple links.

The widespread use of BGP configuration policies in the Internet globally affects the way traffic is routed to its destination [123, 73], often violating the shortest path rule. This effect on routing paths is the consequence of the complex interactions of routing policies at several sites. While network administrators have some degree of freedom on the choice of the routing policies to use within the ISP they operate, there is no guarantee that these policies will achieve the desired requirements. In fact, administrators of neighboring ASes may deploy polices that clash with and prevent the enforcement of the ones of the ISP. It may also be the case that conflicting policies lead to persistent routing oscillations [194, 124, 125, 191], which may affect the reachability of some destinations and is therefore undesirable.

This Chapter presents methods to analyze the interaction of routing polices at different Autonomous Systems. In particular, it describes a methodology to verify whether traffic to a certain destination prefix can traverse an arbitrarily chosen sequence of ASes (*feasibility* of an AS-path). It also presents an approach to determine the level of preference assigned by an AS to two equally long AS-paths (*path preference comparison*). Independently from this, it presents a theoretical model and some interesting properties to study scenarios in which persistent routing oscillations may occur. This model is partly based on the Stable Paths Problem formalism as proposed in [125].

7.1 Checking the Feasibility of AS-paths

This Section proposes a methodology that, based on the probing primitives presented in Chapter 3, allows to determine whether an arbitrarily chosen sequence of ASes (AS-path) could be traversed by traffic to a specific destination. This property of an AS-path has already been addressed as *feasibility* in Section 3.1: an AS-path $A_n \dots A_2 A_1$, where A_1 is the origin AS, is *feasible* for a prefix p if the policies of each A_i permit A_i to announce p to A_{i+1} with AS-path $A_i \dots A_1$. Observe that the successful delivery of a packet to p depends on the interaction of interdomain routing policies at ASes $A_n \dots A_1$ along the chosen AS-path. Also consider that feasible AS-paths are not necessarily observed in BGP routing tables obtained from collection points [173, 198], for example because some of them may only take over in case of link faults. In the following we also make use of the notion of *level* of an AS defined in Chapter 3: the *level* of an AS X is the length of the shortest directed path from AS Z, the originator of p, to X.

Suppose that we are interested in knowing whether a certain AS-path $\mathcal{P}$ is feasible for a prefix p. For this purpose, we may use the following algorithm, which we name the *nailed-path* algorithm.

Assume $\mathcal{P}$ ends at an AS A containing a collector-peer or looking glass C and consider a feasibility graph obtained using one of the methods described in Section 3.4.1. Prohibit all the ASes in levels up to and including the level of A except for the ASes in $\mathcal{P}$. Now observe the AS-path $\mathcal{Q}$ seen by C: it is likely that either $\mathcal{Q} = \mathcal{P}$ and the path is feasible, or C does not see the prefix and the path is not feasible. If $\mathcal{Q} \neq \mathcal{P}$ (i.e., ASes or peerings that were not in the initial feasibility graph have been revealed), we repeat the above procedure after including the newly discovered ASes in the prohibited set. If $\mathcal{Q}$ reveals a shortcut between two ASes in $\mathcal{P}$, then the feasibility of $\mathcal{P}$ cannot be determined

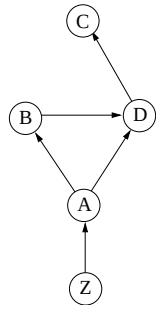


Figure 7.1: *A topology in which the shortcut between A and D impairs the ability to determine the feasibility of path ZABDC using AS-set stuffing.*

(see Figure 7.1).

If $\mathcal{P}$ ends at an AS A that is not a collector-peer a simple approach might be the following:

1. Prohibit all ASes in levels up to and including the level of A except for the ASes in $\mathcal{P}$.
2. Check whether $\mathcal{P}$ is a subpath of one of the paths seen by the route collectors. If this is true, $\mathcal{P}$ is feasible otherwise the feasibility of $\mathcal{P}$ is unknown.

Unfortunately, the first step can prohibit ASes that are eligible for connecting AS A to one of the collectors, thus limiting the effectiveness of this approach. The following algorithm provides a more effective solution.

1. Select any collector-peer or looking glass C. Announce prefix p with no prohibited ASes. Let $\mathcal{S}$ be an empty set of ASes.
2. While C sees an AS-path for p, do
   a) Let $\mathcal{Q}$ be the AS-path for p seen by to C.
   b) If $\mathcal{Q}$ matches $\mathcal{P}$, starting from the origin AS, return *feasible*.
   c) Compare the ASes in $\mathcal{P}$ and $\mathcal{Q}$ in order, starting from the origin AS, and let i be the position where the two paths differ. If $\mathcal{Q}[i]$ appears in $\mathcal{P}$, return *unknown*.
   d) Let $\mathcal{S} = \mathcal{S} \cup \{Q[i]\}$.

e) Announce prefix p prohibiting ASes in $\mathcal{S}$.

3. Return *unknown*.

This algorithm is more accurate than the simple approach shown above since it incrementally prohibits ASes and cannot accidentally cut a feasible path from A to C. Nevertheless, its capability to recognize the feasibility of $\mathcal{P}$ still depends on the choice of C, therefore performing one execution for each possible choice of C may improve effectiveness. However, even for a single choice of C, this algorithm may require many more announcements than the simple approach.

To verify our path feasibility determination techniques we obtained an initial feasibility graph using withdrawal observation, chose arbitrary paths on the graph starting from the origin AS and ending in a route collector, and applied the nailed-path algorithm to determine which of the paths were feasible. Examples are shown in Table 7.1. Column "Prefix" shows the prefix we tested and thus whether the test was performed in the IPv6 or IPv4 network. Column "Path" shows the path we tested. Column "UTC Time" specifies the time at which the BGP announcement was sent and column "AS-set" shows the AS-set announced. Column "Observed Path" shows the path that was observed after BGP propagation and column "Feasible" shows whether the path was feasible. Remember that if no AS-path was observed then we can affirm that a path is not feasible; if another AS-path was observed, it is not possible to determine the feasibility.

7.2 Revealing the Preference Associated to AS-paths

Given two feasible AS-paths $\mathcal{P}_1$ and $\mathcal{P}_2$ ending at the same observation point (a collector-peer or looking glass) C in AS A, we may use AS-set stuffing to determine which of the two AS-paths is preferred by C.

To determine which path C prefers, we obtain a feasibility graph as described in Section 3.4.1 and attempt to ensure that the only announcements received by C for p have the paths $\mathcal{P}_1$ and $\mathcal{P}_2$. Namely, we prohibit all the ASes in all levels up to the level of A except the ASes in $\mathcal{P}_1 \cup \mathcal{P}_2$. This may be enough for C to see either $\mathcal{P}_1$ or $\mathcal{P}_2$. If not, the announcement may lead to the discovery of new ASes which are not in $\mathcal{P}_1$ or in $\mathcal{P}_2$ and were not previously visible in the feasibility graph. In this case it is sufficient to prohibit the new ASes and repeat the announcement until no new ASes are discovered.

Prefix	UTC Time	Path	AS-set	Observed Path	Feasible
2001:a30::/32	2005-02-03 18:43:40	3257 2497 2500 4691 33 15589 5397	{4725, 5511, 7660, 18084, 7684, 6939, 10566, 1275, 3320, 5609, 2914, 14277, 6175, 5623, 278, 4697, 3549, 13944, 4555, 15897, 680, 31103, 1299, 5539, 9112, 3246, 8657, 8447, 1257, 4725, 17715, 6435, 145, 12779, 25358, 20965, 1853, 3265, 16713, 109, 6762, 559, 29686, 3344, 8664, 12968, 13110, 8763}	3257 2497 2500 4691 33 15589 15589 5397	Yes
2001:a30::/32	2005-02-21 17:02:27	3333 3265 6175 13977 6939 15589 5397	{10566, 33, 1275, 3320, 5623, 7580, 6435, 5539, 12477, 4716, 15897, 4697, 5609, 1299, 2607, 31103, 293, 4555, 2042, 8175, 145, 6342, 8447, 3549, 12779, 1257, 3257, 8763, 1752, 2500, 4725, 25358, 3246, 20965, 1853, 559, 1103, 29686, 3245, 513}	3333 3265 6175 13977 6939 6939 15589 15589 5397	Yes
2001:a30::/32	2005-02-21 16:18:25	3333 1103 2607 1275 15589 5397	{6939, 10566, 33, 3320, 5623, 13944, 7580, 6435, 6175, 5539, 14277, 4716, 15897, 4697, 5609, 2914, 1299, 31103, 293, 4555, 2042, 8175, 145, 6342, 8447, 3549, 12779, 3265, 1257, 3257, 8763, 1752, 2500, 4725, 25358, 3246, 20965, 1853, 559, 29686, 3425, 513, 278, 12859, 8472, 6830, 18084, 2497, 7684, 29377, 1930, 11537, 7660, 5511}	3333 1103 2607 1275 15589 15589 5397	Yes
2001:a30::/32	2005-02-23 15:27:39	6175 4555 13944 6939 15589 5397	{1275, 3320, 33, 10566, 2607, 109, 5609, 293, 513, 31103, 1299, 2497, 5623, 4725, 17715, 3549, 5539, 14277, 2914, 4697, 4716, 6435, 7580, 3748, 2042, 15897, 2549, 6762, 3425, 9264, 5430, 20965, 1103, 29686, 1752, 1853, 25358, 6830, 559, 3257, 12779, 3265, 2500, 145, 8763, 4691, 3786}	—	No
2001:a30::/32	2005-02-23 15:38:30	6175 145 7580 10566 15589 5397	{1275, 3320, 6939, 33, 2607, 109, 5609, 293, 513, 31103, 1299, 2497, 5623, 4725, 17715, 3549, 5539, 14277, 2914, 4697, 4716, 6435, 13944, 3748, 2042, 15897}	—	No
2001:a30::/32	2005-04-19 13:56:56	559 1299 3320 1275 15589 15589 5397	{33, 109,145, 278, 293, 513, 559, 1103, 1257, 1752, 1853, 2042, 2497, 2500, 2607, 2914, 3257, 3265, 3292, 3352, 3425, 3549, 3748, 3786, 4691, 4697, 4716, 4725, 5609, 5623, 6175, 6320, 6342, 6435, 6830, 6939, 7033, 8447, 10566, 12779, 13944, 14277, 17715, 17965, 20965, 24136, 24895, 29686, 31103, 32266}	559 1299 3320 15589 15589 5397	Unknown
84.205.89.0/24	2005-07-05 11:31:44	8468 6461 701 702 13030 12654 12654	{8210, 8220, 3320, 286, 8447, 20932, 9044, 12793, 13237, 2497, 8434, 513, 8289, 8342, 16034, 1239, 209, 3561, 6762, 2914, 1299, 3356}	8468 6461 13030 12654 12654	Unknown
84.205.89.0/24	2005-07-05 11:36:01	8468 6461 13030 12654 12654	{8210, 8220, 3320, 286, 8447, 20932, 9044, 12793, 13237, 2497, 8434, 513, 8289, 8342, 16034, 1239, 209, 3561, 6762, 2914, 1299, 3356, 701, 702}	8468 6461 13030 12654 12654	Yes
84.205.89.0/24	2005-07-05 12:31:01	8468 9044 13030 12654 12654	{8210, 8220, 3320, 286, 8447, 20932, 12793, 13237, 2497, 8434, 513, 8289, 8342, 16034, 1239, 209, 3561, 6762, 2914, 1299, 3356, 701, 702, 6461}	8468 9044 13030 12654 12654	Yes
84.205.73.0/24	2005-07-05 13:29:29	2116 1299 1239 13030 12654 12654	{209, 286, 513, 701, 702, 2497, 2914, 3320, 3356, 3561, 6461, 6762, 8210, 8220, 8289, 8342, 8434, 8447, 8468, 9044, 12793, 13237, 16034, 20932}	—	No

Table 7.1: *Path feasibility determination results.*

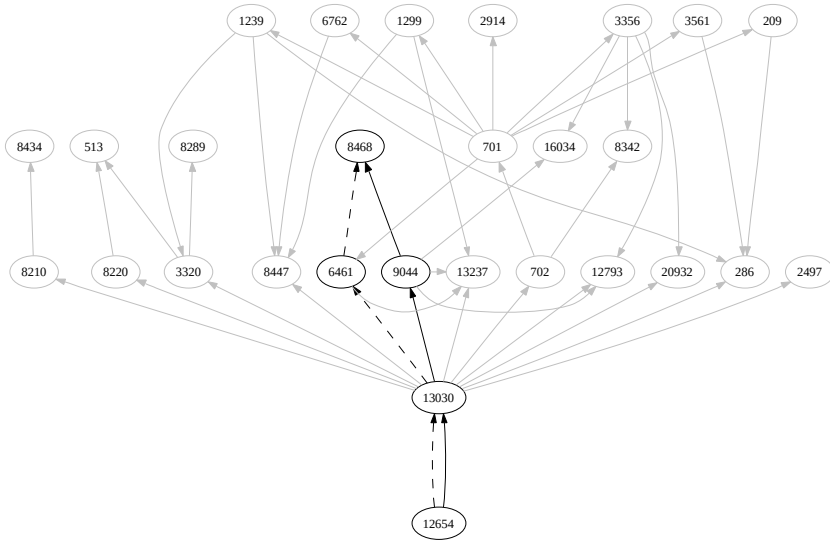


Figure 7.2: *Experiment setting for comparing the preference assigned by AS8468 to paths 8468 6461 13030 12654 (dashed) and 8468 9044 13030 12654 (solid). ASes in gray are prohibited.*

The announcement may also lead to the observation of another path made up exclusively of ASes which belong to either $\mathcal{P}_1$ or $\mathcal{P}_2$. In this case it is not possible to determine whether C prefers $\mathcal{P}_1$ or $\mathcal{P}_2$. This may occur only if there is a feasible peering between an AS in $\mathcal{P}_1$ and an AS in $\mathcal{P}_2$. Furthermore, if $\mathcal{P}_1$ and $\mathcal{P}_2$ have ASes in common in addition to A, it is not possible to determine which AS-path is preferred by C, since routers in one of the common ASes may have chosen between the two paths and only re-announced one of them, resulting in only one of them reaching A. Finally, since this technique requires us to determine whether $\mathcal{P}_1$ and $\mathcal{P}_2$ are feasible, it cannot be applied if it is not possible to determine the feasibility of the paths as described in Section 7.1.

We tested our path preference comparison technique on various paths ending in route collectors. Table 7.2 shows some of our results.

Figure 7.2 shows the setting used for the experiment in the third row. The graph was obtained using withdrawal observation. The aim is to determine whether the collector-peer in AS8468 prefers the path through AS6461 or AS9044. All the ASes that are not part of the two paths (shown in gray) are

Prefix	UTC Time	Collector	AS-path $\mathcal{P}_1$	AS-path $\mathcal{P}_2$	Observed path	Preferred path
2001:a30::/32	2005-02-21 16:30:28	3333	3333 3265 6175 13944 6939 15589 5397	3333 1103 3425 293 3320 15589 5397	3333 1103 3425 293 3320 15589 5397	$\mathcal{P}_2$
2001:a30::/32	2005-04-19 12:38:09	1103	1103 2607 1275 15589 5397	1103 20965 1299 3320 15589 5397	1103 20965 1299 3320 15589 15589 5397	$\mathcal{P}_2$
84.205.89.0/24	2005-07-05 12:31:01	8468	8468 6461 13030 12654 12654	8468 9044 13030 12654 12654	8468 9044 13030 12654 12654	$\mathcal{P}_2$

Table 7.2: *Path preference comparison results.*

prohibited, therefore the preferred path is the one seen by AS8468.

The second example in the table is particularly interesting because it shows an AS preferring a longer path over a shorter one: between 1103 2607 1275 15589 5397 and 1103 20965 1299 3320 15589 15589 5397, AS1103 (SURFnet, the Dutch research network) prefers the latter. This suggests that AS1103 is explicitly configuring its routers to prefer paths coming from AS20965 (the Géant European research network).

7.3 Instabilities Caused by Routing Policies

This Section introduces a model that can be used to study abnormal routing scenarios in which bad interactions of routing policies deployed at different Autonomous Systems result in routing choices changing indefinitely. This kind of problem is not unlikely to happen in real networks because, for example, link or device failures may lead to unexpected policy interactions [185].

Approaches to study configurations leading to persistent oscillations have already been presented in works by Griffin, Gao et al. [190, 192, 194, 124, 125, 191] and, independently, by Varadhan et al. [102]. Based on some of the conclusions drawn in [102], the works by Griffin, Gao et al. take advantage of a modelling of the unstable routing system via a formalism known as *Stable Paths Problem* to formally prove convergence properties. The authors describe several sample routing policy scenarios that, if accidentally implemented on a live network, lead to persistent oscillations. They characterize the likelihood of a network incurring oscillations in terms of the presence or absence of a generalized structure (the *dispute wheel*) that models bad policy interactions. They also prove that checking for the existence of a stable routing state is NP-complete. Many of the works studying routing instabilities also rely on the Stable Paths Problem model.

Several authors have proposed improvements to the currently running BGP version 4 [252] in the intent to make it robust against routing oscillations. Varadhan et al. [102] suggested the use of provably safe procedures, such as shortest path routing, to prevent instabilities, and proposed to rely on information from the Internet Routing Registry [216, 60, 66] in order to detect policies that would bring about oscillations.

Griffin et al. [194, 191] have introduced a *simple path vector protocol (SPVP)* which, in its *safe* variant, suppresses cycling messages and prevents instabilities from happening based on the collection of a history of the announcements.

The authors also discuss possible extensions to BGP that accommodate the stabilizing properties of SPVP.

Gao and Rexford have described in [124] a set of guidelines for deploying policies that ensure the convergence of a routing system while preserving typical traffic engineering choices of ISPs. The idea is to impose a partial order on the routes to a certain destination by relying on a hierarchical model of the Internet [260, 122, 115, 187, 64]. The same authors have proposed in [125] a model for backup routing that preserves routing stability under any combination of link or router failures. They show how to implement this model in BGP by using a new attribute that conveys the *avoidance level* of a route.

Griffin et al. in [189] and Sobrinho in [97, 98] have introduced models to capture the characteristics of path vector protocols and to abstract them from the implementation. The work by Griffin et al. [189] addresses issues related to the design of policy specification languages. They argue that policy languages should prevent operators from being able to deploy configurations that lead to unexpected routing anomalies, and they clearly define and explore the dimensions of the design space of these languages. Sobrinho [97, 98] introduces a solid theoretical model based on an algebraic framework which allows to study the convergence properties of path vector protocols. Conditions ensuring convergence are presented and proved. The author also describes several scenarios that can be implemented with BGP, including policies that provide for the enforcement of commercial relationships or backup routing. He reformulates the guidelines from [124] and proves their validity in the algebraic framework.

In more recent times, Sobrinho and Griffin [193] have collaborated to define a theoretical model that allows to describe routing protocols while applying a clear separation of routing protocol mechanisms from routing policies. Based on Sobrinho's algebraic framework [97], they introduce a *Routing Algebra Meta-Language (RAML)*, which potentially allows an operator to implement an arbitrary routing protocol on a RAML-enabled device. The use of an underlying routing algebra allows to derive convergence conditions for a genering routing protocol in a straightforward way, once a RAML description is available.

Cobb et al. [101] have proposed a routing model in which the participant entities have freedom of choice on routing policies, provided that each node selects a path that is consistent with those chosen by its descendants in the routing tree. Stability of the routing system is ensured by handling *tentative paths* via diffusing computations, and by appropriately eliminating nodes from the routing tree in case faulty conditions occur.

Subramanian et al. have introduced in [113, 114] a hybrid link-state and path-vector routing protocol which overcomes some of the major shortcomings

of BGP. The hybrid protocol, named *HLP*, provides for better scalability, limits the global impact of local network faults, and achieves linear-time convergence by constraining the path exploration process.

Wang et al. in [68] have relaxed the constraints on route selection algorithms by considering an entire class of algorithms they call *rational*, which choose better routes whenever available. They extend the usual routing model which accommodates ranks for egress routes only, with a model that takes into account routing choices based on inbound traffic. They show that, under this model, even configurations that obey restrictions imposed by commercial agreements may lead to oscillating routing.

7.3.1 The Stable Paths Problem Model

This Section briefly describes the model that is most used in the literature to identify and examine BGP routing instabilities.

An instance of the *Stable Paths Problem* (SPP) model consists of a triple $S = (G, \mathcal{P}, \lambda)$ where:

- $G = (V, E)$ is a simple graph (i.e., an undirected graph with no loops or multiple edges). Node $0 \in E$ is special in that all the other nodes attempt to establish a path towards it.
- $\mathcal{P} = \bigcup_{v \in V} \mathcal{P}^v$, where $\mathcal{P}^v$ is the set of *permitted paths* from v to 0. Node v can only use paths in $\mathcal{P}^v$ to reach 0. Paths in $\mathcal{P}^v$ have distinct next hops (that is, the second node u differs in every path $(v\ u\ \ldots\ w\ 0) \in \mathcal{P}^v$).
- $\Lambda = \{\lambda^v | v \in V - \{0\}\}$, where $\lambda^v : \mathcal{P}^v \rightarrow \mathbf{N}$ is a nonnegative integer-valued *ranking function* that represents the level of preference assigned by v to its permitted paths.

A *path assignment* function $\pi : V \rightarrow \mathcal{P}$ describes the path $\pi(v)$ that node v has selected, among its permitted paths $\mathcal{P}^v$, to reach node 0. Informally, a path assignment is *stable* if every node has chosen the highest ranked path that is coherent with the choice made by the next hop. More formally, a stable path assignment is such that $\forall v \in V : \pi(v) = \text{best}(\text{choices}(\pi, v), v)$, where:

$$\text{choices}(\pi, v) = \begin{cases} \{(v\ u)\pi(u) | \{v, u\} \in E\} \cap \mathcal{P}^v & \text{if } v \neq 0 \\ \{(0)\} & \text{if } v = 0 \end{cases}$$

$$\text{best}(P, v) = \begin{cases} p \in P \text{ such that } \lambda^v(p) \text{ is maximal} & \text{if } P \neq \emptyset \\ \epsilon & \text{if } P = \emptyset \end{cases}$$

$(v\ u)\pi(u)$ denotes a path that is the concatenation of edge $\{v, u\} \in E$ with path $\pi(v)$ chosen by v. The symbol ϵ denotes the empty path, which indicates that a node is unable to reach 0. Also, the following assumptions hold:

- $\forall v \in V - \{0\} : \epsilon \in \mathcal{P}^v$: the empty path is always permitted.
- $\forall v \in V - \{0\}, p \in \mathcal{P}^v - \{\epsilon\} : \lambda^v(\epsilon) < \lambda^v(p)$: the empty path is the lowest ranked.
- $\forall v \in V - \{0\}, p_1, p_2 \in \mathcal{P}^v, p_1 \neq p_2 : \lambda^v(p_1) = \lambda^v(p_2) \Rightarrow p_1 = (v\ u)p_1' \wedge p_2 = (v\ u)p_2'$: same rank implies same next hop; paths using different next hops must be assigned different ranks.
- $\forall p \in \mathcal{P} : p$ is simple: paths cannot have repeated nodes.
- $\mathcal{P}^0 = \{(0)\};\ \pi(0) = (0)$.

A *solution* to the Stable Paths Problem is a stable path assignment. Griffin et al. have proved [191] that checking whether a solution to an instance of SPP exists is an NP-complete problem.

Observe that the Stable Paths Problem is a good abstraction of an inter-domain routing system:

- The nodes and edges of G may correspond to Autonomous Systems and to the BGP peerings existing between pairs of them. Node 0 may represent an AS that originates a specific network prefix: all the other nodes search for a path to reach it.
- Permitted paths at each node may describe BGP filters (`prefix-lists`, `route-maps`, etc.) that selectively discard announcements and, therefore, make the corresponding paths unusable.
- The ranking of paths at each node may be representative of the level of preference assigned by routing policies to paths using different next hops. This may be implemented in BGP by means of `local-preferences`, `metrics`, and so on.
- The empty path ϵ corresponds to a node having no more alternatives to reach the prefix originator in its routing table, and thus being unable to send traffic to node 0.

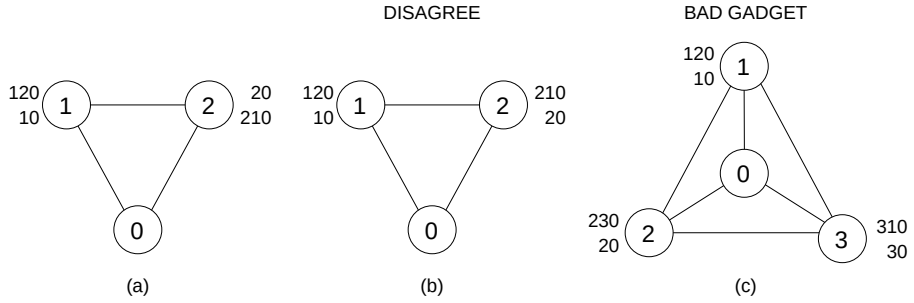


Figure 7.3: *(a) An instance of SPP that admits a solution. (b) An instance of SPP with multiple stable states. (c) An instance of SPP that does not admit any stable path assignment, regardless of the initial state.*

It is possible to completely describe an instance of SPP via an intuitive drawing. Figure 7.3(a) shows an instance that admits a stable path assignment. Nodes and edges describe the graph $G = (V, E)$. Permitted paths are listed beside each node. The rank assigned to each of them is implicitly coded in the ordering of the paths: less preferred paths come after the more preferred. For the instance of Figure 7.3(a), the solution to SPP is given by the path assignment $\pi(1) = (1\ 2\ 0)$, $\pi(2) = (2\ 0)$.

By simply reversing the ranking function at node 2 it is possible to build an instance that, depending on the timing of events, admits more than one solution or even incurs permanent instability. Figure 7.3(b) shows an instance that is known in the literature as Disagree. If node 1 chooses path (1 0) before any other event occurs, the system stabilizes on the assignment $\pi(1) = (1\ 0)$, $\pi(2) = (2\ 1\ 0)$. If, instead, node 2 selects path (2 0) before node 1 performs its choice, the stable assignment is $\pi(1) = (1\ 2\ 0)$, $\pi(2) = (2\ 0)$. If nodes 1 and 2 perform their choices synchronously, the system keeps oscillating between the assignments $\pi(1) = (1\ 2\ 0)$, $\pi(2) = (2\ 1\ 0)$ and $\pi(1) = (1\ 0)$, $\pi(2) = (2\ 0)$. In fact, whenever nodes 1 and 2 select the direct path to 0, a better indirect path through the neighboring node becomes available and is chosen. Yet, at the very same time at which the indirect path is selected, it becomes unavailable because it is not consistent with the choice of the neighboring node. For example, node 1 cannot keep the assignment $\pi(1) = (1\ 2\ 0)$ because it is not consistent with the path chosen by node 2, that is (2 1 0).

The SPP instance in Figure 7.3(c), that is commonly known as Bad Gadget, admits no solution, and path assignments indefinitely keep cycling. For

example, consider the following sequence of events:

1. At first, every node chooses a direct path, so that $\pi(1) = (1\ 0)$, $\pi(2) = (2\ 0)$, and $\pi(3) = (3\ 0)$.

2. Node 1 then detects that a better path through 2 is available, and changes its assignment to $\pi(1) = (1\ 2\ 0)$. Nodes 2 and 3 keep their current assignment.

3. Node 2 now switches to a better path through 3, and changes its assignment to $\pi(2) = (2\ 3\ 0)$. Nodes 1 and 3 keep their assignment unchanged.

4. Path (1 2 0) is no longer available at node 1 because of the change performed by 2. Node 1 therefore switches to $\pi(1) = (1\ 0)$. Nodes 2 and 3 keep their current assignment.

5. A better path through 1 is now available for node 3, which changes its assignment to $\pi(3) = (3\ 1\ 0)$. Nodes 1 and 2 keep their assignment unchanged.

6. Node 2 loses its best path because of the modified assignment at 3, and switches to $\pi(2) = (2\ 0)$. Nodes 1 and 3 keep their assignment unchanged.

7. Node 1 can now switch back to its best path, and turn its assignment into $\pi(1) = (1\ 2\ 0)$. Nodes 2 and 3 keep their current assignment.

8. The best path through 1 at node 3 is no longer available, and the assignment becomes $\pi(3) = (3\ 0)$. Nodes 1 and 3 keep their current assignment.

9. Continue from step 2.

Note that, regardless of the sequence with which events occur, every assignment eventually leads to a change on the part of some other node. This is a typical example of persistent instability. A generalization of the Bad Gadget structure has been introduced in [190] under the name of *dispute wheel*, and the absence of any such pattern of policies in a network has been proved to be a sufficient condition for the SPP instance to admit a solution [190, 191].

Also observe that abnormal settings such as the ones in Figure 7.3(b) and 7.3(c) are not unlikely to happen in real world. Figure 7.4 shows a possible description in RPSL [22, 44, 111] of the Disagree scenario.

Other kinds of interactions may lead to triggering of previously unfeasible path assignments. There is a class of configurations for which only one stable

```
aut-num: AS0
as-name: THE-ORIGIN
export:  to AS1 action aspath.prepend(AS0,AS0,AS0); announce AS0
export:  to AS2 announce AS0

aut-num: AS1
as-name: POOR-PROVIDER
import:  from AS-ANY accept ANY
export:  to AS-ANY announce ANY

aut-num: AS2
as-name: GOOD-PROVIDER
import:  from AS1 action pref = 0; accept ANY
import:  from AS0 action pref = 10; accept ANY
export:  to AS1 announce ANY
```

Figure 7.4: *RPSL code describing a configuration of policies that implements the* Disagree *structure. Remember that lower values of the RPSL* `pref` *attribute correspond to more preferred routes [22].*

state is possible, until a network event (such as a link failure) triggers a different stable path assignment that was originally unintended. Even if the event is recovered (the link is repaired), the configurations are such that the system does not revert to the original state. Such configurations are termed *BGP Wedgies* in [185]. Figure 7.5 shows a simple example of this kind of configuration. Underlined paths represent path assignments at each node. The initial state (a) is such that the backup link directly connecting 3 to 0 is intentionally unused. After a failure of the primary link between 1 and 0, the system switches to a different stable path assignment (b) that utilizes the backup link. Note that the path assignment in Figure 7.5(b) would not have been stable if the primary link had still been available, as the direct path from 1 to 0 is preferred by node 1. After recovering from the fault (c), the system settles to a new stable state, different from the initial one. There is no way of getting out of this state unless other events occur, such as a fault of the backup link.

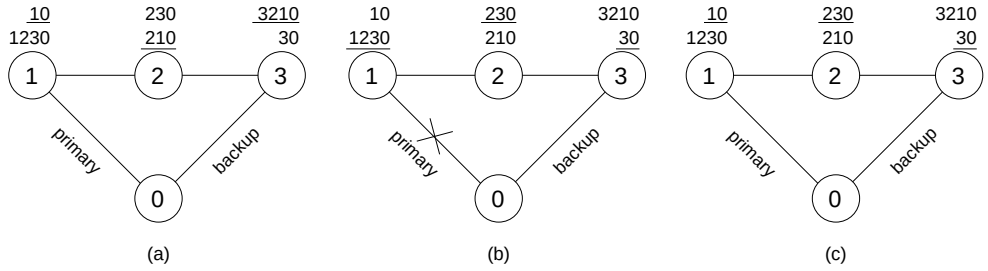


Figure 7.5: *An example of BGP wedgie: an instance of SPP in which a network failure reveals a previously unfeasible stable path assignment which is kept even after recovering from the fault.*

7.3.2 An Alternative Model to Investigate BGP Routing Instabilities

This Section introduces a variant of the Stable Paths Problem model that takes into account the timing with which events take place. The goal is to attempt to answer a question that is similar to that addressed by the Stable Paths Problem: given a scenario consisting of a network and a configuration of routing policies, is the resulting routing system stable? Yet, we would like to answer this question inside a framework that is general enough to relax any constraint on the sequencing of information exchanged between the nodes of the network and to capture all the possible timings of routing events.

We believe this is very important for understanding routing dynamics and policy interactions. In fact, while there are configurations that reach a stable state regardless of the sequence of events (see Figure 7.3(a)) and others in which any sequencing of message exchanges leads to persistent instability (see Figure 7.3(c)), there exist structures in which timings play an important role. The DISAGREE structure in Figure 7.3(b) and the wedgie of Figure 7.5 are examples of situations in which this is especially true.

Our model of BGP routing is based on the definition of instance of Stable Paths Problem introduced in Section 7.3.1. We slightly change this definition to model other aspects of the routing system. Even if we use terms and definitions that come from the BGP world (for example, we refer to the nodes as "routers" and assume that they exchange "update" messages), the model can still be considered a reasonable abstraction of a generic path vector protocol.

We replace the concept of path assignment with that of routing state. The

routing state of a router i is an $(n+1)$-uple $s_i =< p_{i0}, p_{i1}, \ldots, p_{in} >$ where:

$$n = |V| - 1 \tag{7.1}$$
$$p_{00} = (0) \tag{7.2}$$
$$\forall j \in V - \{0\} \quad : \quad p_{0j} = \epsilon \tag{7.3}$$
$$\forall j \in V \quad : \quad p_{ij} \in \mathcal{P}^i \tag{7.4}$$
$$\forall i, j \in V \quad : \quad \{i, j\} \notin E \Rightarrow p_{ij} = \epsilon \tag{7.5}$$
$$\forall i \in V - \{0\} \quad : \quad p_{ii} = \epsilon \tag{7.6}$$

Path p_{ij} represents how i can reach through its neighbor j the prefix announced by 0.

The *initial routing state* of router 0 is $< 0, \epsilon, \ldots, \epsilon >$; the initial routing state of any other router is $< \epsilon, \epsilon, \ldots, \epsilon >$.

The path that router i is currently using to reach 0 is the highest ranked among the p_{ij} and is identified by best(i). This function roughly corresponds to the path assignment π of the standard Stable Paths Problem formulation.

The *routing state S of the system* is an $(n+1)$-uple $S =< s_0, s_1, \ldots, s_n >$, where s_i is the state of router i. In the *initial routing state of the system* all the routers are in an initial state. Since S is an $(n+1)$-uple of $(n+1)$-uples, it can be represented in a compact way by an $(n+1) \times (n+1)$ matrix of paths, where each row represents the state of a router. We call this matrix the *state matrix* of the system. With reference to the example in figure 7.5(a), the following are two possible, randomly chosen, state matrices:

$$\begin{bmatrix} (0) & \varepsilon & \varepsilon & \varepsilon \\ (1\ 0) & \varepsilon & \epsilon & \varepsilon \\ \varepsilon & (2\ 1\ 0) & \varepsilon & \epsilon \\ \epsilon & \varepsilon & (3\ 2\ 1\ 0) & \varepsilon \end{bmatrix} \quad \begin{bmatrix} (0) & \varepsilon & \varepsilon & \varepsilon \\ (1\ 0) & \varepsilon & (1\ 2\ 3\ 0) & \varepsilon \\ \varepsilon & (2\ 1\ 0) & \varepsilon & (2\ 3\ 0) \\ (3\ 0) & \varepsilon & \epsilon & \varepsilon \end{bmatrix}$$

In the following we use a different epsilon symbol ε to indicate matrix coefficients that are forcedly set to ϵ as imposed by conditions 7.3, 7.5, and 7.6.

Our following findings are based on the observation that best(i) is the only portion of the state of the system that actually influences the paths used by packets to 0.

As routers exchange update messages, the system can change its current state. We indicate with best$_S(i)$ the path that is highest ranked at router i in state S. A *valid transition* $T : S' \rightarrow S''$ from state S' to state S'' is such that, $\forall i \in V - \{0\}, j \in V, \{i, j\} \in E$:

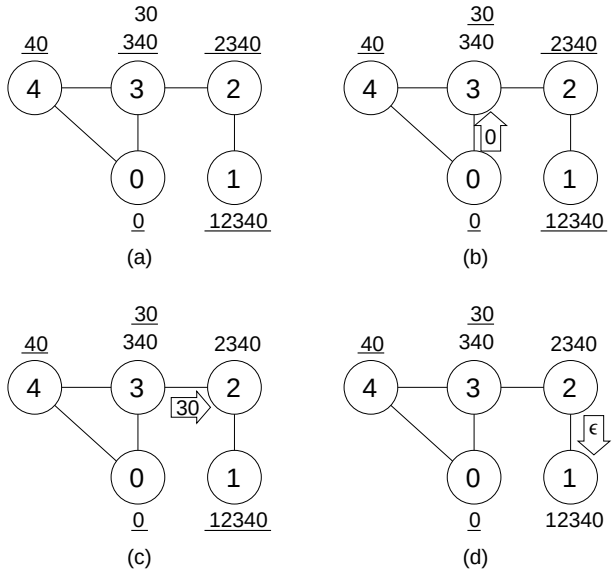


Figure 7.6: *An example showing how transitions can change the state of a routing system.*

- either $p''_{ij} = p'_{ij}$ (*i has not received any update from j*)
- or let k be a router in V such that $\text{best}_{S'}(j) = p'_{jk}$:
  - if $p'_{jk} = \epsilon$, then $p''_{ij} = \epsilon$ (*i has received a withdrawal from j*)
  - else (*i has received an announcement from j*):
    - if $ip'_{jk} \in \mathcal{P}^i$, then $p''_{ij} = ip'_{jk}$
    - else $p''_{ij} = \epsilon$.

Observe that this definition of valid transition allows to consider a wide variety of event timings in the system. Also consider that a transition might bring a router in a state such that its best choice is worse than the one in the previous state.

Figure 7.6 shows an example of transitions taking place in a routing system. The best paths at each router are underlined.

(a). We start from state (a), described by the following matrix:

$$\begin{bmatrix} 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & (1\ 2\ 3\ 4\ 0) & \varepsilon & \varepsilon \\ \varepsilon & \epsilon & \varepsilon & (2\ 3\ 4\ 0) & \varepsilon \\ \epsilon & \varepsilon & \epsilon & \varepsilon & (3\ 4\ 0) \\ (4\ 0) & \varepsilon & \varepsilon & \epsilon & \varepsilon \end{bmatrix}$$

(b). Now, suppose that router 3 receives an announcement directly from 0. Since path (3 0) is permitted at 3, the announcement is accepted and p_{30} is updated in the state matrix. Moreover, path (3 0) is the highest ranked at router 3, therefore best(3) is also updated. After this transition, state (b) is described by the following matrix:

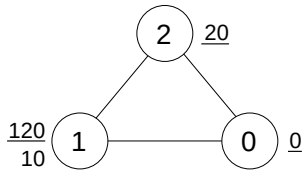
$$\begin{bmatrix} 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & (1\ 2\ 3\ 4\ 0) & \varepsilon & \varepsilon \\ \varepsilon & \epsilon & \varepsilon & (2\ 3\ 4\ 0) & \varepsilon \\ (3\ 0) & \varepsilon & \epsilon & \varepsilon & (3\ 4\ 0) \\ (4\ 0) & \varepsilon & \varepsilon & \epsilon & \varepsilon \end{bmatrix}$$

(c). At this point, router 3 propagates its newly selected best path (3 0) to its neighbor 2. As path (2 3 0) is not permitted at node 2, ϵ is stored in p_{23} in the state matrix. Also, since router 2 does not know any other path to reach 0 (i.e., $\forall j \in V : p_{2j} = \epsilon$), its connectivity to 0 is disrupted. After this change has taken place, state (c) is described by the following matrix:

$$\begin{bmatrix} 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & (1\ 2\ 3\ 4\ 0) & \varepsilon & \varepsilon \\ \varepsilon & \epsilon & \varepsilon & \epsilon & \varepsilon \\ (3\ 0) & \varepsilon & \epsilon & \varepsilon & (3\ 4\ 0) \\ (4\ 0) & \varepsilon & \varepsilon & \epsilon & \varepsilon \end{bmatrix}$$

(d). The event that triggers the last transition in this example is a withdrawal issued by router 2 to router 1. Upon receiving it, router 1 replaces p_{12} with the empty path ϵ. As a consequence, also router 1 loses its connectivity to 0. The matrix that describes state (d) is the following:

$$\begin{bmatrix} 0 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \epsilon & \varepsilon & \varepsilon \\ \varepsilon & \epsilon & \varepsilon & \epsilon & \varepsilon \\ (3\ 0) & \varepsilon & \epsilon & \varepsilon & (3\ 4\ 0) \\ (4\ 0) & \varepsilon & \varepsilon & \epsilon & \varepsilon \end{bmatrix}$$



$$\begin{bmatrix} 0 & \varepsilon & \varepsilon \\ \epsilon & \varepsilon & (1\ 2\ 0) \\ (2\ 0) & \epsilon & \varepsilon \end{bmatrix}$$

Figure 7.7: *An example of forwarding stable routing state that is not strictly stable.*

Stable Routing States

We now define the notion of stable routing state and introduce some properties that are valid in our model.

A routing state S is *strictly stable* if, for any valid transition $T : S \to S'$, we have that $S = S'$. Even if the the state matrix does not change, transitions can still occur in a strictly stable state, either because of absence of updates (trivial transition) or because of repeated updates.

A routing state S is (*forwarding*) *stable* if, for any valid transition $T : S \to S'$, we have that $\text{best}_{S'}(i) = \text{best}_S(i)$ for any $i \in V$.

The following Lemma trivially stems from the above definitions.

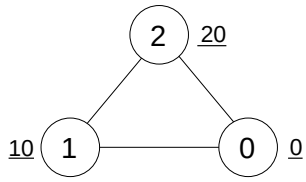
Lemma 7.1 *If a routing state is strictly stable, then it is also forwarding stable.*

Observe that the opposite does not hold. Figure 7.7 shows an example of routing state that is forwarding stable but not strictly stable. Forwarding stability can be easily checked. Strict stability does not hold because, at the very least, router 0 may announce the path (0) to router 1. Since (1 0) is permitted at 1, this announcement updates p_{10} with the new path, thus changing the state matrix to the following:

$$\begin{bmatrix} 0 & \varepsilon & \varepsilon \\ (1\ 0) & \varepsilon & (1\ 2\ 0) \\ (2\ 0) & \epsilon & \varepsilon \end{bmatrix}$$

Yet, node 1 does not change its best choice because (1 2 0) is still the highest ranked available path.

It is also interesting to notice that, in a strictly stable state, not all the paths of a router are necessarily consistent with the best paths of its neighbors. For



$$\begin{bmatrix} 0 & \varepsilon & \varepsilon \\ (1\ 0) & \varepsilon & \epsilon \\ (2\ 0) & \epsilon & \varepsilon \end{bmatrix}$$

Figure 7.8: *A strictly stable state for which not all the paths in the state matrix are consistent with the best choices of the routers.*

example, in Figure 7.8, $p_{12} = \epsilon$ is not consistent with best$(2) = (2\ 0)$. This is due to the restrictions imposed by permitted paths.

Lemma 7.2 $\forall i \in V - \{0\}, j, k \in V : p_{ij} \neq \epsilon \vee p_{ik} \neq \epsilon \Rightarrow p_{ij} \neq p_{ik}$. *The property remains valid also if p_{ij} and p_{ik} are considered in two different states of the system.*

The above Lemma easily follows from the fact that paths using a different next hop must necessarily be different.

We now prove a Lemma that will be useful to assert an interesting property of forwarding stable states.

Lemma 7.3 *If S is a forwarding stable state, then no pair of consecutive transitions $T' : S \rightarrow S'$, $T'' : S' \rightarrow S''$ exists such that $best_{S''}(i) \neq best_{S'}(i)$ for some $i \in V$.*

Proof. We proceed by contradiction. Suppose, ab absurdo, that the transitions T' and T'' do exist, and they are valid. Since S is forwarding stable, S' must be such that $\forall i \in V : \text{best}_S(i) = \text{best}_{S'}(i)$. Therefore, there must be (at least) one router j for which $\text{best}_{S''}(j) \neq \text{best}_{S'}(j)$. Assume, without restriction, that $\text{best}_S(j) = \text{best}_{S'}(j) = p'_{jh}$ and $\text{best}_{S''}(j) = p''_{jk}$. Also consider that the following straightforward properties hold:

$$\text{best}_S(h) = \text{best}_{S'}(h) \Rightarrow j\text{best}_S(h) = j\text{best}_{S'}(h) \tag{7.7}$$

$$\text{best}_S(k) = \text{best}_{S'}(k) \Rightarrow j\text{best}_S(k) = j\text{best}_{S'}(k) \tag{7.8}$$

$$\lambda^j(p''_{jk}) > \lambda^j(p''_{jh}) \tag{7.9}$$

Property 7.9 is true because j changes its choice in state S'' from h to k.

We now have three possibilities to satisfy 7.9, which we prove separately:

- $p''_{jh} = p'_{jh} \wedge p''_{jk} \neq p'_{jk}$
  We know from 7.9 that $\lambda^j(p''_{jk}) > \lambda^j(p''_{jh})$. Since all the transitions are valid and p''_{jk} has been updated, it must also be that $p''_{jk} = j\text{best}_{S'}(k)$ (note that p''_{jk} cannot be the empty path as it is higher ranked than p''_{jh}; this also ensures that $j\text{best}_{S'}(k) \in \mathcal{P}^j$). Because of 7.8 and of the hypotheses made in this branch of the proof, we can rewrite the inequality as $\lambda^j(j\text{best}_S(k)) > \lambda^j(p'_{jh})$. Therefore, there exists a transition $T^* : S \rightarrow S^*$ such that:

  - $\forall m \neq j, n \neq k : p^*_{mn} = p'_{mn}$; in particular, $p^*_{jh} = p'_{jh}$, which implies that $\lambda^j(j\text{best}_S(k)) > \lambda^j(p^*_{jh})$.
  - $p^*_{jk} = j\text{best}_S(k)$ (we know that $j\text{best}_{S'}(k) \in \mathcal{P}^j$, hence from 7.8 we also have that $j\text{best}_S(k) \in \mathcal{P}^j$).
  - $\text{best}_{S^*}(j) \neq p^*_{jh}$ because, at least, $\lambda^j(j\text{best}_S(k)) > \lambda^j(p^*_{jh})$.
  - $\text{best}_S(j) = p'_{jh} = p^*_{jh}$.

  Therefore, S cannot be a forwarding stable state, which is a contradiction.

- $p''_{jh} \neq p'_{jh} \wedge p''_{jk} = p'_{jk}$
  We proceed in the same way as in the first case. We know from 7.9 that $\lambda^j(p''_{jh}) < \lambda^j(p''_{jk})$. Since all the transitions are valid and p''_{jh} has been updated, we must have that $p''_{jh} \in \{j\text{best}_{S'}(h), \epsilon\}$. Because of 7.7 and of the hypotheses made in this branch of the proof, we can rewrite the inequality as $\lambda^j(j\text{best}_S(h)) < \lambda^j(p'_{jk})$ (the case for ϵ is trivial). Therefore, there exists a transition $T^* : S \rightarrow S^*$ such that:

  - $\forall m \neq j, n \neq h : p^*_{mn} = p'_{mn}$; in particular, $p^*_{jk} = p'_{jk}$, which implies that $\lambda^j(j\text{best}_S(h)) < \lambda^j(p^*_{jk})$.
  - $p^*_{jh} \in \{j\text{best}_S(h), \epsilon\}$ (ϵ accounts both for the case in which $j\text{best}_S(h)$ is not permitted at j and for the case in which j has received an empty path from h).
  - $\text{best}_{S^*}(j) \neq p^*_{jh}$ because:
    - if $p^*_{jh} = \epsilon$ it trivially follows that p^*_{jk} is more preferred (observe that $p^*_{jk} = p'_{jk}$ cannot be empty too because it also holds that $\lambda^j(p''_{jh}) < \lambda^j(p'_{jk})$);
    - if $p^*_{jh} = j\text{best}_S(h)$ then, at least, $\lambda^j(j\text{best}_S(h)) < \lambda^j(p^*_{jk})$.

    Therefore, $\exists g \in V, g \neq h : \text{best}_{S^*}(j) = p^*_{jg}$ and $p^*_{jg} \neq \epsilon$.

– $\text{best}_S(j) = p'_{jh} \neq p^*_{jg}$ from Lemma 7.2.

Hence, S cannot be a forwarding stable state, which is a contradiction.

- $p''_{jh} \neq p'_{jh} \wedge p''_{jk} \neq p'_{jk}$
  In analogy with the previous cases, we know from 7.9 that $\lambda^j(p''_{jh}) < \lambda^j(p''_{jk})$. Since all the transitions are valid and both p''_{jh} and p''_{jk} have been updated, we must have that $p''_{jh} \in \{j\text{best}_{S'}(h), \epsilon\}$ and $p''_{jk} = j\text{best}_{S'}(k)$ (note that p''_{jk} cannot be the empty path as it is higher ranked than p''_{jh}; this also ensures that $j\text{best}_{S'}(k) \in \mathcal{P}^j$). Because of 7.7 and 7.8 and of the hypotheses made in this branch of the proof, we can rewrite the inequality as $\lambda^j(j\text{best}_S(h)) < \lambda^j(j\text{best}_S(k))$ (the case for ϵ is trivial). Therefore, there exists a transition $T^* : S \to S^*$ such that:

  – $\forall m \neq j, n \neq h, n \neq k : p^*_{mn} = p'_{mn}$.

  – $p^*_{jh} \in \{j\text{best}_S(h), \epsilon\}$ (ϵ accounts both for the case in which $j\text{best}_S(h)$ is not permitted at j and for the case in which j has received an empty path from h).

  – $p^*_{jk} = j\text{best}_S(k)$ (we know that $j\text{best}_{S'}(k) \in \mathcal{P}^j$, hence from 7.8 we also have that $j\text{best}_S(k) \in \mathcal{P}^j$).

  – $\text{best}_{S^*}(j) \neq p^*_{jh}$ because:

    * if $p^*_{jh} = \epsilon$ it trivially follows that $p^*_{jk} = j\text{best}_S(k)$ is more preferred (observe that p^*_{jk} cannot be empty too because $p''_{jk} = j\text{best}_{S'}(k) = j\text{best}_S(k)$ is not empty);
    * if $p^*_{jh} = j\text{best}_S(h)$ then, at least, $\lambda^j(j\text{best}_S(h)) < \lambda^j(j\text{best}_S(k))$.

    Therefore, $\exists g \in V, g \neq h : \text{best}_{S^*}(j) = p^*_{jg}$ and $p^*_{jg} \neq \epsilon$.

  – $\text{best}_S(j) = p'_{jh} \neq p^*_{jg}$ from Lemma 7.2.

  In both cases, S cannot be a forwarding stable state, which again is a contradiction. □

Intuitively, the idea on which this proof is based is that, if two transitions $T' : S \to S'$, $T'' : S' \to S''$ exist such that $\text{best}_{S''}(i) \neq \text{best}_{S'}(i)$ for some $i \in V$, then the changes introduced by T' and T'' that alter the best choice at i can be "anticipated" in an appropriately made transition $T^* : S \to S^*$, the existence of which makes state S not forwarding stable.

Based on Lemma 7.3, we can now prove the following Theorem:

Theorem 7.1 *If S is a forwarding stable state, then no finite sequence of valid transitions exists that brings the system into a state S' such that $best_{S'}(i) \neq best_S(i)$ for some $i \in V$.*

Proof. The proof consists in iteratively identifying the first intermediate transition that changes a best choice and applying Lemma 7.3.

By contradiction, assume that a sequence of valid transitions

$$\mathcal{T} = \left\langle T^1 : S^0 \rightarrow S^1, T^2 : S^1 \rightarrow S^2, \ldots, T^n : S^{n-1} \rightarrow S^n \right\rangle, S = S^0, S' = S^n$$

with $n > 2$ exists such that $\text{best}_{S'}(i) \neq \text{best}_S(i)$ for some $i \in V$. Let $\left(T^k, T^{k+1}\right)$, $1 \leq k \leq n-1$ be a pair of transitions such that, for some $j \in V$: $\text{best}_{S^{k+1}}(j) \neq \text{best}_{S^{k-1}}$. Then, for Lemma 7.3, state S^{k-1} cannot be forwarding stable, which means that there must be a transition $T^* : S^{k-1} \rightarrow S^*$ such that $\text{best}_{S^*}(m) \neq \text{best}_{S^{k-1}}(m)$ for some $m \in V$. Replace $\left(T^k, T^{k+1}\right)$ in sequence $\mathcal{T}$ with T^*. By iteratively applying this argument, eventually a single transition $\overline{T} : S \rightarrow \overline{S}$ is left in $\mathcal{T}$ such that $\text{best}_{\overline{S}}(h) \neq \text{best}_S(h)$ for some $h \in V$, which means that S was not forwarding stable. □

Observe that the proof works correctly even for the case in which only one transition in the sequence $\mathcal{T}$ changes a best choice: such a transition may be simply coupled with the preceding one in order to apply Lemma 7.3.

Even if stable states are equipped with very nice properties, the system may still be unable to reach them. We say that a state S'' is *reachable* from a state S' if there exists a sequence of valid transitions

$$\left\langle T^1 : S^0 \rightarrow S^1, T^2 : S^1 \rightarrow S^2, \ldots, T^n : S^{n-1} \rightarrow S^n \right\rangle$$

such that $S^0 = S'$ and $S^n = S''$.

Figure 7.9(a) shows an example of state that cannot be reached from the initial state. This is due do the fact that the state of router 1 is not reachable unless 2 has chosen (2 0) as its best, even temporarily. Yet, if this happened, then 2 would never turn to its worse choice (2 3 0). Observe that, by simply reversing the preference function at router 2, the same state becomes reachable. In fact, in the system of Figure 7.9(b), router 3 may have announced its path (3 0) to router 2 only after the latter has propagated its choice (2 0) to 1.

7.4 Conclusions

In this Chapter we present some effective approaches to study the interaction of routing policies deployed at different Autonomous Systems.

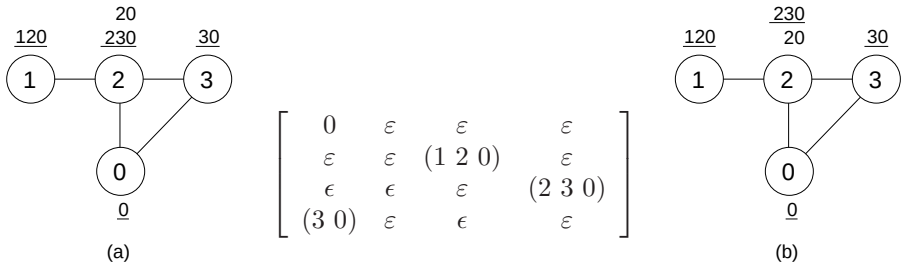


Figure 7.9: *Example of a routing state that is unreachable (a) or reachable (b) from the initial state.*

We show how to check whether an arbitrarily chosen AS-path can be traversed by the announcements of a specific BGP prefix, and also propose a methodology to check what is the preference assigned by a router to two distinct equally long AS-paths. While these are very powerful applications of the primitives described in Chapter 3, we believe there is still room for improvements. In particular, the probing algorithms in Sections 7.1 and 7.2 could still be tuned to minimize the number of necessary announcements and to provide more accurate results in the case in which the investigated paths do not end at a collector-peer.

We also introduce a model to study interdomain routing instabilities which is a variant of the well known Stable Paths Problem formalism. Our model takes into account all the possible timings of message exchanges among routers and easily accommodates two different notions of stability. We also prove interesting properties that relate the two kinds of stable state and introduce the concept of reachability between states. Our model is still incomplete in that it lacks conclusions about the way to identify stable states in a system and to determine whether they are reachable. Among these yet unexplored conclusions we could search for properties that allow to simplify the network instance under consideration in order to make the model scale better to large Internet-like instances. However, we believe this approach provides valuable and, to a certain extent, novel instruments to study routing instabilities, and is therefore worth being kept into account within future research to propose extensions and to apply it to case studies from the real world.

Part IV

Experimenting Routing by Emulation

Chapter 8

Emulation of Computer Networks with Netkit

> The one who could repeat the summer day
> Were greater than itself, though he
> Minutest of mankind might be.
>
> The One who could repeat the Summer Day
> Emily Dickinson

Testing configurations is a common need both for network administrators and for computer scientists interested in networking. The first can take advantage of a testing phase for preventing trouble from happening on a live network, while the latter can exploit test results in order to validate theoretical models with practical experimentation. Ideally, testing should take place under the very same conditions in which the configuration is to be eventually deployed. However, this often means injecting artificially generated, potentially harmful traffic into a live network, which may cause damage to it.

An effective alternative to live testing consists in implementing the network configuration of interest inside a safe, isolated software environment which closely reproduces the real target setting. Such environments are usually available in two flavors:

- **Simulation environments** allow to compute the outcome of running a set of network devices on a complex network by carrying into effect a black-boxed set of procedures that are specific to the simulator. With the network as an

input and the outcome (possibly a network state) as an output, simulators do not necessarily reproduce the same sequence of events that would take place in the real system, but rather apply an internal set of transformation routines that brings the network to a final state that is as close as possible to the one the real system would evolve to.

As this approach can be optimized in performance, the simulated network can typically scale well in size. The drawback is that the simulated devices may have limited functionalities and their behavior may not closely resemble that of real world devices. The following is a list of some of the most representative network simulators.

C-BGP [17, 20, 18] is a routing solver that computes the outcome of the decision process of the BGP protocol [252] on a topology that may consist of several routers. While it does provide complete support for policy routing (see Part III), session establishment and routing timers are not modelled.

NS-2 [227] is an object oriented network simulator supporting both C++ and OTcl. It provides a very comprehensive interface for describing network models which is based on an event-driven approach. Events are fired by a user selectable scheduler and are managed by a user defined handler. Nodes in the simulated network can use a variety of routing agents to forward packets over links that simulate different types of media. Traffic generators can be configured to inject the desired packet patterns into the network. Thanks to a modular structure, a number of extensions is continuously being provided by researchers. NS-2 is typically used in conjunction with topology generators which allow to quickly build large realistic networks.

SSFNet [235] provides open source implementations of several network protocols and network devices based on a standardized API. A network described in the Domain Modeling Language (DML) can be submitted to the simulation kernel, which computes its evolution over a time interval with a fixed discrete time resolution. Measurements can be performed on the nodes to observe their behavior during the simulation (for example, the status of packet queues) and, based on them, an animated view of the network displaying network states or packet transmissions that are of interest for the user can be obtained.

- **Emulation environments** aim at closely reproducing the features and behavior of real world devices. For this reason, they often consist of a software or hardware platform that allows to run the same pieces of software that would be used on real devices. Differently from simulation systems, in an

emulator the network being tested undergoes the same packet exchanges and state changes that would occur in real world.

The real advantage of this approach comes out when the emulator itself is a software piece, as this allows much higher flexibility in carrying out network tests. Since emulation makes use of real routing software, every aspect of the network can be influenced and monitored like it could be in a real network. While this ensures very high accuracy, the computational resources needed to run an emulated device are typically higher than those available in the device itself. Hence, the performance of an emulated device is, in general, lower than that of the real one, and this often poses limits on the scalability of the size of the emulated network.

Since emulation environments are the focus of this Chapter, examples of them are presented in the following Sections.

This Chapter introduces to the fundamental concepts of emulation environments and presents Netkit [35], a lightweight product based on open source software that enables experimenting networking on a personal computer. After describing the architecture and some internals of Netkit, we show how it is possible to easily prepare and run complex network experiences, which can also be facilely packaged and redistributed. An example case study is also presented to show the capabilities of Netkit.

8.1 An Overview of Emulation Environments

An *emulator* is a software or hardware environment that aims at closely reproducing the features and behavior of a real device or system, usually making it possible to use software products unaltered inside a system they were not designed for.

There are a lot of emulation products available, which can be distinguished on the basis of the emulation technique adopted, of the type of device they emulate, and of the license with which they are distributed. This Section attempts to provide a taxonomy of a number of emulators, with the twofold purpose of outlining the landscape of available alternatives and of pointing out those products that are more network oriented.

Table 8.1 shows a fairly comprehensive list of the existing emulation products. The proposed classification coordinates have the following meaning:

- The **scale** ranges from small to large, and describes the number of virtual entities (hosts, routers, switches, or whatever else) each emulator allows

	Scale	Emulation type	Emulated device	License	Link
Bochs	Small	Full emulation	IA-32 x86	GPL	[204]
Cooperative Linux	Medium	Kernel port	Linux box	Free	[206]
CrossOver	Medium	Compatibility layer	Windows APIs	Commercial	[245]
DosBox	Small	Full emulation	x86 DOS box	GPL	[208]
DosEMU	Small	Compatibility layer	DOS box	GPL	[209]
Einar	Large	Router emulation	Quagga based router	GPL	[210]
Emulab	Large	Testbed	—	Project based	[199]
FAUmachine	Medium	User-mode kernel	x86 box	GPL	[47]
IMUNES	Medium	Virtual image	Linux box	Free	[50]
KVM	Medium	Native virtualization	x86 box	GPL	[221]
MLN	Medium	Paravirtualization/User-mode kernel	Linux box	Free	[222]
Modelnet	Large	Testbed	Linux box	GPL/BSD	[48]
NCTUns	Medium	Simulation	Host/Router	Free	[182]
Netkit	Medium	User-mode kernel	Linux box	GPL	[35]
Parallels	Medium	Full virtualization	x86 box	Commercial	[156]
PearPC	Small	Full emulation	PowerPC box	GPL	[228]
Planetlab	Large	Overlay network	—	Membership	[161]
Plex86	Medium	User-mode kernel	Linux box	Free	[231]
Q	Small	Full virtualization	x86 box	Free	[232]
QEMU	Small	Full virtualization	x86 box	GPL	[51]
SVISTA	Small	Full virtualization	x86 box	Commercial	[236]
UML	Medium	User-mode kernel	Linux box	GPL	[238]
UMLMON	Medium	User-mode kernel	Linux box	GPL	[62]
vBET	Medium	User-mode kernel	Linux box	N/A	[49]
VDE	Large	Overlay network	—	GPL	[166]
VINI	Large	User-mode kernel	Linux box	Membership	[241]
VirtualBox	Small	Full virtualization	x86 box	GPL/Commercial	[80]
Virtual PC	Small	Full virtualization	x86 box	Free	[146]
Virtuozzo	Small	Full virtualization	x86 box	Commercial	[242]
VMware	Small	Full virtualization	x86 box	Commercial	[243]
VNUML	Medium	User-mode kernel	Linux box	GPL	[186]
Win4Lin	Medium	Full virtualization	x86 box	Commercial	[244]
Wine	Medium	Compatibility layer	Windows APIs	GLPL	[245]
Xen	Medium	Paravirtualization	x86 box	GPL/Commercial	[149]

Table 8.1: *A taxonomy of emulation-related products.*

to start on the platform it is intended to be used on (typically a standard workstation). A *small* scale emulator is usually conceived for running very few instances of virtual machines, as their resource requirements may be rather high. A *large* scale emulator is usually designed to run on a distributed architecture (possibly a cluster of geographically distributed workstations connected by an overlay network), which allows to perform arbitrarily wide-ranging experiments. *Medium* scale emulators typically allow to run around tens of virtual machines on a single workstation.

- **Emulation type** specifies the technique used for virtualizing resources. *Full virtualization* indicates that the emulated entity is a full-fledged system consisting of system buses, CPU, memory, disk, and other devices, and that optimization techniques are used to improve the performance of the emulation. Among these techniques *dynamic translation* is often used, which consists in translating blocks of binary code being executed in the emulated machine into instructions for the real host, and in caching the translated pieces of code for future execution.
  Full emulation adopts a complementary approach, in which every instruction of the emulated CPU is implemented as an entire function or procedure in the emulator. While this ensures compatibility and makes it easier to debug the code of the emulator, performance is severely impacted by this technique, and a high-end workstation is usually needed to achieve nearly native speed emulation.
  Native virtualization takes place whenever the emulator takes advantage of extensions available on recent families of processors (Intel®VT [81], AMD-V™ [3]), which allow a more effective distribution of resources between the emulated machine and the host it runs on, thus achieving much better performance.
  In a *paravirtualization* environment each virtual entity is presented a special hardware abstraction layer. Virtual machines must run slightly modified versions of the standard operating systems, so that system calls are submitted to this abstraction layer (called *hypervisor* or *virtual machine monitor*) instead of the host operating system.
  Emulation environments using a *virtual image* are based on virtualization extensions to the FreeBSD kernel [212] which allow to maintain multiple independent network stack instances within a single operating system kernel.
  Some products exploit a *user-mode kernel*, often called *User-Mode Linux* (UML) [238, 92, 91], which is a slightly modified version of a standard

Linux kernel that is compiled to run as a userspace process. An instance of User-Mode Linux uses its own filesystem image and allocates a subset of the memory available on the hosting machine. A UML kernel can start and schedule processes on its own and has its own virtual memory manager as well as every other kernel subsystem. Device drivers are suitably rewritten so that UML can provide some support for virtualized hardware (disks, network interfaces, consoles). Differently from other emulation systems, User-Mode Linux does not directly interface with the hardware but achieves virtualization based on the system calls interface provided by the standard kernel.
A *compatibility layer* is not a real emulator, but rather a porting of a system call interface and a set of libraries on a different platform. In this case there is no hardware being emulated. However, programs that solely perform standard system calls and invoke library functions can run unmodified because they are presented the same software interface as that of their native system.
Testbeds and *overlay networks* are large scale environments consisting of tens or hundreds of servers that can be geographically distributed (in which case they are often connected so as to form an overlay network). Such large scale architectures can be typically accessed by research institutions to perform controlled experiments on a realistic setting.
Last, *simulation* has already been defined and described in the introduction of this Chapter.

- The **emulated device** specifies the machine whose features are reproduced by the emulator. In most cases it is a standard PC which, by running suitable pieces of software, can be turned to a router, switch, or other network device.

- **License** specifies the license agreement under which the emulator is being distributed.

Not all the products listed in Table 8.1 are tailored for performing networking experiments. Some of them are general purpose, often small scale emulators that allow to run entire operating systems. We are more interested in the environments that provide configuration capabilities and tools to ease building and running emulated networks.

VDE and Xen are components that are often used in emulation environments. Virtual Distributed Ethernet (VDE) [166, 169, 168] is a set of tools to create and manage a virtual network that can be spawned over a set of arbi-

trarily distributed physical computers (see [167]). VDE can be used to handle tunnels that separate actual connectivity from the topology established by VDE virtual cables, thus providing with the ability to transparently distribute local network experiences on different nodes. Xen [149, 79, 78, 77, 107, 159] is a *virtual machine monitor* or *hypervisor*, that is, a software that provides an abstraction of hardware resources. It consists of a kernel patch and some userspace tools. Because it directly interfaces with hardware resources, using a Xen enabled kernel allows to run virtual machines (also called *domains*) with very high performance levels. Operating systems cannot run unmodified inside Xen domains, as the system calls they make must be mapped to software traps to the hypervisor. Yet, there are plans to extend Xen to support virtualization technologies found on recent processors (Intel®VT [81] and, in the future, AMD-V^{TM} [3]), which would provide with the ability to run unmodified software.

Einar [210] is a live CD router emulator based on the Xen hypervisor [149] and developed by a team of 5 students at the Stockholm KTH Royal Institute of Technology. It provides a wizard interface to quickly set up and run experiences consisting of several switches, routers, and computers, possibly running some services. Each virtual machine is implemented as a Xen domain, and it possible to configure bandwidth and delay limits on network interfaces. The traffic exchanged among the virtual machines can be investigated by using the Ethereal network sniffer. As it runs from a live CD, the product requires no installation.

Emulab, Modelnet, PlanetLab, and VINI share similar purposes and architecture. They all are large scale testbeds which allow to run experiments involving a set of geographically distributed nodes. The University of Utah makes available to researchers Emulab [199, 87, 87], a cluster of networked high-end workstations that can be configured to run fully customizable tests. Faculty or senior staff members coordinating an experiment have fine grained control on the software running on workstation nodes, from the operating system to user level applications, and get exclusive administrator level access during the time interval they are assigned. This allows to configure each node to act as an arbitrary network device. Modelnet [48, 162, 86, 7] is a software emulator developed at the University of California, San Diego, that allows to build distributed network experiences running on an Internet-like environment. PlanetLab [161, 2, 118, 117, 183] is an overlay testbed conceived to allow researchers to perform controlled experiments that would benefit from running on geographically distributed nodes. The PlanetLab network consists of more than 350 nodes at the time this Chapter is being written, and is managed by a

consortium of academic, industrial, and government institutions. Experiments are initiated on the basis of an Acceptable Use Policy [230] that defines the rules for determining whether they are considered as appropriate. VINI [241, 10] is a virtual network infrastructure running on about 15 PlanetLab nodes. It exploits User-Mode Linux [238] to provide a realistic test environment while keeping the network conditions under control.

IMUNES [50] is a software emulator that takes advantage of an extension of the FreeBSD kernel providing with the ability of running multiple independent network stack instances on a single operating system kernel [138, 212]. A graphical topology editor allows to quickly prepare experiments consisting of hubs, switches, routers, and hosts. Intermediate systems can arbitrarily use Quagga [233] or XORP [82] as routing software. The product is available in the form of a live CD, which takes off the burden of having to deploy a modified kernel on the host machine.

MLN [222] is a Perl script that can be used to quickly create a network consisting of Xen [149] and User-Mode Linux [238] based virtual machines. By means of an ad-hoc language (MLN), it is possible to specify the configuration of each device participating in the virtual network, including the emulation type (Xen or UML), the filesystem being used, the amount of available memory, the startup commands to be executed, and, of course, the topology of the network.

NCTUns [182, 176, 181, 177, 1, 178] is rather a simulator than an emulator. After defining a topology with the integrated graphical editor, it is possible to launch traffic generators and sinks at a given time during the simulation. The resulting traffic flows can then be investigated either by looking at sniffer traces or by displaying an easy to read animation of the edges on the graphical topological map.

Netkit, UMLMON, and VNUML are all medium scale software emulators that utilize a User-Mode Linux [238] kernel to run the emulated network experiences. Netkit is extensively described in the following of this Chapter. UMLMON [62] is a solution for managing a set of User-Mode Linux virtual machines. It comes in the form of a daemon written in Objective Caml and acts as a supervisor that allows to configure and run UML instances by using a Remote Procedure Call interface. Interaction between the end user and the daemon is possible by means of a command line or a web based interface. VNUML [186, 54, 53] consists of an XML based language and an interpreter that can be used to describe and run an emulated network of User-Mode Linux virtual machines. VNUML developers also propose an interesting set of ready to use examples that implement some typical networking case studies.

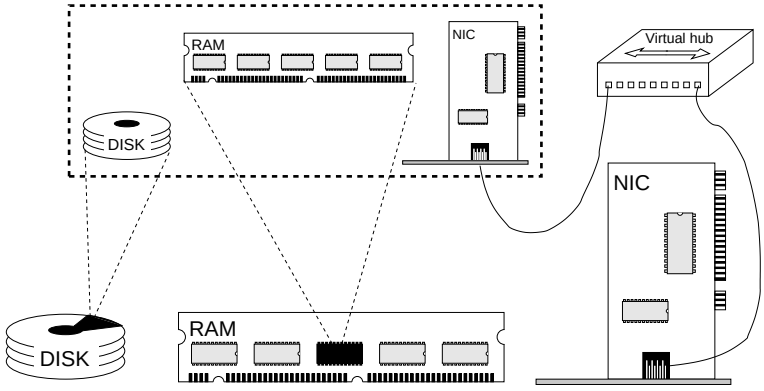


Figure 8.1: *Resources of a Netkit virtual machine. Entities enclosed in the thick dashed box are virtualized, while all the others correspond to devices or processes on the host machine.*

8.2 The Architecture of Netkit

In the rest of this Chapter we describe in detail Netkit [35], a lightweight network emulator developed by people at the University of Roma Tre [240], including myself. Netkit consists of two main components: a set of scripts and tools that ease setting up complex network scenarios consisting of several virtual devices, and a set of ready-to-use virtual experiences that can be used to analyze typical case studies. Netkit is conceived for easy installation and usage and does not require administrative privileges for either one of these operations.

As time goes on, old bugs are fixed and new features are introduced into Netkit. This Chapter refers to the version of Netkit available at the time of writing, which is 2.4, filesystem F2.2, kernel K2.2 (User-Mode Linux kernel 2.6.11.7). Further information and documentation about Netkit can be found in the Netkit `man` pages, in the on-line tutorials [36], and in Netkit related publications [140, 37].

The emulation approach adopted in Netkit is simple. Basically, every device that makes up a network is implemented inside Netkit as a virtual machine. Each virtual machine owns a set of virtual resources that are mapped to portions of the corresponding resources on the host. Figure 8.1 shows how this mapping takes place. Virtual machines are equipped with a disk, whose raw

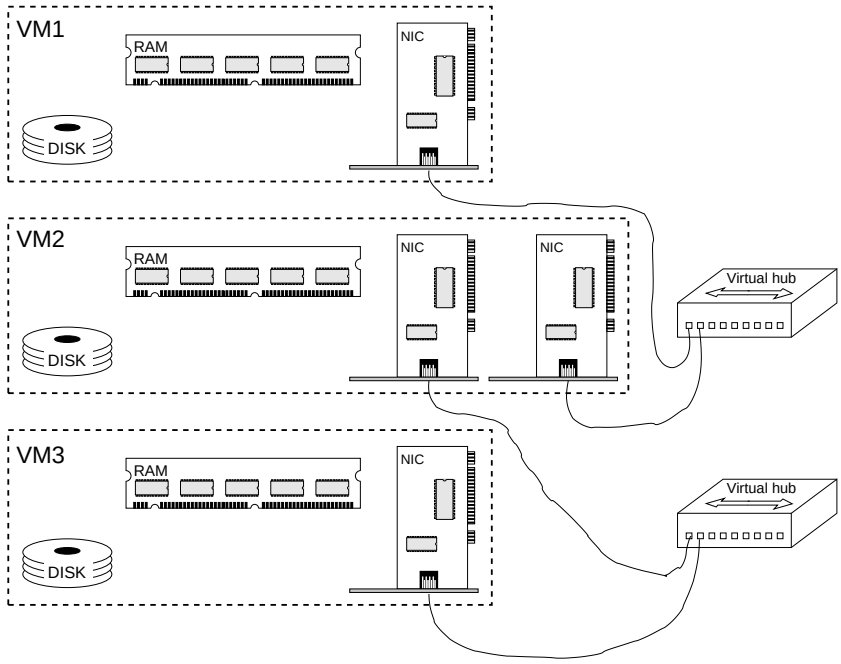


Figure 8.2: *Sample Netkit emulated network.*

image is a file in the host machine; they have their own memory region, whose size can be established upon startup; and they can be configured with an arbitrary number of virtual network interfaces which are connected to a virtual hub. The thick dashed box in Figure 8.1 encloses virtualized resources, while everything outside of that box is a device or process on the host. It is possible to observe that the virtual hub lives on the real host: indeed, it is a special process that replicates packets on all the connected interfaces. If requested, the virtual hub can be connected to a network interface of the host machine, so that a virtual machine can reach an external network such as the Internet.

To make a clear distinction between an emulated device and the real machine it is running on, in the following we label Netkit virtual machines and the software they run as *guest*, and we refer to the real machine and the software it runs as *host*.

Netkit virtual machines can be networked by means of the virtual hub. In

practice, a hub works as a sort of cable that connects multiple guests. Notice that a guest must always connect to a virtual hub, and cannot be directly connected to another guest. Figure 8.2 shows an example of how a typical Netkit network looks like. `VM1`, `VM2`, and `VM3` are virtual machines and they make up a simple topology consisting of two collision domains: one including `VM1` and `VM2` and the other including `VM2` and `VM3`. In this topology `VM2` operates as a virtual switch that forwards traffic only on the ports connected to the LAN segment containing the destination host. For this purpose, `VM2` has been equipped with two network interfaces.

The example in Figure 8.2 also introduces another principle of the Netkit emulation approach: the functionalities of an emulated device depend on the software installed in the virtual machine that implements it. For instance, `VM2` can be turned to a switch by running standard ethernet bridge administration software such as `brctl`.

Netkit virtual machines are based on the User-Mode Linux kernel [238], which is described in more detail in Section 8.2.1. Starting a virtual machine means starting a UML instance, which often requires dealing with somewhat complex command line arguments. For this reason Netkit supports straightforward configuration and management of virtual machines by means of an intuitive interface consisting of several scripts.

Figure 8.3 synthetically describes the architecture of Netkit, consisting of the blocks inside the dashed box. Each block represents a piece of software that runs on top of the ones beneath and is controlled by the tools on its left. Virtual machines are UML instances that directly run on the host kernel and are managed by a set of commands whose names are `l`-prefixed (*ltools*) and `v`-prefixed (*vtools*). Virtual machines can run routing software, as well as other tools. Virtual hubs are implemented as processes running on the host kernel.

It is interesting to observe that there is no need for the end user to directly interact with UML kernels or virtual hubs. On the other hand, both the `ltools` and the `vtools` are accessible to the user. The difference between the two is that the ltools provide a higher level interface to virtual machines and therefore exploit the `vtools` in order to implement their functionalities.

Because of its architecture, taking advantage of Netkit can be beneficial in several contexts. The most common application is within didactics: as it gives students the feeling of what is actually going on inside a network, Netkit has been successfully used to teach networking protocols within University level degree courses. This is further supported by the fact that Netkit comes with a set of ready to use network experiences that implement case studies spanning from routing protocols (TCP, RIP, BGP, etc.) to application level services

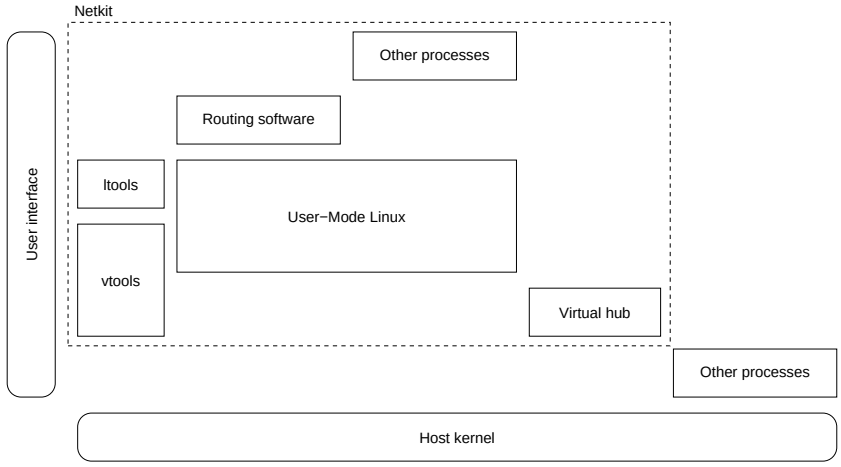


Figure 8.3: *Architecture of Netkit. Each software runs on top of the ones beneath and is controlled by the tools on its left.*

(DNS, e-mail, etc.) [36]. Another valuable application consists in preparing virtual networks that act as sandboxes for safe debugging purposes: this spares the need to perform potentially harmful tests on a live network. The ability to carry out experiments in a safe environment, combined with the easiness of maintaining a one-to-one reproduction of a real network, also comes handy for testing a configuration before deploying it. For example, Netkit has been used for helping in determining the OSPF weights to be assigned within portions of the GARR Italian Academic Research Network [213].

With respect to the other available network emulators, Netkit has the advantage of being lightweight and easy to install and run. It is possible to launch a complex network experience consisting of 200 virtual machines in about 30 minutes on a typical workstation (Pentium IV 3.2GHz 2MB cache, 2GB RAM)[1]. Netkit fully runs in userspace and has no dependencies on other software pieces. It natively uses state of the art routing software [233, 108] and comes with most of the commonly used networking tools (sniffers, servers, etc.). Should it be needed, it is possible to also install additional packets inside virtual machines. As detailed in Section 8.3, emulated network experiences

[1]This test has been performed on a SKAS enabled host kernel. For more details about this, see Section 8.2.1

can be easily redistributed in a preconfigured, ready to use, and automatically starting up package without the need to carry any heavy filesystem image.

On the other side, Netkit only works on top of Linux hosts and provides emulated Linux virtual machines. While preliminary versions for the Windows OS are being released, and a live CD [179] is also available to make it easier to play around with Netkit, the second restriction is inherent in Netkit and cannot therefore be overcome. Also, at present Netkit provides a somewhat fixed implementation of the link-level network layer which does not provide support for the emulation of physical link properties (latency, packet loss, reordering, etc.) or the behavior of wireless mobile stations. However, there are plans to implement some of these features in future releases. Moreover, being an emulator, Netkit is not suited for reproducing real time performance of network protocols and services.

There are a couple of projects that propose XML based languages for describing networks and can therefore be exploited to delineate and implement emulated scenarios. NetML [38, 84] is a project carried on by the same group that maintains Netkit, and for this reason it is strongly integrated with the emulator. Besides an XML Schema, NetML consists of a set of tools that allow to turn a vendor independent description of a network into configuration statements for specific routing software (Cisco, Juniper, Zebra) or, optionally, a package implementing a Netkit network. NDL [197, 95, 94, 93] is a language based on the Resource Description Framework (RDF) and a set of tools that ease producing network descriptions. While NetML is conceived to describe small as well as broad networks, NDL is more suitable for large scenarios, as it also takes into account the geographical location of network nodes. NDL tools also allow to automatically sketch visual representations of a network.

8.2.1 User-Mode Linux: a Kernel in the Kernel

In an emulation environment virtual machines have nearly the same characteristics of a real host, including their own operating system running on top of their own kernel. Netkit exploits User-Mode Linux as kernel for the virtual machines. User-Mode Linux is widely used by kernel hackers, who are doing filesystem and memory management development and debugging, as well as by hardware developers, who are prototyping new types of device in software. It also meets the interests of the security community, as it fits well the creation of jails and honeypots, and is often employed by the hosting industry to run virtual servers [120, 110]. The fundamentals of UML are illustrated by its de-

signer Jeff Dike in some publications [92, 88] which the following description is based on.

User-Mode Linux [238, 92, 52, 109, 91, 14, 90, 89] is a port of the standard Linux kernel [220] which is designed to run as a userspace process. Being a kernel in itself, UML comes with its own kernel subsystems, including scheduler, memory manager, filesystem, network, and devices. In this sense an instance of UML provides a virtualized environment in which everything (processes, memory, filesystem, etc.) is controlled by itself instead of the host kernel. In practice, UML appears as a userspace process on the hosting machine and acts as a kernel for its own processes.

Virtual machine settings can be passed to UML via a command line interface. While this is an effective way of specifying configuration parameters, it is often the case that users interested in just emulating networks are not willing to deal with complex kernel invocation commands. For this reason, Netkit provides a higher level user interface to UML which is described in more detail in Section 8.2.4.

There is an important difference in the approach adopted by full emulation/virtualization products and that adopted in Netkit. Full emulation products usually attain virtualization by directly interfacing with the host hardware, and provide an abstraction layer implementing an architecture that may also be different from the one of the host they are running on. In the case of UML, virtualization takes place within the host kernel rather than at the hardware layer. In other words, UML provides simulated hardware constructed on the basis of services provided by the host kernel. Essentially, UML is a port of the Linux kernel to the Linux system call interface rather than to a specific hardware interface. User space code simply runs natively (no emulation), while processes in kernel mode see a special environment which limits access to host resources. This makes the emulation faster and more responsive, and is the reason why Netkit is considered a lightweight emulator. The drawback of this approach is that it only allows to run emulated Linux boxes.

Basically, what UML does is to provide virtualization for system calls. Normally, a process in the virtual machine doing a system call is trapped directly into the host kernel. Instead, UML intercepts system calls so that they are run in virtual kernel mode. This is implemented by using threads and the `ptrace` system call. `ptrace` allows one process to control the execution of another, as well as change its core image and be notified when it receives a signal. In UML a special thread called *tracing thread* makes use of `ptrace` to intercept and dispatch to the virtual kernel the system calls issued by processes in the virtual machine. Other traps caused by pieces of hardware (clock, I/O devices,

Host console

```
max@foo:~$ ps -o pid,command
  PID COMMAND
12959 /bin/bash
12987 /bin/sh /usr/local/netkit/netkit2/bin/vstart pc1
12990 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(tracing thread)]
12991 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
12998 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13000 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13002 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13004 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13006 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13007 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13009 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13011 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13013 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13015 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13016 xterm -T Virtual Console #0 (pc1) -e port-helper -uml-socket
                                                    /tmp/xterm-pipexvvhYG
13018 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [(kernel thread)]
13021 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [/sbin/init]
13696 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [/sbin/klogd]
13792 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [/sbin/syslogd]
13834 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [-bash]
13836 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [-bash]
13868 /usr/local/netkit/netkit2/kernel/netkit-kernel (pc1) [tcpdump]
13895 ps -o pid,command
```

Figure 8.4: *Processes and threads in a UML virtual machine are implemented as processes on the host.*

etc.) are implemented in UML by using signals.

For each process or thread running inside the UML virtual machine, the tracing thread creates a new process on the host. Figure 8.4 shows a process listing on the host while `tcpdump` is running inside a virtual machine. It can be easily seen that `tcpdump` appears as a process entry (pid 13868) on the host.

After performing preliminary initializations, UML mounts a virtual disk device provided by the user and boots the Linux distribution it finds inside it. Virtual disks are managed by a User-mode Block Device (UBD) driver, which uses a standard file (called *backing file*) on the host filesystem as storage area. The backing file can be filled with Linux software in a way that is very similar to what would happen on a real host. Indeed, the backing file can be made available as a loopback device on the host machine by using the `losetup`

command. Once done, it can be initialized by using `mkfs` and populated by using tools such as `debootstrap`. In order to support hassle free usage, Netkit comes with a ready to use filesystem containing most state of the art networking tools. For more details, see Section 8.2.3.

UML supports the emulation of an arbitrary number of network interfaces. This aspect is discussed in more detail in Section 8.2.2.

Being a lightweight environment, it is possible to implement complex setups consisting of several instances of UML based virtual machines. Since each virtual machine writes on its own filesystem, this potentially implies using several backing files, which size is often not negligible (hundreds of megabytes). However, the ubd block driver is able to support sharing of a filesystem among different virtual machines. This is achieved by writing changes to the backing file into a different file, a technique that is also known as *Copy-On-Write* (COW). Therefore, a typical setup of a complex emulated scenario consists of a single large backing file containing a model filesystem and several small (typically less than 10MB) COW files that store the changes to the model filesystem. Thus, filesystem information for a virtual machine can only be reconstructed based on both its own COW file and the backing file. Each COW file can only be used together with the backing file it was created from. However, by using the UML utility `uml_moo` [239, 155], it is also possible to merge the two and get a standalone backing file that contains all the filesystem information for that virtual machine. COW files are implemented as *sparse files*. A sparse file is one which efficiently uses the filesystem by allocating space only when data is actually written to the file. Figure 8.5 shows an example of the space allocation strategy for a sparse file. Apparently, the size of `pc1.disk` is 602MB (output of `ls`), but the actual disk space consumption for this file is of 620KB (first column of the output of `ls` and output of `du`). This means that `pc1.disk` has a lot of contiguous empty regions which are not actually written to disk: this is often the case for COW files, as the filesystem of an only just started virtual machine only slightly strays from the initial status.

Once a UML virtual machine has started, it is possible to interact with it by means of a terminal interface. The interface can be attached to arbitrary file descriptors, `pty` pseudo terminal devices, or to a user space application like `xterm` or a server like `telnetd`. This is possible thanks to the `port-helper` tool, that is part of the UML utilities [239, 155]. Essentially, `port-helper` uses a UNIX socket to pass to the UML kernel a file descriptor obtained from an application (like `xterm`) and used to perform input/output. This allows UML to directly interface with the application.

UML instances can be managed from the host machine by means of the

Host console

```
max@foo:~$ ls -ls --block-size=1 pc1.disk
634880 -rw-r--r-- 1 max max 630358016 2007-01-19 18:24 pc1.disk
max@foo:~$ du --block-size=1 pc1.disk
634880  pc1.disk
```

Figure 8.5: *Size and actual disk space consumption of sparse files. |pc1.disk— takes about 620KB of disk space, but is apparently as large as 602MB.*

`uml_mconsole` utility [239, 155], which allows to halt or reboot a virtual machine, to send it magic SysRq sequences, to configure emulated devices on the fly, and to pause or continue its execution. `uml_mconsole` is used in the Netkit scripts to manage running virtual machines.

In the past UML used to be available in the form of a patch to a standard Linux kernel [238, 220]. Most recent kernels already include user-mode code, thus a UML kernel can be simply built by specifying `um` as target architecture while compiling a vanilla kernel. Starting from 2002, Paolo Giarrusso [154] maintains a set of patches called SKAS that can be optionally applied to the host kernel to change the way UML behaves. The patches have beneficial effects in terms of both security and performance. Essentially, when using the standard technique with tracing thread, the address space of each virtual machine contains the image of the UML kernel, and can have write access to it. Since UML runs on the host, this also means gaining access to the host machine, which affects security. On the other hand, UML takes advantage of signals in order to implement system call dispatching, and this affects performance. A SKAS (*Separate Kernel Address Space*) enabled host kernel allows to overcome both these issues. As for security, SKAS makes UML run in a different address space from the processes it controls. In turn, this limits the number of signals to deliver, as fewer context switches are needed within this rearranged setting, and this improves performance. Benchmarks have shown that starting 100 virtual machines with the default configuration (8MB, minimal services running) on a Pentium IV 3.2GHz with 2MB of cache and 2GB of RAM takes about 30 minutes with a plain 2.6.16.16 host kernel. The time required to start the same set of virtual machines on the same workstation running a 2.6.16.16 SKAS enabled kernel dramatically reduces to about 10 minutes, and starting 100 more virtual machines in this setting takes less than 30 more minutes, thus having a scenario consisting of 200 devices running on a single workstation within less

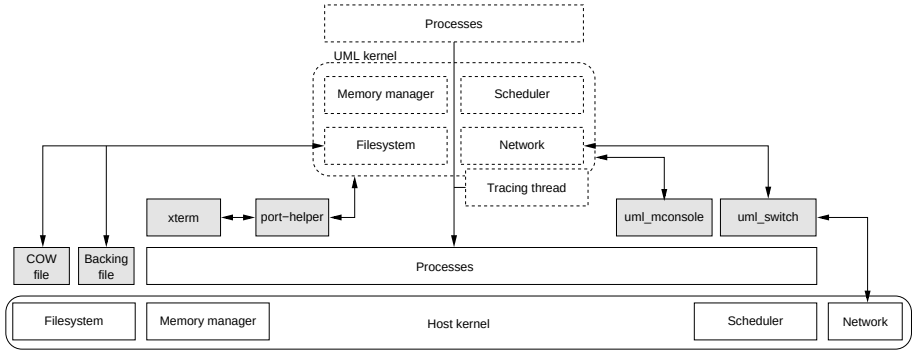


Figure 8.6: *Architecture of a User-Mode Linux kernel. Boxes represent kernel entities (subsystems, interfaces, or processes). Dashed boxes represent virtualized resources, while gray filled ones are instantiations of kernel entities (processes or files).*

than 40 minutes.[2] Even though the SKAS patch improves security and boosts performance, Netkit still scales rather well without the need to replace the host kernel.

Figure 8.6 provides a sketch of the architecture of the UML kernel described in this Section. Boxes represent kernel entities (subsystems, interfaces, or processes). Dashed boxes represent virtualized resources, while gray filled ones are instantiations of kernel entities (processes or files). A virtual machine is a set of processes. The tracing thread takes care of mapping processes inside the virtual machine to real processes on the host. The virtual machine kernel has its own subsystems that are independent from those of the host kernel. The virtual machine filesystem is implemented in terms of files on the host. Other processes running on the host provide a user interface to the virtual machine (`xterm`) or are used for its management (`uml_mconsole`). The `uml_switch` is a user space utility to support networking whose functionalities are explained in Section 8.2.2.

[2]In order not to overload the host with several virtual machines booting up at the same time, Netkit adopts an optimization technique that ensures that no more than a fixed number of virtual machines are booting simultaneously at any given time. For the case of these performance tests the number has been always set to 3. For more details about this feature, see Section 8.2.4.

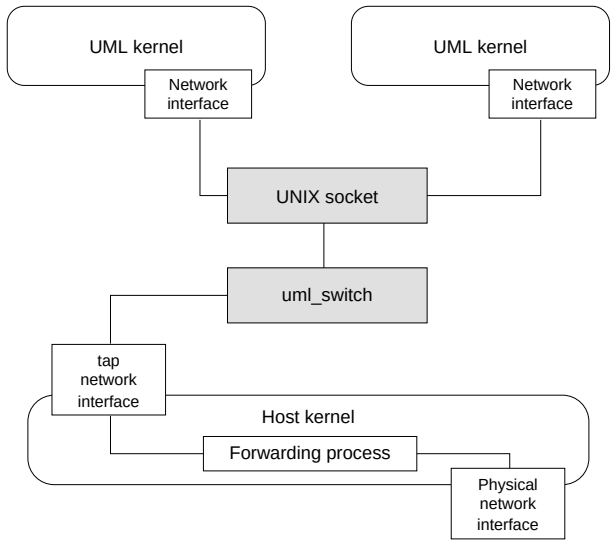


Figure 8.7: *This diagram shows how Netkit virtual machines are networked, possibly with a connection to an external network.*

8.2.2 Networking Support in Netkit

Communication between different virtual machines is made possible in Netkit by emulated networking. Figure 8.7 describes graphically how this emulation takes place.

UML allows to configure virtual machines with an arbitrary number of network interfaces. By using appropriate UML command line arguments, these interfaces can be attached to a `uml_switch` process [239, 155] running on the host, which simulates the behavior of a network switch or hub. In this way, different virtual machines attached to the same switch can exchange data with each other.

More specifically, UML virtual network interfaces can be attached to a UNIX socket. In turn, a `uml_switch` can be attached to the same socket and forward packets among the virtual machines that are connected to that socket. Netkit scripts take care of automatically setting up `uml_switch`es according to user's needs.

From the point of view of the network stack, Netkit provides implementations of the ISO-OSI layers as described in Figure 8.8.

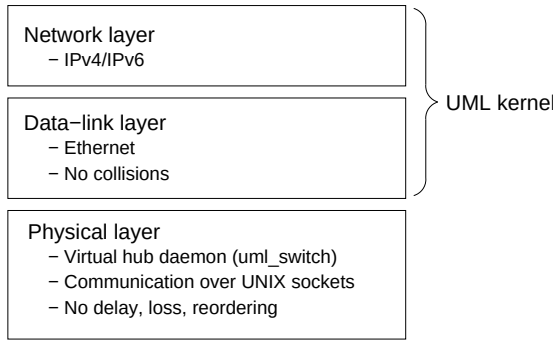


Figure 8.8: *The emulated network stack in Netkit.*

- The physical layer is implemented by a set of `uml_switch` processes running on the host. They are configured to behave as hubs, and packets are forwarded to interfaces of other virtual machines by using UNIX sockets. For this reason, in this Chapter we also refer to the `uml_switch` as *virtual hub*. At present this mechanism does not provide support for simulating delay, packet loss, and reordering.
- The data link layer supports the Ethernet protocol, but collisions cannot happen because the `uml_switch` avoids them. Unless differently specified, emulated network interfaces are assigned an automatically generated MAC address.
- The network layer supports both IPv4 and IPv6 by means of kernel code and user space utilities.
- What happens on upper layers is up to the specific software being run inside virtual machines. For example, running a web server would introduce support to HTTP.

Notice that layers from data-link through transport are (at least partly) implemented inside the UML kernel. Therefore, changing the kernel results in making new implementations and features available.

The configuration of network interfaces in Netkit is straightforward thanks to the existence of scripts set up for the purpose. In order to slightly abstract from the technicalities of how networking is implemented, Netkit presents `uml_switch`es as virtual collision domains. Each virtual network interface must

be attached to a collision domain that is identified by an arbitrary name. Therefore, connecting virtual machines is simply a matter of attaching their interfaces on the same collision domain.

For example, the following command line starts up a virtual machine named `foo` with a single network interface attached to collision domain `COLL-DOM-A`.

```
vstart foo --eth0=COLL-DOM-A
```

IP addresses for the interfaces can then be configured in the usual way by using `ifconfig` inside virtual machines.

Configuring a virtual machine to reach an external network requires setting up a `TAP` device. `TUN/TAP` is a device driver inside the Linux kernel that sets up a special network interface connected to a user space application instead of a physical medium. Packets sent to a `TUN/TAP` interface are written to the underlying application, while information generated by the application appears as it were received from the interface. The only difference between `TUN` and `TAP` is that the former expects the application to handle IP packets, while the latter allows to directly transfer Ethernet frames.

As shown in Figure 8.7, the setup for an external connection involves configuring a `TAP` device on the host and using an `uml_switch` as application that sends and receives data from it. Since the `uml_switch` can still be attached to virtual machines via UNIX sockets, this allows to connect a virtual network segment to a real network one. Depending on the specific configuration set up on the host, the two segments can be bridged or routed. In order to prevent IP addresses used for experiments from leaking on a real network, Netkit enables routing instead of bridging, and uses Masquerading to hide addresses assigned to emulated network interfaces. Masquerading is a variant of port-based NAT in which outgoing packets are mangled to appear as originated from the interface they are sent through; incoming response packets are forwarded to the interface that had actually initiated the network transfer. Masquerading can be enabled in Linux by taking advantage of the netfilter framework [224] inside the kernel. This can be achieved by using `iptables` to configure an appropriate rule with target `MASQUERADE` in the `POSTROUTING` chain of the `nat` table.

Netkit scripts take care of automatically setting interfaces connected to an external network, so that configuring them is as easy as configuring standard interfaces. The following is a command line that configures virtual machine `foo` to have a `TAP` (LAN or Internet connected) interface:

```
vstart foo --eth0=tap,10.0.0.1,10.0.0.2
```

The two IP addresses are automatically assigned to the `TAP` interface on the host and to the emulated interface inside the guest, respectively. These addresses are also used to set a proper routing table inside the guest. Both addresses must be provided so that, once the virtual machine has started up, it is already able to reach the external network. Notice that the configuration of a `TAP` interface requires administrative privileges, therefore Netkit will ask for the root password when the above command is executed.

8.2.3 A Filesystem of Networking Tools

Netkit is meant to be an environment that makes it easy and quick to set up complex network experiences. For this reason, virtual machines are equipped with a filesystem that contains most well known servers and tools for network analysis.

The Netkit filesystem contains a full-fledged Debian GNU/Linux distribution [207] which has been suitably tuned to run inside UML. This provides users with a familiar environment and allows to quickly grab new software pieces that might be needed for specific experimentation purposes, or to flexibly upgrade currently installed tools. Actually, installing or upgrading software pieces is simply a matter of starting an Internet connected virtual machine (see Section 8.2.2) and running the `apt` tools [46].

Among the network services available in Netkit there are the `apache` web server, the `bind` Domain Name Server, a DHCP server, the `exim4` Mail Transport Agent, an FTP server, the netfilter configuration tool `iptables`, an NFS server, and a Samba server. Utilities include `traceroute`, `ping`, `arping`, `netcat`, `tcpdump`. For a complete list of installed packages, see [226].

Netkit makes use of the Copy-On-Write technique described in Section 8.2.1 to save disk space when multiple virtual machines are run. In this way there is only one shared backing file containing the model filesystem downloaded with Netkit, and each virtual machine stores changes to that filesystem inside its own COW file. In this way, not only the filesystem is shared among all the virtual machines, so that they see completely aligned tools and services, but it is also possible to easily revert to the original filesystem contents by simply deleting a COW file in case things mess up. There is also an option of the Netkit commands that allows to write changes to the model filesystem permanently, which comes handy for example when installing new packages which are supposed to be available inside all virtual machines.

In order to facilitate the transfer of files between the host and a virtual machine, the special directory `/hosthome` inside a virtual machine makes the

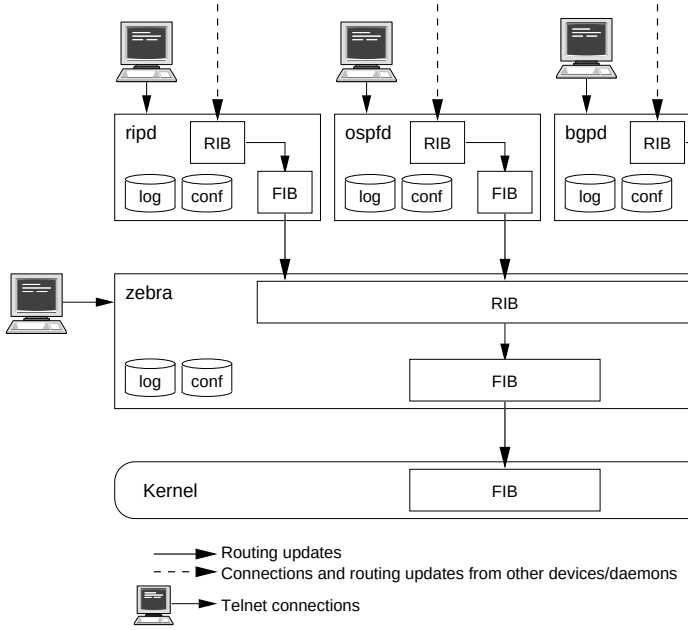


Figure 8.9: *An abstraction of the architecture of the Zebra routing software.*

user's home directory on the host always accessible. The same technique of providing special directories pointing to the host filesystem is also used by Netkit to automatically transfer settings for preconfigured network experiences. For more details about this, see Section 8.3.

The Zebra Routing Software Suite

A special mention is due to the routing software installed in the Netkit filesystem. In order to experiment with routing protocols, Netkit comes with an installed release of the Zebra routing software [108]. Zebra is a suite of daemons that provide support for several routing protocols, including RIP [55], OSPF [85], and BGP [252]. Routing protocols take care of spreading information about available destinations on a network in order to automatically update the routing tables of each device.

Figure 8.9 describes an abstraction of the architecture of Zebra and of the

way in which it injects information in the kernel routing table. Each Zebra routing daemon manages a specific routing protocol, has its own configuration file, and writes to its own log. They listen on different TCP ports, so that messages of a particular routing protocol can be sent to the appropriate daemon. For each routing protocol, a Routing Information Base (RIB) and a Forwarding Information Base (FIB) are maintained. The RIB is the set of all destinations known to that protocol, together with the path to reach them and some additional reachability information. The FIB contains, for each possible destination on the network, only the alternative that is considered the best one to reach it. Zebra in itself is a routing daemon: it receives information from the FIBs of the other daemons and, for each destination, selects the best alternative among those made available by different routing protocols. Zebra's best routes are finally injected into the routing table of the kernel, which is used to actually forward packets.

All the routing daemons, including Zebra, can be contacted via `telnet` on a dedicated TCP port to check the status of routing protocols and perform "on the fly" configuration. The daemons provide a command line interface which closely resembles that of Cisco routers. Most of the commands available on real devices can be used, but each daemon only provides those commands that are specifically oriented to the routing protocol it manages. For example, `ripd` does not provide the `show ip bgp` command, and `bgpd` must be contacted in order to be able to issue it. However, the Zebra suite also comes with `vtysh`, an integrated shell that provides a unique interface to all the daemons. Figure 8.10 shows a sample session of usage of the `bgpd` prompt inside a virtual machine.

Unfortunately, Zebra development is somewhat slow. For this reason, and also in order to create a community that does not rely on a centralized model, the Quagga project has been started [233]. Quagga is essentially a fork of Zebra in which proposals from a community of users are usually discussed and more quickly acknowledged. As a result, Quagga provides bug fixes and functionalities that are missing in Zebra, sometimes at the expense of stability (both stable and unstable releases of Quagga are available). At present, Netkit does not provide the Quagga routing suite. However, it can be easily installed in case it is needed, and there are plans to include it in future releases.

An alternative routing software suite is XORP [82, 136, 137, 134], an extensible routing platform that overcomes some of the limitations of Zebra/Quagga. XORP leverages a framework for abstracting and decoupling routing policies from protocols [8]. This allows a greater degree of flexibility than in products like Zebra/Quagga, where policy constructions are hardwired: for example, there is no way to express a matching condition like $metric < 3$ in Ze-

Virtual machine console

```
router:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to router.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
bgpd> enable
bgpd# configure terminal
bgpd(config)# router bgp 1
bgpd(config-router)# network 10.0.0.0/8
bgpd(config-router)# neighbor 192.168.0.1 remote-as 2
bgpd(config-router)# end
bgpd# disable
bgpd> show ip bgp
BGP table version is 0, local router ID is 0.0.0.0
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 10.0.0.0         0.0.0.0                  0         32768 i

Total number of prefixes 1
bgpd> exit
```

Figure 8.10: *Session of usage of the* `bgpd` *daemon. Available commands closely resemble those of Cisco routers.*

bra/Quagga. Another difference is that the Zebra/Quagga interface supports Cisco IOS-like commands, while XORP's interface is quite similar to that of Juniper JunOS platforms. XORP intends to be an attractive alternative both for researchers and for hardware vendors, as stability for mission-critical production use is one of the main goals of the project. The current version of the Netkit filesystem does not include XORP. Yet, it can be installed in case it is needed.

8.2.4 User Interface

One of the most interesting features of Netkit is the user interface, which includes tools for quickly and easily setting up complex and redistributable network experiences. Netkit virtual machines can be managed by means of a set of commands consisting of `v`-prefixed (*ltools*) and `l`-prefixed (*vtools*) utilities. All these commands must be run on the host machine and are implemented as shell scripts.

`vtools` are conceived to configure and manage single virtual machines, and they provide the following functionalities:

`vstart` can be used to configure and start a new virtual machine, identified by an arbitrary name. It allows to set parameters such as the amount of available memory, the kernel and filesystem to be used, the type of terminal device to make available, as well as the network interfaces and the collision domains they are attached to. Netkit's default settings usually fit most of the needs, so that starting a virtual network device often simply consists in specifying the network interfaces it should be equipped with.

For example, the following command starts a new virtual machine named `pc1`, with no network interfaces.

```
vstart pc1
```

If Netkit is properly installed, the above command prints some informative messages on the host terminal and boots `pc1` inside a new terminal window. Figure 8.11 shows the booting phase of `pc1`. It is possible to notice that boot time messages are exactly those that would be observed on a standard Debian system, and that at the end the user is presented a shell prompt as root in the virtual machine.

The following commands start two virtual machines, `pc2` and `pc3`, with two network interfaces each. Notice that `pc2`'s `eth0` and `pc3`'s `eth0` are attached to the same collision domain `COLL-DOM-A`, therefore the two virtual machines will be able to communicate with each other.

```
vstart pc2 --eth0=COLL-DOM-A --eth1=COLL-DOM-B
vstart pc3 --eth0=COLL-DOM-A --eth1=COLL-DOM-C
```

Once they have finished booting, `pc2` and `pc3`'s network interfaces can be configured by using `ifconfig` as shown in Figure 8.12, so that the

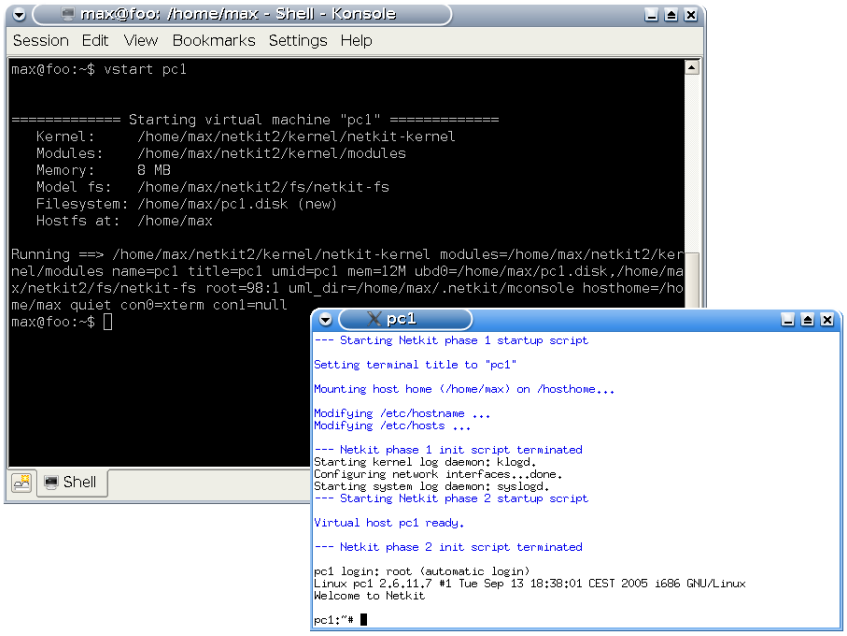


Figure 8.11: *A Netkit virtual machine starting up. The window with black background is a terminal on the host, while the other one is the virtual machine terminal.*

two machines can reach each other. Also notice that informational messages on the host terminal show that the necessary `uml_switch`es have automatically been started without any need for user intervention.

`vconfig` can be used to attach a network interface "on the fly" to a running virtual machine. This is useful to alter the configuration of an already running scenario or just to avoid having to reboot some machine because one of its interfaces has been forgotten.

The syntax of this command is analogous to that of `vstart`, so that the following command adds to `pc2` an interface `eth2` attached to collision domain `COLL-DOM-D`:

```
vconfig pc2 --eth2=COLL-DOM-D
```

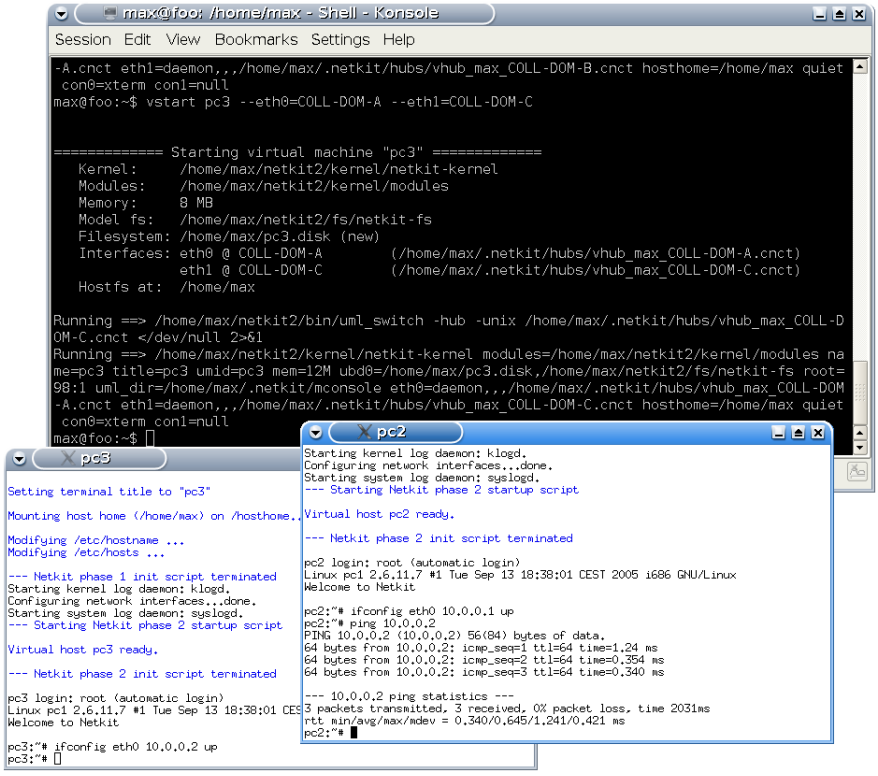


Figure 8.12: *A simple Netkit experiment consisting of two directly connected virtual machines.*

`vlist` is a command that provides information about currently running virtual machines. If invoked with no arguments, it produces a list of virtual machines, including the user that has started them, the PID of the tracing thread, the amount of consumed memory, and the list of network interfaces with the collision domains they are attached to (see, e.g., Figure 8.13). If a virtual machine name is provided as argument, `vlist` provides detailed information about it, as shown in Figure 8.14.

`vhalt` is exactly the equivalent of running `halt` inside a virtual machine. It gracefully stops the machine by running its shutdown scripts, properly

Host console

```
max@foo:~$ vlist
USER              VHOST                PID       SIZE  INTERFACES
max               pc2                  920      13140  eth0 @ COLL-DOM-A,
                                                       eth1 @ COLL-DOM-B
max               pc3                 1124      12380  eth0 @ COLL-DOM-A,
                                                       eth1 @ COLL-DOM-C

Total virtual machines:       2    (you),        2    (all users).
Total consumed memory:    25520 KB (you),    25520 KB (all users).
```

Figure 8.13: *Sample output of* `vlist`*. The command shows that there are two virtual machines running and reports their owner, the PID of the tracing thread, the amount of consumed memory, and a list of network interfaces, together with the collision domains they are attached to.*

Host console

```
max@foo:~$ vlist pc3

============= Information for virtual machine "pc3" =============
 --- Accounting information ---
   PID:        1124
   Owner:      max
   Used mem:   12380 KB
 --- Emulation parameters ---
   Kernel:     /home/max/netkit2/kernel/netkit-kernel
   Modules:    /home/max/netkit2/kernel/modules
   Memory:     8 MB
   Model fs:   /home/max/netkit2/fs/netkit-fs
   Filesystem: /home/max/pc3.disk
   Interfaces: eth0 @ COLL-DOM-A  (/home/max/.netkit/hubs/
                                                vhub_max_COLL-DOM-A.cnct)
               eth1 @ COLL-DOM-C  (/home/max/.netkit/hubs/
                                                vhub_max_COLL-DOM-C.cnct)
   Hostfs at:  /home/max
   Console 1:  terminal emulator
   Console 2:  disabled
   Other args: umid=pc3 root=98:1 uml_dir=/home/max/.netkit/mconsole quiet
   Mconsole:   /home/max/.netkit/mconsole/pc3/mconsole
```

Figure 8.14: *Usage of* `vlist` *to get detailed information about a running virtual machine.*

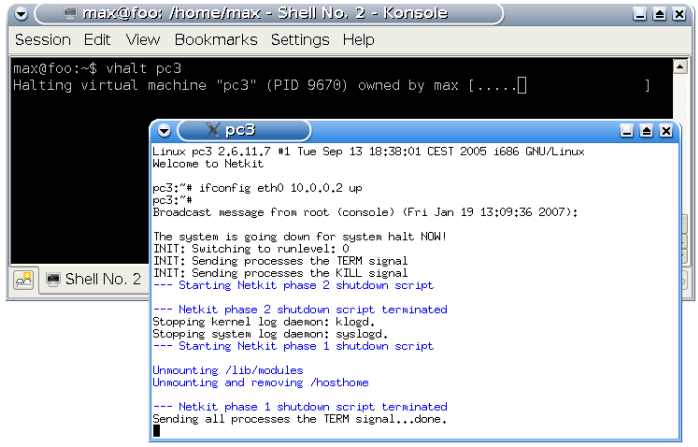


Figure 8.15: *Shutdown of a virtual machine by using* `vhalt`.

unmounting the filesystem and stopping it. `vhalt` is provided on the host as a convenient tool for performing the shutdown of multiple virtual machines within a script. Figure 8.15 shows the usage of `vhalt` for stopping `pc3`.

Host console

```
max@foo:~$ vclean
Killing unsued virtual hubs owned by user max:
   /home/max/.netkit/hubs/vhub_max_COLL-DOM-D.cnct (PID 4168): Killing... done.
```

Figure 8.16: *Usage of* `vclean` *to kill unused virtual hubs.*

`vcrash` provides a method to immediately stop a running virtual machine. Actually, using `vcrash` is equivalent to abruptly unplugging the virtual machine power cord. This is achieved by first asking the UML kernel to stop itself through a special UNIX management socket. The UML utility [155, 239] `uml_mconsole` is used for the purpose. In case this attempt fails, for example because UML is somewhy frozen, the virtual machine processes are automatically killed by `vcrash`. Upon the subsequent boot, a crashed virtual machine will run a filesystem check to recover inconsis-

tencies. This can be avoided, and the filesystem can be reverted to the original contents, by simply removing the COW file.

`vcrash` is often used in networking experiments consisting of several machines, as it is much faster than `vhalt`.

`vclean` is the Netkit "panic button". Should things mess up, some virtual machine be stuck, or tunnel configurations be left on the host, `vclean` helps in getting rid of all this with a single command. `vclean` can be invoked to perform several operations: it can simply remove unused `uml_switches`, kill all running virtual machines owned by a specific user, and remove tunnels connecting to an external network. Figure 8.16 shows the usage of `vclean` to kill unused virtual hubs.

`vclean` also comes handy when used in combination with `vconfig`. In fact, due to the mechanism by which they are configured, network interfaces attached by `vconfig` do not show up in the output of `vlist` nor the `uml_switch`es they are attached to can be automatically killed when the virtual machine stops: `vclean` must be used for the purpose.

`vtools` can be profitably used for configuring, starting, and managing few virtual machines. In principle, if used within a suitably written shell script, they could also be exploited to automatically start several virtual machines by invoking a single command. However, the setup of a complex experience usually involves lots of configurations not just of the emulated hardware but also of the services that should be available on the emulated network. Also, it may be difficult to translate the network topology in terms of `vstart` command line options. For this reason, Netkit provides higher level `ltools` which allow to easily set up, launch, or shutdown a complex scenario in a straightforward way. The `l` in `ltools` stands for "laboratory" (in the following abbreviated as "lab"), which is the name that is often associated to preconfigured redistributable Netkit scenarios. `ltools` rely on functionalities provided by `vtools` and offer the following interface:

`lstart` is the command that is used to start virtual machines that make up a lab. Actually, starting all of them is as simple as issuing `lstart` alone in a shell on the host. Details about how to prepare a self running scenario are provided in Section 8.3. Optionally, `lstart` can be used to start only a subset of the virtual machines that are part of the lab.

`ltest` supports the creation of redistributable self testing labs. Basically, the principle behind `ltest` is that each virtual machine is automatically in-

structed to perform a set of hardwired and user defined dumps of the significant information that contributes to define its status. For example, the dumps can include the contents of a routing table or the results of a `ping`. These dumps can then be collected and saved as a signature of a correctly running emulated network. Once the lab is moved to a different host, checking that it is still properly running is simply a matter of running it in test mode and verifying that the obtained dumps match the signature.

`lhalt` and `lcrash` behave like their counterparts `vhalt` and `vcrash`, but they automatically perform the shutdown or crash operation on all the virtual machines that make up a lab. Optionally, `lhalt` and `lcrash` can affect only a subset of the machines of the lab, for example in case some of them need to be rebooted without having to restart the whole lab.

`linfo` can be used to get basic information about a lab, including descriptive data and a list of the virtual machines that make it up. Figure 8.17 shows an example of its usage. `linfo` can also be used to get a sketch of the data-link topology of a lab, including hosts, interfaces, and collision domains. This feature requires the Graphviz [214] graph drawing library to be correctly installed. Figure 8.18 shows an example of topology generated by `linfo`. Ellipses represent hosts and are surrounded by integer values indicating network interfaces. Diamonds represent collision domains (virtual hubs).

`lclean` just performs a cleanup of temporary files left over after running a lab (COW files, logs, etc.). It has nothing to do with its counterpart `vclean`, which instead performs cleanup on running processes.

Both the `vtools` and the `ltools`, as well as the other Netkit components, are fully documented by means of `man` pages that are available with the Netkit distribution.

8.3 Setting up a Virtual Lab

It has already been pointed out in this Chapter that setting up an emulated network experiment involves dealing with lots of configuration files and a potentially complex topology. Netkit provides tools to facilitate this task and to support the creation of easily redistributable network scenarios that can be

Host console

```
max@foo:~/netkit-lab_bgp-6-multi-homed-stub$ linfo

====================== Lab information ==========================
Lab directory: /home/max/netkit-lab_bgp-6-multi-homed-stub
Version:       1.0
Author:        The Netkit Team
Email:         netkit.users@list.dia.uniroma3.it
Web:           <unknown>
Description:
<unknown>

The lab is made up of 4 virtual machines ( as100r1 as200r1 as20r1 as20r2).
================================================================
```

Figure 8.17: *Sample usage of* `linfo` *to get basic information about a Netkit lab.*

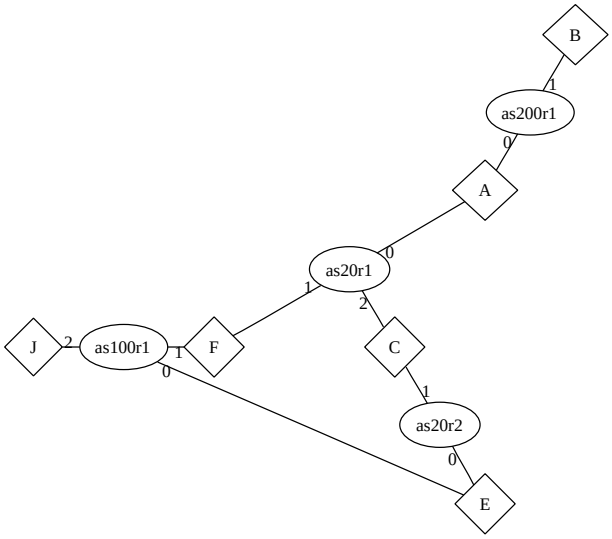


Figure 8.18: *A sketch of the data-link layer topology generated by* `linfo`*. Ellipses represent hosts; diamonds represent collision domains (virtual hubs); integers around ellipses indicate network interfaces.*

Directories: each directory specifies the existence of a virtual machine named as the directory itself; files contained in a directory `vm` are automatically copied to the root (`/`) of `vm`'s filesystem; files contained in the `shared` directory are copied to the root (`/`) of every virtual machine.

`lab.conf`**:** a file describing the link-level topology and other configuration parameters for virtual machines (amount of memory, etc.).

Startup and shutdown scripts: they can be used to apply some settings (e.g., configure IP addresses, start services) during the boot phase; they can be shared or specific to each virtual machine.

`_test`**:** a directory that contains scripts for dumping the status of virtual machines and that accommodates the dumps themselves.

`lab.dep`**:** a file describing dependencies in the boot order of virtual machines.

Figure 8.19: *Summary of the components of a Netkit lab.*

launched with a single command. This Section describes how to prepare such a scenario and manage it with Netkit's `ltools`.

A Netkit *lab* is a set of fully preconfigured virtual machines that can be started or stopped together. Netkit comes with a set of ready to use labs implementing interesting network scenarios involving bridging, transport level congestion control algorithms, routing protocols (with special attention to BGP), and application level services (DNS, E-mail). The labs can be downloaded from [36]. Further labs are periodically added.

Figure 8.19 summarizes the components of a Netkit lab. Actually, a lab consists of a hierarchy of files and directories having special roles. The following Sections describe these roles in detail and explain how to create a custom Netkit lab.

8.3.1 Defining the Topology

Before starting to configure a lab, it is strongly advised to prepare a detailed map of the topology that is about to be implemented. We propose an effective formalism to do this, which is also used in the documentation and examples supplied with Netkit. Figure 8.20 shows an example of usage of this formalism. As explained in the legend, router symbols represent emulated devices; they

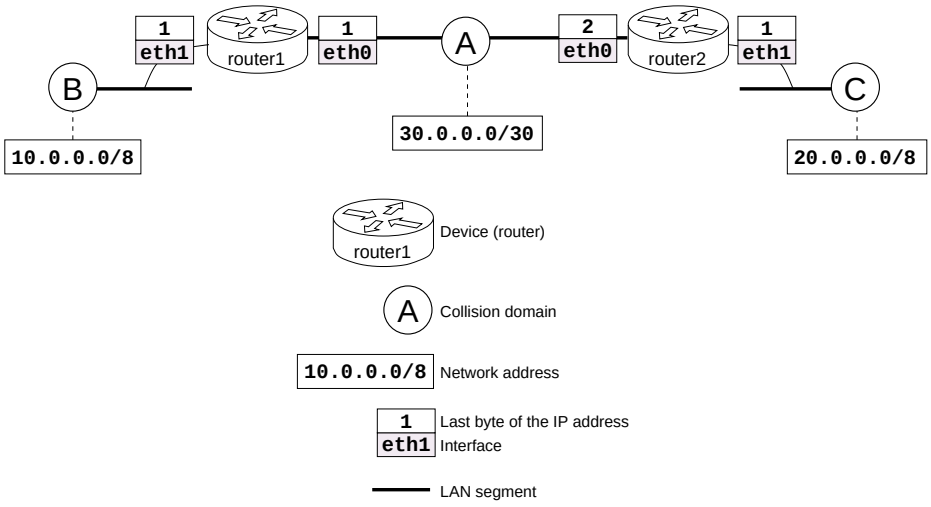


Figure 8.20: *A possible formalism for describing the link-level topology of a lab.*

do not necessarily have to be routers but, since most of the labs are likely to deal with routing issues, they are represented as such. Circled letters represent collision domains; the letter serves as identifier for the collision domain, and will be used by Netkit to associate a virtual hub to it. Each collision domain is linked to a square box containing the address of the corresponding subnet. Boxes surrounding devices detail information about their network interfaces: the lower half contains the name of the interface and the upper half specifies the last byte of its IP address on the subnet it is attached to. A thick line represents a local network; the devices that are connected to it may be missing in case there is no reason to emulate them (for example because they would be standard PCs providing no particular service).

In the following we use the topology in Figure 8.20 as a reference for implementing a sample Netkit lab. Collision domains B and C represent internal LAN segments.

8.3.2 Implementing the Topology

The topology of a network can be specified in Netkit in two steps.

Host console

```
max@foo:~/sample_lab$ ls -l
total 8
drwxr-xr-x 2 max max 4096 2007-01-20 15:49 router1
drwxr-xr-x 2 max max 4096 2007-01-20 15:49 router2
max@foo:~/sample_lab$ linfo

====================== Lab information ==========================
Lab directory: /home/max/sample_lab
Version:       <unknown>
Author:        <unknown>
Email:         <unknown>
Web:           <unknown>
Description:
<unknown>

The lab is made up of 2 virtual machines ( router1 router2).
=================================================================
```

Figure 8.21: *Sample Netkit lab made up of two virtual machines.*

```
LAB_DESCRIPTION="Sample lab for testing purposes"
LAB_VERSION="0.1"
LAB_AUTHOR="Massimo Rimondini"
LAB_EMAIL="contact@netkit.org"
LAB_WEB="http://www.netkit.org/"

router1[0]=A
router1[1]=B

router2[0]=A
router2[1]=C
```

Figure 8.22: *A sample* `lab.conf` *file.*

First, Netkit needs to know the virtual machines that make up the lab. Each directory in a lab represents a virtual machine named as the directory itself. The sole existence of a directory tells Netkit to start a virtual machine with that name. Figure 8.21 shows an example of lab consisting of two virtual machines, `router1` and `router2`, which is also confirmed by `linfo`.

Second, the link-level topology of the lab must be defined. The file `lab.conf` contains the description of the topology of the lab in terms of connections

```
ifconfig eth0 30.0.0.1 netmask 255.255.255.252 up
ifconfig eth1 10.0.0.1 netmask 255.0.0.0 up
/etc/init.d/zebra start
```

Figure 8.23: *A startup file for* `router1` *(*`router1.startup`*).*

between different virtual machines. An example of `lab.conf` file is shown in Figure 8.22. The first part of the file contains optional descriptive information about the lab, which may be useful when the lab is redistributed to other people. The rest of the file contains a mapping between network interfaces and collision domains. For example, the line:

```
router1[0]=A
```

means that interface `eth0` of `router1` is attached to collision domain A.

Optionally, other configuration parameters can be put inside `lab.conf`. For example, the amount of memory for a virtual machine can be increased to accommodate larger routing tables with a line similar to the following:

```
router[mem]=128
```

Basically, any valid option to `vstart` can be used between square brackets and assigned a value.

8.3.3 Setting Network Addresses and Startup Time Services

Virtual machines can be instructed to run specific commands on startup and shutdown. During the startup phase, each virtual machine `vm` runs the scripts `shared.startup` and `vm.startup`. These scripts must therefore contain commands that are available in the virtual machine filesystem. Apart from this restriction, almost anything can be put in the startup files. In practice they are generally used to set IP addresses for network interfaces and to start network services. Hence, a typical startup file often looks like the one in Figure 8.23, which tells `router1` to self configure IP addresses for its network interfaces and to enable the Zebra routing daemon. It comes straightforward that `router2.startup` will look much similar.

In the same way, virtual machines can run user defined shutdown scripts. Upon halting, a virtual machine `vm` first runs `vm.shutdown` and then `shared.shutdown`.

Host console

```
max@foo:~/sample_lab$ tree -n
.
|-- lab.conf
|-- router1
|   \-- etc
|       \-- zebra
|           |-- daemons
|           \-- ospfd.conf
|-- router1.startup
|-- router2
|   \-- etc
|       \-- zebra
|           |-- daemons
|           \-- ospfd.conf
\-- router2.startup
```

Figure 8.24: *A possible hierarchy of directories for a lab. Configuration files are placed inside* `router1` *and* `router2`*. All the files inside these two directories are mirrored inside the homonymous virtual machines upon their startup.*

Consider that shutdown scripts are only executed if the lab is gracefully stopped with `lhalt`.

8.3.4 Configuring Services

After specifying which services should be started at boot time, configuration files for them must also be provided. In Netkit this is made possible by a mechanism that automatically makes some of the files that are part of the lab available inside a virtual machine.

In particular, upon starting up a virtual machine Netkit copies all the files inside the directory associated with that machine inside its filesystem. For example, if `vm` is a virtual machine, all the files inside directory `vm/` on the host are copied to the root directory (`/`) of `vm`'s filesystem during its boot phase. If several machines need to access the same files, these can be placed in a directory named `shared`: everything inside `shared` is copied to the root (`/`) of any virtual machine upon its startup. Netkit recognizes `shared` to be a special directory and does not start a virtual machine for it.

This approach of mirroring files from the host has a twofold advantage. It does not add any overhead to the configuration, because the only files to be prepared are those that will be actually fetched by the servers. Moreover, in

```
# This file tells the zebra package
# which daemons to start.
# Entries are in the format: <daemon>=(yes|no|priority)
# where 'yes' is equivalent to infinitely low priority, and
# lower numbers mean higher priority. Read
# /usr/doc/zebra/README.Debian for details.
# Daemons are: bgpd zebra ospfd ospf6d ripd ripngd
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

Figure 8.25: *A sample* `daemons` *file, telling Zebra which routing daemons to start.*

```
hostname ospfd
password zebra
!
router ospf
network 30.0.0.0/30 area 0
redistribute connected
```

Figure 8.26: *A sample* `ospfd.conf` *file.*

this way there is no restriction on the number or type of services that can be configured.

In the case of our example, we may have a hierarchy of files like the one in Figure 8.24. It is possible to notice that we have provided two configuration files for each router. `daemons`, shown in Figure 8.25, tells Zebra which routing daemons should be activated. `ospfd.conf`, shown in Figure 8.26, enables basic OSPF routing in order to make the internal LAN of `router1` reachable from `router2` and vice versa. Due to the symmetrical nature of the example, the two files `daemons` and `ospfd.conf` are identical on the two routers.

This configuration pattern is common when the Zebra routing daemon is being used: routing protocols to be supported are listed inside `etc/zebra/daemons`, while protocol specific configurations go inside etc/zebra/*`daemon_name`*`.conf` files.

8.3.5 Tuning Lab Startup

Once the lab has been prepared, it is possible to alter some other settings that affect the way in which it will be started. By default, Netkit starts the virtual machines of a lab one after the other, and waits for the previous one to end the boot phase before starting the next one. This is done in order to prevent overloading of the host machine. An option of `lstart` allows to disable this precautionary measure and to start multiple virtual machines at the same time. A variant of the same option also allows to ensure that at every time instant no more than a fixed number of virtual machines is booting: this is a good compromise between lab startup speed and host side load.

If Netkit is instructed to simultaneously start multiple virtual machines, some of the services may not start up properly. This is usually the case when a virtual machine attempts to initialize a service that depends on another one hosted on a virtual machine that has not been started yet (for example, some mail service relying on DNS). For this reason, the startup order of virtual machines can be influenced in two ways. One possibility is to explicitly tell Netkit the startup order. This can be done by either explicitly listing virtual machines on the command line of `lstart` or by placing an assignment like the following one inside `lab.conf`:

```
machines="first_vm second_vm third_vm fourth_vm fifth_vm"
```

A more flexible way to influence the startup order is to just specify dependencies between virtual machines. This can be done by means of a file `lab.dep` whose syntax is exactly the same used in Makefiles. For example, in Figure 8.27 line

```
pc2: router pc1
```

means that `router` and `pc1` have no dependencies on each other and can therefore be started simultaneously (provided that the maximum number of booting machines is not exceeded). However, `pc2` can only be started after both `router` and `pc1` have completed their boot phase. Observe that, according to Figure 8.27, also `pc2` and `pc3` can start up at the same time after `router` and `pc1`. Instead, `pc4` cannot be started until all the other machines are running.

In the example of lab presented in these Sections we do not make use of the `lab.dep` file.

```
pc2: router pc1
pc3: router pc1
pc4: pc2 pc3
```

Figure 8.27: *A sample* `lab.dep` *file specifying dependencies on the startup order of virtual machines.*

```
#!/bin/sh

# Connectivity tests
ping -c 3 -i 0.3 195.11.14.1 | head -n -3 | sed 's/time=.*//'

sleep 5

# Inspect the arp cache (should be populated)
arp | sort

sleep 5
halt
```

Figure 8.28: *A sample script for performing diagnoses when a lab is launched in test mode.*

8.3.6 Testing the Lab

Netkit labs are conceived to facilitate distribution and execution on any other platform equipped with a release of Netkit. If things are configured properly, network experiences should run in the same way and exhibit the same evolution on any platform. However, there may still be particular settings that affect the behavior of poorly configured labs or cause malfunctions of the emulation environment. For this reason, a lab can be equipped with a special set of scripts that instruct virtual machines to perform aimed self diagnoses. Once the lab has been found to behave properly, the results of these diagnoses can be collected and constitute a signature of a correctly running lab. When moving it to a different platform, the lab can again be self tested and the results be compared against the signature. If they match, the lab is running properly.

The scripts for self diagnosis must be placed inside the directory `_test`. Netkit understands that this is a special directory and does not start a virtual machine for it. The script `vm.test` inside `_test` is automatically executed by virtual machine `vm` when the lab is in launched in test mode (see Section 8.2.4).

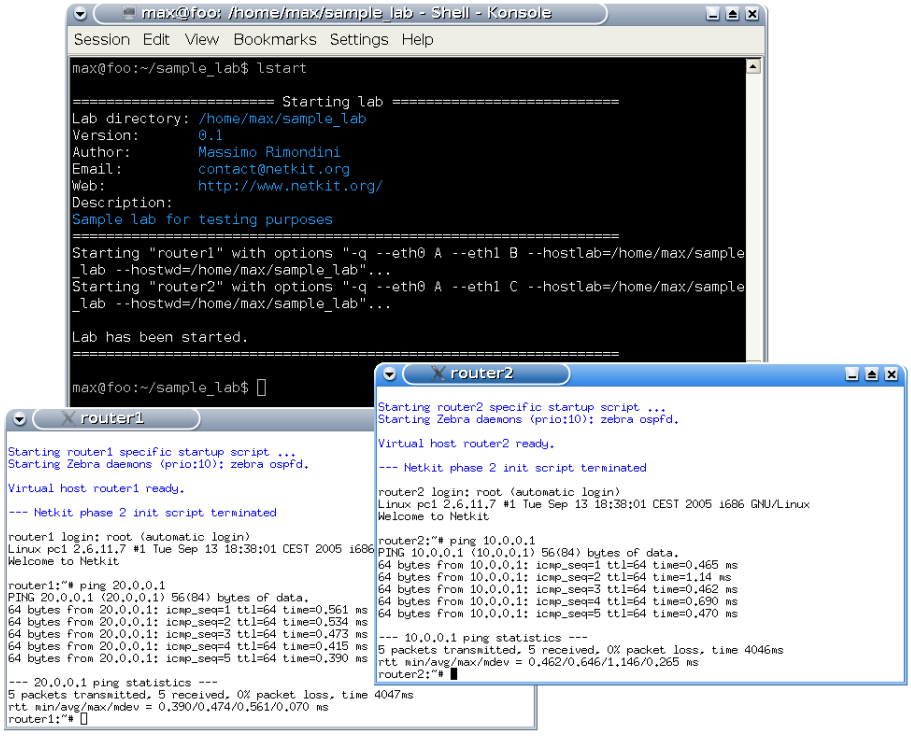


Figure 8.29: *Startup of the Netkit lab described in Section 8.3*

The output of this script is stored in `results/vm.user` inside the `_test` directory. In addition, Netkit also performs some default diagnoses including the status of network interfaces, the contents of the kernel forwarding table, a summary of the ports in listening status, and a listing of running processes. The results of these default diagnoses are stored inside `results/vm.default` inside directory `_test`.

Figure 8.28 shows a possible example of script for self diagnosis. It reports the success of a connectivity test and the contents of the ARP cache. Observe that information that is subject to unpredictable changes (round-trip time for the `ping`, ordering of the entries in the ARP cache) is filtered out, so that the results of tests performed on different platforms can be immediately compared.

router1 console

```
router1:~# telnet localhost ospfd
Trying 127.0.0.1...
Connected to router1.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
ospfd> show ip ospf database

       OSPF Router with ID (30.0.0.1)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
30.0.0.1        30.0.0.1        1193 0x80000004 0x0abd 1
30.0.0.2        30.0.0.2        1194 0x80000003 0x0abb 1

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
30.0.0.2        30.0.0.2        1199 0x80000001 0xfce0

                AS External Link States

Link ID         ADV Router      Age  Seq#       CkSum  Route
10.0.0.0        30.0.0.1        1198 0x80000002 0x1282 E2 10.0.0.0/8 [0x0]
20.0.0.0        30.0.0.2        1198 0x80000002 0x89ff E2 20.0.0.0/8 [0x0]
```

Figure 8.30: *A telnet session with* `ospfd` *showing the information collected by OSPF.*

8.4 Managing a Virtual Lab

Once a Netkit virtual lab has been set up, it can be easily managed by means of the `ltools`. The lab can be simply started by issuing `lstart` on the host, as demonstrated in Figure 8.29. The Figure also shows that, thanks to OSPF, each router can also `ping` the LAN it is not directly connected to. Other investigations can be performed on the running lab by simply interacting with one of the virtual machine terminal windows. For example, Figure 8.30

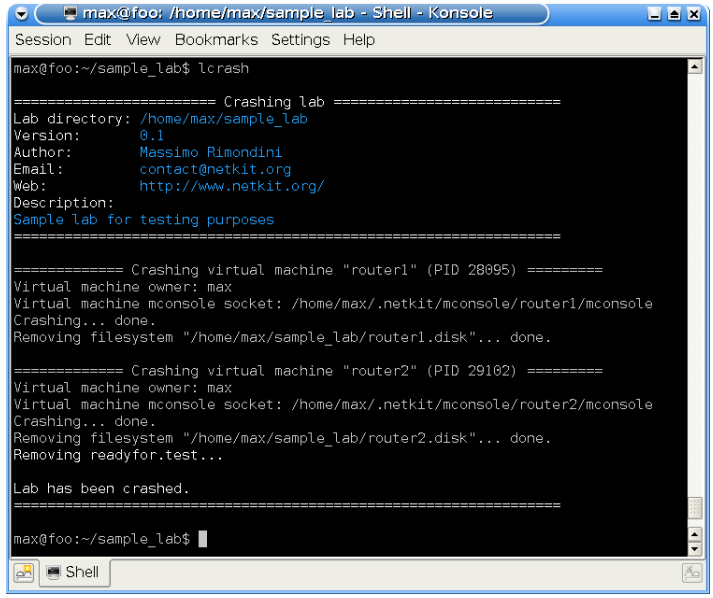


Figure 8.31: *Shutdown of a Netkit lab by using* `lcrash`.

shows the contents of the OSPF database.

Stopping a running lab is as simple as issuing `lcrash` or `lhalt` (slower) on the host (see Figure 8.31). Optionally, `lclean` can be used to get rid of temporary files (COW files, logs) left over after running the lab.

Netkit labs can be easily moved to another workstation by simply wrapping them into an archive, typically a `tar.gz`. Notice that, as there is no need to carry potentially large filesystem images, a lab can be packed into a very small file and can therefore be also transferred very quickly on the Internet. Taking advantage of this, lots of ready to use lab experiences are made available on the Netkit web site [36].

Thanks to the architecture described in Section 8.2, Netkit makes it simple to build even large network scenarios consisting of several virtual devices. In theory there is no limit to the number of virtual machines that can compose a lab. In practice, experiments have shown that it is possible to run experiences consisting of more than 100 virtual machines on a typical workstation (see Section 8.2.1).

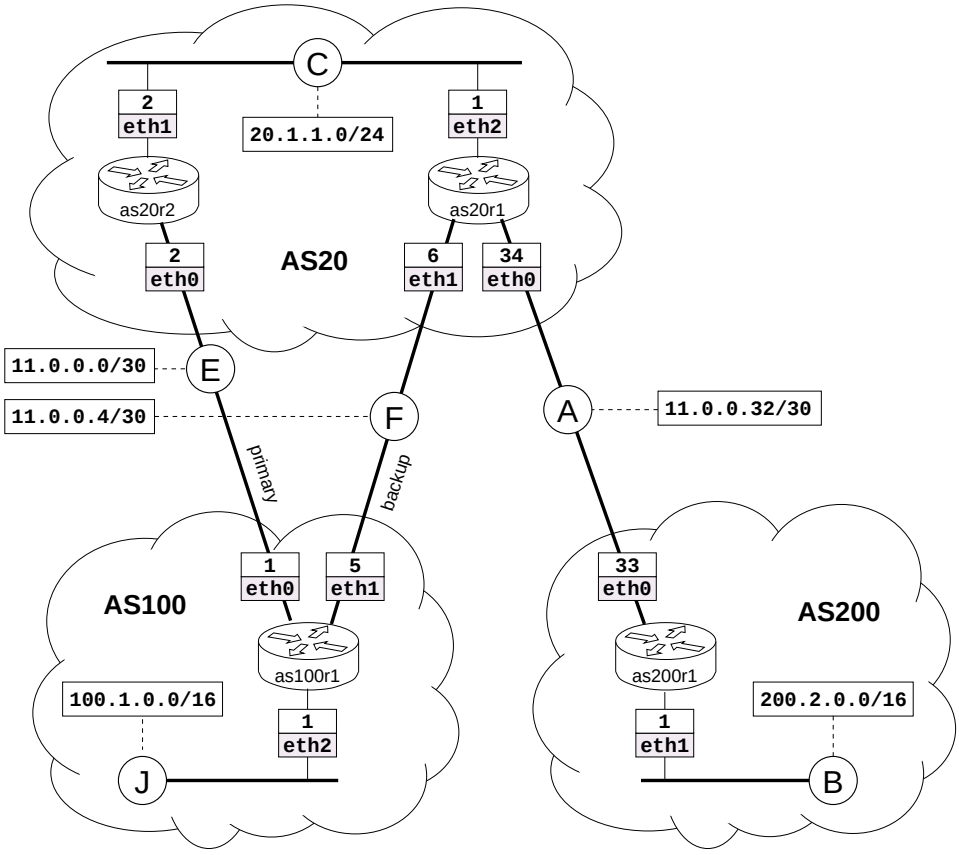


Figure 8.32: *The topology of the lab for studying multihomed stub networks.*

8.5 A Case Study: Multihoming

This Section provides an example of usage of Netkit to study issues that may arise in the interaction between routing protocols. We take as reference one of the labs made available on the Netkit web site [36], which considers a multihomed stub network as case study for pointing out these issues.

Figure 8.32 shows the topology of the lab we are considering. It consists of three Autonomous Systems: AS100 and AS200 are customers, while AS20

as100r1 console

```
as100r1:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to as100r1.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
bgpd> show ip bgp
BGP table version is 0, local router ID is 100.1.0.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 0.0.0.0          11.0.0.2                             0 20 i
*                   11.0.0.6                    90       0 20 i
*> 100.1.0.0/16     0.0.0.0                  0          32768 i

Total number of prefixes 2
```

Figure 8.33: *The BGP routing table of* `as100r1`.

is their provider. AS100 is a multihomed stub, meaning that it has multiple connections to its ISP but does not provide transit service to its neighbors. Instead, AS200 is single homed. As a provider, AS20 supplies transit service to its customers. The Autonomous Systems in this topology exchange routing information with each other by using BGP. In particular, AS100 exploits its multiple connections to the provider AS20 to enforce a backup policy: the link on collision domain F is never used unless link E fails. Keeping a link completely unused is of course an unrealistic situation. However, it is a reasonable assumption for the purpose of showing an example of the effect of routing policies on the choice of routing paths. The two routers inside AS20 establish an iBGP peering to exchange external routes they have learned via BGP.

Figure 8.34 shows the hierarchy of files and directories that make up the lab. Apart from the file `CHANGES`, which is only a log of the fixes and changes made to the lab, it can be easily seen from the configuration files that routers in this lab run the BGP routing protocol. The topology of Figure 8.32 is implemented

Host console

```
max@foo:~/netkit-lab_bgp-6-multi-homed-stub$ tree -n
.
|-- CHANGES
|-- as100r1
|   \-- etc
|       \-- zebra
|           |-- bgpd.conf
|           \-- daemons
|-- as100r1.startup
|-- as200r1
|   \-- etc
|       \-- zebra
|           |-- bgpd.conf
|           \-- daemons
|-- as200r1.startup
|-- as20r1
|   \-- etc
|       \-- zebra
|           |-- bgpd.conf
|           \-- daemons
|-- as20r1.startup
|-- as20r2
|   \-- etc
|       \-- zebra
|           |-- bgpd.conf
|           \-- daemons
|-- as20r2.startup
\-- lab.conf
```

Figure 8.34: *Directory structure for the multihoming lab.*

```
as20r1[0]="A"
as20r1[1]="F"
as20r1[2]="C"

as20r2[0]="E"
as20r2[1]="C"

as200r1[0]="A"
as200r1[1]="B"

as100r1[0]="E"
as100r1[1]="F"
as100r1[2]="J"
```

Figure 8.35: `lab.conf` *file describing the topology in Figure 8.32*

as100r1 console

```
as100r1:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to as100r1.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
bgpd> enable
Password: zebra
bgpd# show running-config

Current configuration:
!
hostname bgpd
password zebra
enable password zebra
log file /var/log/zebra/bgpd.log
!
debug bgp
debug bgp events
debug bgp keepalives
debug bgp updates
debug bgp fsm
debug bgp filters
!
router bgp 100
 network 100.1.0.0/16
 neighbor 11.0.0.2 remote-as 20
 neighbor 11.0.0.2 description Router as20r2 (primary)
 neighbor 11.0.0.2 prefix-list defaultIn in
 neighbor 11.0.0.2 prefix-list mineOutOnly out
 neighbor 11.0.0.6 remote-as 20
 neighbor 11.0.0.6 description Router as20r1 (backup)
 neighbor 11.0.0.6 prefix-list defaultIn in
 neighbor 11.0.0.6 prefix-list mineOutOnly out
 neighbor 11.0.0.6 route-map localPrefIn in
 neighbor 11.0.0.6 route-map metricOut out
!
access-list myAggregate permit 100.1.0.0/16
!
ip prefix-list defaultIn seq 5 permit 0.0.0.0/0
ip prefix-list mineOutOnly seq 5 permit 100.1.0.0/16
!
route-map metricOut permit 10
 match ip address myAggregate
 set metric 10
!
route-map localPrefIn permit 10
 set local-preference 90
!
line vty
!
end
```

Figure 8.36: *Configuration of BGP on router* `as100r1`.

as100r1 console

```
as100r1:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
11.0.0.4        *               255.255.255.252 U     0      0        0 eth1
11.0.0.0        *               255.255.255.252 U     0      0        0 eth0
100.1.0.0       *               255.255.0.0     U     0      0        0 eth2
default         11.0.0.2        0.0.0.0         UG    0      0        0 eth0
```

Figure 8.37: *Kernel forwarding table of* `as100r1`.

inside the file `lab.conf` as shown in Figure 8.35.

It is interesting to investigate in the BGP configuration of the customer `as100r1`. Figure 8.36 shows how to obtain this configuration through the command line interface of `bgpd` instead of the file `bgpd.conf`. Assuming that the provider AS20 does not apply countermeasures, the backup policy is enforced by the customer AS100 by using local-preference and metric. In particular, the route-map `localPrefIn` lowers to 90 the local-preference on announcements coming from neighbor `as20r1` (link F, used as backup); for the announcements received from link E the default value of 100 is fine. The other route-map, `metricOut`, is used to increase to 10 the metric on outgoing announcements made by `as100r1` to its neighbor `as20r1`; as for outgoing announcements directed to `as20r2`, the default metric value of 0 is retained. In this way, both the outgoing and the incoming traffic are discouraged from taking a path through link F. The other prefix-lists prevent transit traffic from traversing the customer `as100r1` by discarding announcements of extraneous prefixes.

By looking at the BGP routing table of `as100r1` (Figure 8.33), it can be easily seen that the chosen backup policy is actually being enforced. In fact, due to the higher value of the local-preference, the instance of BGP running on `as100r1` has chosen the path through neighbor `11.0.0.2` (`as20r2`) as best alternative to reach the default route announced by the provider. The "greater than" symbol (`>`) indicates the currently selected route. This information is further confirmed by the entry injected in the kernel forwarding table (see Figure 8.37).

Figures 8.38 and 8.39 show the status (`show ip bgp summary`) and routing table (`show ip bgp`) of BGP on `as20r2` and `as20r1`, respectively. The status confirms that the iBGP peering between `20.1.1.1` and `20.1.1.2` is active. However, while `as20r2` chooses `11.0.0.1` (link E, used as primary) as best alternative to reach `as100r1`, the same does not happen on `as20r1` despite

as20r2 console

```
as20r2:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to as20r2.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
bgpd> show ip bgp summary
BGP router identifier 20.1.1.2, local AS number 20
3 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
11.0.0.1        4   100     116     117        0    0    0 01:53:34        1
20.1.1.1        4    20     118     118        0    0    0 01:53:39        6

Total number of neighbors 2
bgpd> show ip bgp
BGP table version is 0, local router ID is 20.1.1.2
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i0.0.0.0          20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 11.0.0.0/30      0.0.0.0                  0         32768 i
*>i11.0.0.4/30      20.1.1.1                 0    100      0 i
*>i11.0.0.32/30     20.1.1.1                 0    100      0 i
* i20.1.1.0/24      20.1.1.1                 0    100      0 i
*>                  0.0.0.0                  0         32768 i
*> 100.1.0.0/16     11.0.0.1                 0             0 100 i
* i                 11.0.0.5                10    100      0 100 i
* i200.2.0.0/16     11.0.0.33                0    100      0 200 i

Total number of prefixes 7
```

Figure 8.38: *Peering status and BGP routing table of* `as20r2`.

as20r1 console

```
as20r1:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to as20r1.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
bgpd> show ip bgp summary
BGP router identifier 20.1.1.1, local AS number 20
3 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
11.0.0.5        4   100     118     121        0    0    0 01:56:49        1
11.0.0.33       4   200     118     120        0    0    0 01:56:50        1
20.1.1.2        4    20     119     122        0    0    0 01:56:41        4

Total number of neighbors 3
bgpd> show ip bgp
BGP table version is 0, local router ID is 20.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i0.0.0.0         20.1.1.2                 0    100      0 i
*>                 0.0.0.0                  0         32768 i
*>i11.0.0.0/30     20.1.1.2                 0    100      0 i
*> 11.0.0.4/30     0.0.0.0                  0         32768 i
*> 11.0.0.32/30    0.0.0.0                  0         32768 i
* i20.1.1.0/24     20.1.1.2                 0    100      0 i
*>                 0.0.0.0                  0         32768 i
* i100.1.0.0/16    11.0.0.1                 0    100      0 100 i
*>                 11.0.0.5                10             0 100 i
*> 200.2.0.0/16    11.0.0.33                0             0 200 i

Total number of prefixes 7
```

Figure 8.39: *Peering status and BGP routing table of* `as20r1`.

as20r2 console

```
as20r2:~# ping 200.2.0.1
connect: Network is unreachable
```

Figure 8.40: *A failed* `ping` *from* `as20r2`*.*

as200r1 console

```
as200r1:~# traceroute 100.1.0.1
traceroute to 100.1.0.1 (100.1.0.1), 64 hops max, 40 byte packets
 1  11.0.0.34 (11.0.0.34)  1 ms  2 ms  1 ms
 2  100.1.0.1 (100.1.0.1)  2 ms  2 ms  2 ms
```

Figure 8.41: *Traffic from* `as200r1` *to* `100.1.0.1` *takes the path through the backup link F, which is undesired.*

the fact that the entry with next hop `11.0.0.1` is present in the routing table of BGP, has lower metric and higher local-preference. Even more strangely, Figure 8.38 shows that the only available alternative on `as20r2` to reach prefix `200.2.0.0/16` has not been selected as best, and has therefore not been injected in the kernel forwarding table. As a consequence, a `ping` from `as20r2` to `200.2.0.1` unavoidably fails (see Figure 8.40). To complete the picture of oddities, Figure 8.41 shows that a `traceroute` from `as200r1` to `100.1.0.1` reveals that the backup link F is being used for traffic directed to `as100r1`, which is undesired.

Obviously, something is going wrong with routing, even if the deployed configuration is correct. The point here is that BGP only considers an entry of the routing table as usable if the path to reach its next hop has been learned by some IGP or is statically configured. In our case neither of the conditions occurs, as `as20r2` learns how to reach `11.0.0.33` only via iBGP and `as20r1` learns how to reach `11.0.0.1` again only via iBGP. The origin of learned routes can be checked by querying the Zebra daemon. For example, Figure 8.42 shows that the route to `11.0.0.33` has been learned by BGP.

There are two possible fixes to this situation. The first one is to enable some IGP inside AS20, so that information about directly connected networks is propagated inside the AS and can be used by BGP to reach all the next hops. The second one, which we apply here, is to configure two static entries in the forwarding tables of `as20r2` and `as20r1` telling them how to reach the next hops `11.0.0.33` and `11.0.0.1`, respectively. The commands for doing this are shown in Figures 8.44 and 8.45. The statically configured routes are

as20r2 console

```
as20r2:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to as20r2.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verification

Password: zebra
Router> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

C>* 11.0.0.0/30 is directly connected, eth0
B>* 11.0.0.4/30 [200/0] via 20.1.1.1, eth1, 03:19:58
B>* 11.0.0.32/30 [200/0] via 20.1.1.1, eth1, 03:19:58
C>* 20.1.1.0/24 is directly connected, eth1
B>* 100.1.0.0/16 [20/0] via 11.0.0.1, eth0, 03:19:53
C>* 127.0.0.0/8 is directly connected, lo
```

Figure 8.42: *The protocol by which routes in the forwarding table have been learned can be checked by querying the* `zebra` *routing daemon.*

as20r2 console

```
as20r2:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to as20r2.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.

User Access Verification

Password: zebra
Router> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

C>* 11.0.0.0/30 is directly connected, eth0
B>* 11.0.0.4/30 [200/0] via 20.1.1.1, eth1, 00:00:43
K>* 11.0.0.32/30 via 20.1.1.1, eth1
B   11.0.0.32/30 [200/0] via 20.1.1.1, eth1, 00:00:43
C>* 20.1.1.0/24 is directly connected, eth1
B>* 100.1.0.0/16 [20/0] via 11.0.0.1, eth0, 00:00:41
C>* 127.0.0.0/8 is directly connected, lo
```

Figure 8.43: *Statically configured routes are marked as* `kernel` *by Zebra and are preferred over those learned via other routing protocols.*

as20r2 console
```
as20r2:~# route add -net 11.0.0.32/30 gw 20.1.1.1 dev eth1
```

Figure 8.44: *Configuration of a static entry to reach `11.0.0.33` in the forwarding table of `as20r2`.*

as20r1 console
```
as20r1:~# route add -net 11.0.0.0/30 gw 20.1.1.2 dev eth2
```

Figure 8.45: *Configuration of a static entry to reach `11.0.0.1` in the forwarding table of `as20r1`.*

as100r1 console
```
as100r1:~# traceroute 200.2.0.1
traceroute to 200.2.0.1 (200.2.0.1), 64 hops max, 40 byte packets
 1  11.0.0.2 (11.0.0.2)  1 ms  1 ms  1 ms
 2  20.1.1.1 (20.1.1.1)  1 ms  2 ms  1 ms
 3  200.2.0.1 (200.2.0.1)  3 ms  2 ms  2 ms
```

Figure 8.46: *After fixing the forwarding tables of routers in AS20, a `traceroute` from `as100r1` to `200.2.0.1` correctly uses `as100r1`'s primary upstream link E.*

marked by Zebra as being learned from the kernel (see Figure 8.43) and are preferred over routes learned via other routing protocols.

After the fix has been applied, everything starts to work as desired. Figures 8.46 and 8.47 show that `traceroute`s actually use the primary link E.

Now that everything is working as expected, we can check that the backup policy is operational: we intentionally break link E and check that the traffic shifts to link F. Provided that we are inside an emulation environment, a link can be broken in at least two ways. One is to simply bring down one of the network interfaces attached to that link, by using the following command on either `as20r2` or `as100r1`:

```
ifconfig eth0 down
```

The other one is to administratively shutdown the BGP peering between `as20r2` and `as100r1`. We choose the latter alternative, which can be carried into effect by issuing the commands shown in Figure 8.50. Notice that, since a BGP peering must be explicitly configured by both participants, shutting it down on just one side is enough to divert routing.

as200r1 console

```
as200r1:~# traceroute -n 100.1.0.1
traceroute to 100.1.0.1 (100.1.0.1), 64 hops max, 40 byte packets
 1  11.0.0.34  1 ms  1 ms  1 ms
 2  20.1.1.2  1 ms  1 ms  1 ms
 3  100.1.0.1  2 ms  2 ms  2 ms
```

Figure 8.47: *After fixing the forwarding tables of routers in AS20, a* `traceroute` *from* `as200r1` *to* `100.1.0.1` *correctly uses* `as100r1`*'s primary upstream link E.*

as100r1 console

```
as100r1:~# traceroute 200.2.0.1
traceroute to 200.2.0.1 (200.2.0.1), 64 hops max, 40 byte packets
 1  11.0.0.6 (11.0.0.6)  1 ms  1 ms  1 ms
 2  200.2.0.1 (200.2.0.1)  1 ms  2 ms  1 ms
```

Figure 8.48: *After administratively shutting down the peering between* `as20r2` *and* `as100r1`*, traffic goes through the backup link F.*

as200r1 console

```
as200r1:~# traceroute -n 100.1.0.1
traceroute to 100.1.0.1 (100.1.0.1), 64 hops max, 40 byte packets
 1  11.0.0.34  2 ms  1 ms  1 ms
 2  100.1.0.1  1 ms  1 ms  1 ms
```

Figure 8.49: *After administratively shutting down the peering between* `as20r2` *and* `as100r1`*, traffic goes through the backup link F.*

After waiting some time for the routing updates to propagate, we fall into the expected state in which all the traffic uses the backup link F (see Figures 8.48 and 8.49).

We do not show, but it is easy to imagine, that restoring the peering also results in traffic shifting back to the primary link E.

8.6 Conclusions

In this Section we provide a survey of environments for the emulation of computer networks. We describe in detail Netkit, a lightweight network emulator based on User-Mode Linux. We show how Netkit can be effectively exploited to ease experimenting with complex scenarios consisting of several virtual devices.

as20r2 console

```
as20r2:~# telnet localhost bgpd
Trying 127.0.0.1...
Connected to as20r2.
Escape character is '^]'.

Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.


User Access Verification

Password: zebra
bgpd> enable
Password: zebra
bgpd# configure terminal
bgpd(config)# router bgp 20
bgpd(config-router)# neighbor 11.0.0.1 shutdown
bgpd(config-router)# end
bgpd# show ip bgp summary
BGP router identifier 20.1.1.2, local AS number 20
3 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
11.0.0.1        4   100     220     222        0    0    0 00:00:05 Idle (Admin)
20.1.1.1        4    20     226     223        0    0    0 03:37:37        6

Total number of neighbors 2
```

Figure 8.50: *A telnet session showing the commands to administratively shutdown a BGP peering.*

We also explain how these scenarios can be easily packed and redistributed for sharing them with others. Last, we show a sample application of Netkit to the study of a multihoming configuration, pointing out issues related to the interplay between BGP and IGP protocols. Among the other network emulators, Netkit has the advantages of being lightweight and to support installation and execution fully in user space and without administrative privileges. It also facilitates the exchange and redistribution of emulated scenarios and supports reasonably good scalability by allowing to run more than one hundred of virtual devices on a typical workstation. Moreover, Netkit provides users with a familiar environment consisting of a virtual boxes based on a standard Linux distribution and routing daemons using a Cisco-like command syntax.

One of the most natural contexts of application of Netkit is probably didactics: it can be effectively exploited in teaching networking by giving the students the opportunity to experiment with the protocols and services they are learning. However, we believe that Netkit may prove itself useful for both operators and researchers in several other contexts, ranging from testing of configurations before deployment to debugging and development of new services and protocols, from studying abnormal routing behaviors to validating theoretical model by experimentation.

Part V

Conclusions and Bibliography

Conclusions

Investigating the behavior of Internet routing is not an easy task. Not only the scale of the system being analyzed is difficult to handle, but also routing information is not always accessible. This is especially true for the case of inter-domain routing with BGP. Purposefully deployed routing policies limit the spread of BGP routing information, for example for implementing commercial agreements, and this makes it hard even to get the required data.

Yet, studying BGP routing phenomena is very important both for operators, in order to debug routing issues, and for researchers, in order to build sound models that can help in preventing abnormal and unexpected routing behaviors from happening. Several works have focused on BGP in the past, but there are still some aspects that need to be analyzed.

This thesis describes novel techniques to perform different kinds of investigation on BGP routing, with particular attention to the inference of routing policies.

The first, essential step to study the behavior of a routing protocol is to have topological information about the network it operates on. For the case of BGP, this means knowing the peerings established between Autonomous Systems. We propose two new methods to obtain this information: one exploits active probes that make use of standard BGP announcements, while the other processes and accurately purges information contained in the Internet Routing Registry. Experiments have proved the effectiveness of both methods: active probing allows to discover seven times as many peerings in the IPv6 Internet and more than two times as many in the IPv4 Internet, with respect to state of the art methods; accurate processing of IRR data allows us to extract about twice as many peerings as existing IRR tools are able to find. We believe that a seasonable use of these two techniques can effectively help in obtaining richer and more accurate AS-level topologies.

Once the topology is known, the investigation moves on to routing policies. While fragments of topological information can be obtained by looking at BGP

announcements, routing policies cannot be observed as they are part of the configuration of devices. Moreover, administrators are often unwilling to disclose information about routing policies, as they are considered critical for the economic strategies of an ISP.

This thesis proposes some novel techniques to find out and study routing policies. The first kind of analysis that is taken into account is the inference of commercial relationships between Autonomous Systems. Such knowledge would be of help both to debug routing problems and to devise more effective peering strategies for new ISPs. While some inference algorithms have already been introduced to obtain the commercial relationships, it is still unclear to what extent the results they produce are realistic and of practical interest. We propose a comparative analysis of state of the art inference algorithms that are based on the *valley-free* approach and compute measures that help in estimating their trustworthiness. The analysis shows that the results produced by the considered algorithms are fairly independent from background routing noise and only capture the longer term dynamics of commercial agreements. Also, the analyzed algorithms produce remarkably overlapping relationships, which confirms that the valley-free approach holds its validity regardless of the specific algorithm that exploits it.

Routing policies are often used to constrain traffic flows in order to achieve optimal usage of network resources. This thesis proposes two new models to describe different traffic engineering requirements that span from the optimization of bandwidth allocation to fair cost distribution. For each model we propose algorithms to determine the amount of AS-path prepending that should be used in BGP announcements in order to achieve the desired traffic engineering objective. We also introduce some optimized techniques to retrieve the topological information that is necessary for these algorithms to operate properly. We think that the traffic engineering algorithms we propose, used in conjunction with the topology discovery techniques presented in this thesis, can help in determining configurations that allow to achieve arbitrary traffic engineering requirements in a deterministic way, in contrast with the usually adopted trial-and-error approach.

In a large scale environment like the Internet, the propagation of route announcements is influenced by the interaction of routing policies deployed at different Autonomous Systems. Typically, routers limit the propagation of routing information either because of explicit filters or because they are configured to prefer a certain AS-path when multiple choices are available. This thesis proposes novel techniques that, based on the probing primitives we use to get topological information, allow to determine the feasibility of AS-paths that cannot be observed in stable routing states and to establish which is the level of preference associated with different AS-paths having the same length. We prove the effectiveness of

these techniques by applying them to discover the policies used to route both IPv6 and IPv4 prefixes on the live Internet.

Interactions among routing policies may also lead to abnormal and unforeseen behaviors like permanent routing oscillations. Some models have already been introduced to study the characteristics of stable BGP configurations. In this thesis we propose an alternative model to formally describe the conditions that ensure stable routing and prove some properties of stable configurations. The model can be used to analyze an instance of routing system in order to single out conflicting policies potentially leading to permanent oscillations.

Working with solid theoretical models is usually very helpful but yet not enough to understand the dynamics of a routing system. Experimentation too may not be feasible because the necessary devices may not be available or the required tests may involve disruptive actions on the network under consideration. Yet, it is often very useful both for operators and for researchers to be able to see device configurations in action before deploying them in a live network scenario. Emulation systems effectively fill this gap. This thesis presents a short survey of network emulation systems and describes in detail Netkit, a lightweight emulator that allows to easily setup and distribute complex network configurations. Besides being an effective instrument to support teaching of computer networks, Netkit has also proved itself to be helpful in testing the configuration of large scale real world networks. Netkit also comes with a set of ready to use virtual laboratories that allow to immediately experiment with specific case studies. It fully installs and runs in user space and provides users with a familiar environment consisting of a Debian based Linux box and well known routing software and routing tools.

Open Problems

WHILE this thesis introduces novel models and presents interesting results that enable a deeper understanding of BGP routing, there are still some open issues which require further investigation.

Topology discovery techniques based on active probing by BGP announcements could be improved by introducing a strong theoretical foundation driving the exploration process. This would allow to optimize the number of announcements required to discover portions of the Internet and would help in better estimating the limits of our discovery methodology, by discriminating between what can be discovered and what cannot be. Probes may then be combined with network measurements to support *what-if* case analysis of network performance under alternate routing configurations. These considerations also hold for the techniques for determining the feasibility or level of preference of an AS-path.

A natural evolution of the analysis of Registry information to extract BGP peerings would be to focus on routing policies expressed in RPSL. Because of its primary goal of supporting the definition of consistent routing policies, the Internet Routing Registry is a very rich source of policy information. Mining such information may be difficult because of the existence of out-of-date and inconsistent chunks of data in the IRR; yet, the knowledge of routing policies would allow to better understand how network prefixes would be propagated in the Internet before actually announcing them. In principle, routing policies could also be implemented in an emulation system in order to obtain a virtualized instance of (a portion of) the Internet.

The analysis of methods for inferring the commercial relationships between Autonomous Systems can be further extended to take into account recently introduced algorithms and validate their results. Also consider that under the valley-free approach an instance of network may have multiple valid relationship assignments, while inference algorithms usually return just one of them. It would be interesting to investigate the whole space of solutions in order to estimate the degrees of

freedom of an assignment. Once a realistic relationship assignment is available, it could be used as a starting point to compute a classification of the Autonomous Systems into hierarchical levels. While some methodologies for computing a hierarchy have already been proposed, it is still unclear which is the correct classification criterion.

Also the model for achieving traffic engineering by prepending presented in this thesis leaves room for improvements. First of all, both the model and the algorithms for computing the optimal amount of prepending should be tested on a case study to prove their effectiveness. Then, the model itself could be extended to take into account other traffic engineering requirements. It would also be important to determine bounds on the algorithmic complexity of computing the optimal amount of prepending. Another interesting perspective for future research consists in understanding the evolution of the Internet system when its actors apply traffic engineering in a selfish way, with the goal of maximizing their own profit. Such a study would allow to determine whether adopting a selfish behavior on a large scale would impact Internet's stability.

The model for studying BGP routing oscillations still needs to be extended. While it is able to capture different kinds of stability, it misses a methodology for determining whether a network instance is configured in such a way to induce instabilities. Moreover, the correlation between the configuration of a network and the reachability of stable states has still to be investigated.

The Netkit emulation environment currently provides an extensive set of tools and ready to use virtual labs that allow to quickly and effectively experiment with complex network scenarios. The improvements it is susceptible of are mainly of technical nature. Essentially, plans for future development include updates of the filesystem and kernel of virtual machines in order to improve stability and include more recent releases of routing software as well as other tools (for example, related to network security). The possibility to introduce support for physical characteristics of network links (delay, packet loss, reordering, etc.) is also taken into consideration. Another possible improvement would be to simplify the deployment of virtual networks distributed on multiple real hosts. This can be achieved by integration with existing tools for building distributed networks and would bring benefits in terms of scale and complexity of the emulated network.

Acknowledgments

There are a lot of people to whom I owe something more than a simple list of thanks.

Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia acted as irreplaceable guides through the whole PhD, helped me in understanding what to do and what not to, gave me plenty of precious suggestions, and supported me during my research activity; and I am not mentioning their ability in creating a group of people that work so well together.

Lorenzo Colitti has been a clever partner and an effective leader during some of the most difficult phases of the research activity.

Daniel Karrenberg and Henk Uijterwaal gave me the opportunity to share knowledge and experience with people that coordinate and manage routing resources in the Internet.

Tiziana Refice and I collaborated on some of the most important research matters of my studies, and she provided significant support in arranging some travels abroad.

Stefano Pettini improved the interface of Netkit with the idea of organizing the labs as a hierarchy of directories. Configuration options have been improved since then, but we owe the original idea and implementation of the ltools to him. Fabio Ricci contributed with the idea and implementation of the self testing procedure of Netkit labs.

Several useful discussions and exchanges of ideas about different topics have taken place with Guido Drovandi, Gabriele Barbagallo, Andrea Capaldo, and Giacomo Masseroni.

A lot of thanks go to the reviewers for their willingness in going through this thesis and for their kindness in providing useful hints for making it better.

There are many other people from whom I got technical and methodological support. I really acknowledge their help despite the fact that they are not mentioned here.

Substantial moral support (and you cannot make it through a PhD without) came from several other people who I am not mentioning here but are clearly impressed in my heart and to whom I am immensely grateful.

Some of the material presented in this thesis is based on results from published works [140, 66, 67, 131, 139, 130, 65, 141].

Bibliography

[1] NCTUns 1.0. *IEEE Network Magazine*, 17(4), Jul 2003. Appeared in the column "Software Tools for Networking".

[2] Aaron Klingaman, Mark Huang, Steve Muir, and Larry Peterson. PlanetLab Core Specification 4.0. Technical Report PDN–06–032, PlanetLab Consortium, Jun 2006.

[3] Advanced Micro Devices. Introducing AMD Virtualization™. `http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_8826_14287,00.html`.

[4] Albert-László Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[5] Albert-László Barabási. Linked: The New Science of Networks. *Journal of Artificial Societies and Social Simulation*, 6(2), 2003.

[6] Albert-László Barabási, Réka Albert, H. Jeong, and G. Bianconi. Power-Law Distribution of the World Wide Web. *Science*, 287(5461), Mar 2000.

[7] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostic, Jeff Chase, and David Becker. Scalability and Accuracy in a Large-Scale Network Emulator. *ACM SIGOPS Operating Systems Review*, 36(SI):271–284, 2002.

[8] Andrea Bittau and Mark Handley. Decoupling Policy from Protocols. Draft paper, Feb 2006.

[9] Andrea Carmignani, Giuseppe Di Battista, Walter Didimo, Francesco Matera, and Maurizio Pizzonia. Visualization of the High Level

Structure of the Internet with Hermes. *Journal of Graph Algorithms and Applications*, 6(3):281–311, 2002.

[10] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. *ACM SIGCOMM Computer Communication Review*, 36(4):3–14, Sep 2006.

[11] ARIN. American Registry for Internet Numbers. `http://www.arin.net/`.

[12] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang. Collecting the internet AS-level topology. *ACM SIGCOMM Computer Communication Review*, 35(1):53–61, 2005.

[13] Boston University. BRITE Internet Topology Generator. `http://www.cs.bu.edu/brite/`.

[14] Brad Marshall. User Mode Linux, VMWare and Wine – Virtual Machines under Linux, Apr 2003. `http://quark.humbug.org.au/publications/linux/uml.pdf`.

[15] Bradley Huffaker, Daniel Plummer, David Moore, and kc claffy. Topology Discovery by Active Probing. In *Proc. Symposium on Applications and the Internet (SAINT 2002)*, page 90. IEEE Computer Society, Jan 2002.

[16] Bradley Huffaker, Evi Nemeth, and kc claffy. Otter: A General-purpose Network Visualization Tool. In *Proc. International Networking Conference (INET 1999)*, Jun 1999.

[17] Bruno Quoitin. C-BGP. `http://cbgp.info.ucl.ac.be/`.

[18] Bruno Quoitin. CBGP – A new approach to BGP simulation. E-Next Advanced two-day course on Interdomain Routing with BGP4, Nov 2004. `http://cbgp.info.ucl.ac.be/downloads/cbgp-1.pdf`, `http://cbgp.info.ucl.ac.be/downloads/cbgp-2.pdf`.

[19] Bruno Quoitin, Cristel Pelsser, Louis Swinnen, Olivier Bonaventure, and Steve Uhlig. Interdomain Traffic Engineering with BGP. *IEEE Communications Magazine*, 41(5):122–128, May 2003.

[20] Bruno Quoitin and Steve Uhlig. Modeling the Routing of an Autonomous System with C-BGP. *IEEE Network*, 19(6), Nov 2005.

[21] Bruno Quoitin, Steve Uhlig, Cristel Pelsser, and Olivier Bonaventure. Internet Traffic Engineering Techniques. Technical Report Infonet-2002-05, Apr 2002.

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

[23] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546, Nov 1993.

[24] C. Villamizar, R. Chandra, and R. Govindan. BGP Route Flap Damping. RFC 2439, Nov 1998.

[25] CAIDA. Reverse Traceroute and Looking Glass servers in the World. `http://www.caida.org/analysis/routing/reversetrace/`.

[26] CAIDA. Skitter. `http://www.caida.org/tools/measurement/skitter/`.

[27] CAIDA. Skitter Topology Monitors' Status. `http://sk-status.caida.org/cgi-bin/main.pl?mode=status`.

[28] Cengiz Alaettinoglu, Ramesh Govindan, David Kessens, and WeeSan Lee. Providing IRR Consistency. 41st Internet Engineering Task Force (IETF), Apr 1998.

[29] Christos Papadimitriou. Algorithms, Games, and the Internet. In *Proc. 33rd annual ACM Symposium on Theory of Computing (STOC 2001)*, pages 749–753. ACM Press, 2001.

[30] Cisco Systems, Inc. BGP Best Path Selection Algorithm. `http://www.cisco.com/en/US/tech/tk365/technologies_tech_note09186a0080094431.shtml`.

[31] University of Roma Tre Computer Networks Research Group. BGP Probing. `http://www.dia.uniroma3.it/~compunet/bgp-probing/`.

[32] University of Roma Tre Computer Networks Research Group. BGPlay. `http://bgplay.routeviews.org/bgplay/`.

[33] University of Roma Tre Computer Networks Research Group. BGPlay. http://www.ris.ripe.net/bgplay/.

[34] University of Roma Tre Computer Networks Research Group. Hermes. http://tocai.dia.uniroma3.it/~hermes.

[35] University of Roma Tre Computer Networks Research Group. Netkit. http://www.netkit.org.

[36] University of Roma Tre Computer Networks Research Group. Netkit ready to use Labs. http://www.netkit.org/labs.html.

[37] University of Roma Tre Computer Networks Research Group. Netkit related publications. http://www.netkit.org/publications.html.

[38] University of Roma Tre Computer Networks Research Group. NetML. http://www.dia.uniroma3.it/~compunet/netml/.

[39] University of Roma Tre Computer Networks Research Group. RPSL Analysis Service. http://tocai.dia.uniroma3.it/~irr_analysis/.

[40] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed Internet Routing Convergence. In *Proc. ACM SIGCOMM 2000*, pages 175–187, Sep 2000.

[41] Craig Labovitz, Abha Ahuja, Roger Wattenhofer, and Venkatachary Srinivasan. The Impact of Internet Policy and Topology on Delayed Routing Convergence. In *Proc. IEEE INFOCOM 2001*, pages 537–546, Apr 2001.

[42] Curtis Villamizar. Some Practical Advice on Using the IRR, 1998. http://www.isc.org/sw/IRRToolSet/documentation/advice.pdf.gz.

[43] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and Principles of Internet Traffic Engineering. RFC 3272, May 2002.

[44] D. Meyer, J. Schmitz, C. Orange, M. Prior, and C. Alaettinoglu. Using RPSL in Practice. RFC 2650, Aug 1999.

[45] Daniel Karrenberg, Gerard Ross, Paul Wilson, and Leslie Nobile. Development of the Regional Internet Registry System. *Internet Protocol Journal*, 4(4):17–29, Dec 2001.

[46] Debian. APT Howto.
http://www.debian.org/doc/manuals/apt-howto/.

[47] Friedrich Alexander Universität Erlangen-Nürnberg Department of Computer Science. FAUmachine Project (formerly UMLinux).
http://www3.informatik.uni-erlangen.de/Research/FAUmachine/.

[48] University of California, San Diego Department of Computer Science. ModelNet. http://modelnet.ucsd.edu/.

[49] Purdue University Department of Computer Sciences. vBET: a VM-Based Education Testbed.
http://www.cs.purdue.edu/homes/jiangx/vBET/.

[50] University of Zagreb Department of Telecommunications. IMUNES – An Integrated Multiprotocol Network Emulator/Simulator.
http://www.tel.fer.hr/imunes/.

[51] Fabrice Bellard. QEMU Open Source Processor Emulator.
http://www.qemu.org/, http://fabrice.bellard.free.fr/qemu/.

[52] Fabrizio Ciacchi. Linux in Linux. Linux Magazine, May 2005.

[53] Fermín Galán and David Fernández. VNUML: Una Herramienta de Virtualización de Redes Basada en Software Libre. In *Proc. Open Source International Conference 2004*, pages 35–41, Feb 2004. In Spanish.

[54] Fermín Galán, David Fernández, Javier Ruiz, Omar Walid, and Tomás de Miguel. Use of Virtualization Tools in Computer Network Laboratories. In *Proc. 5th International Conference on Information Technology Based Higher Education and Training (ITHET 2004)*, pages 209–214, Jun 2004.

[55] G. Malkin. RIP Version 2. RFC 2453, Nov 1998.

[56] Geoff Huston. BGP Reports. http://bgp.potaroo.net/.

[57] Geoff Huston. *ISP Survival Guide: Strategies for Running a Competitive ISP*. Paperback, New York, NY, 1998.

[58] Geoff Huston. Interconnection, Peering and Settlements – Part I. *Internet Protocol Journal*, 2(1), Mar 1999.

[59] Geoff Huston. Interconnection, Peering and Settlements – Part II. *Internet Protocol Journal*, 2(2), Jun 1999.

[60] Georgos Siganos and Michalis Faloutsos. Analyzing BGP Policies: Methodology and Tool. In *Proc. IEEE INFOCOM 2004*. IEEE, Mar 2004.

[61] Georgos Siganos and Michalis Faloutsos. Nemecis: A Tool to Analyze the Internet Routing Registries. RIPE Meeting 48, May 2004.

[62] Gerd Stolpmann. UMLMON. `http://www.gerd-stolpmann.de/buero/umlmon.html.en`.

[63] Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Archives of BGP Updates: Integration and Visualization. In *Proc. International Workshop on Inter-domain Performance and Simulation*, pages 123–129, Feb 2003.

[64] Giuseppe Di Battista, Maurizio Patrignani, and Maurizio Pizzonia. Computing the Types of the Relationships between Autonomous Systems. In *Proc. IEEE INFOCOM 2003*, volume 1, pages 156–165. IEEE, Apr 2003.

[65] Giuseppe Di Battista, Maurizio Patrignani, Maurizio Pizzonia, and Massimo Rimondini. Towards Optimal Prepending for Incoming Traffic Engineering. In *Proc. 3rd International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe 2005)*, Mar 2005.

[66] Giuseppe Di Battista, Tiziana Refice, and Massimo Rimondini. How to Extract BGP Peering Information from the Internet Routing Registry. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data (MineNet 2006)*, pages 317–322. ACM Press, 2006.

[67] Giuseppe Di Battista, Tiziana Refice, and Massimo Rimondini. How to Extract BGP Peering Information from the Internet Routing Registry. Technical Report RT-DIA-108-2006, University of Roma Tre, Dept. of Computer Science and Automation, May 2006. `http://dipartimento.dia.uniroma3.it/ricerca/rapporti/rapporti.php`.

[68] Hao Wang, Haiyong Xie, Yang Richard Yang, Avi Silberschatz, Li Erran Li, and Yanbin Liu. On the Stability of Rational, Heterogeneous

Interdomain Route Selection. In *Proc. 13th IEEE International Conference on Network Protocols (ICNP 2005)*, pages 40–52. IEEE Computer Society, 2005.

[69] Hao Wang, Haiyong Xie, Yang Richard Yang, Avi Silberschatz Silberschatz, Li Erran Li, and Yanbin Liu. Stable Egress Route Selection for Interdomain Traffic Engineering: Model and Analysis. In *Proc. 13th IEEE International Conference on Network Protocols (ICNP 2005)*, pages 16–29. IEEE Computer Society, 2005.

[70] Hao Wang, Haiyong Xie, Yang Richard Yang, Li Erran Li, Yanbin Liu, and Avi Silberschatz. On Stable Route Selection for Interdomain Traffic Engineering: Models and Analysis. Technical Report YALEU/DCS/TR-1316, Computer Science Department, Yale University, Feb 2005.

[71] Henk Uijterwaal, Antony Antony, and Daniel Karrenberg. RIS Observations. RIPE Meeting 38, Ja 2001.

[72] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag New York, Inc., 1987.

[73] Hongsuda Tangmunarunkit, Ramesh Govindan, Scott Shenker, and Deborah Estrin. The Impact of Routing Policy on Internet Paths. In *Proc. IEEE INFOCOM 2001*, pages 736–742, Apr 2001.

[74] Hongwei Zhang, Anish Arora, and Zhijun Liu. G-BGP: Stable and Fast Convergence of the Border Gateway Protocol. Technical Report OSU-CISRC-6/03-TR36, Ohio State University, Jun 2003.

[75] Hongwei Zhang, Anish Arora, and Zhijun Liu. A Stability-Oriented Approach to Improving BGP Convergence. In *Proc. 23rd IEEE International Symposium on Reliable Distributed Systems (SRDS 2004)*, pages 90–99. IEEE Computer Society, 2004.

[76] Hyunseok Chang, Ramesh Govindan, Sugih Jamin, Scott J. Shenker, and Walter Willinger. Towards Capturing Representative AS-level Internet Topologies. *Computer Networks*, 44(6):737–755, 2004.

[77] Ian Pratt. Xen and the Art of Virtualization. Ottawa Linux Symposium 2004, 2004.

[78] Ian Pratt. Xen 3.0 and the Art of Virtualization. Ottawa Linux Symposium 2005, 2005.

[79] Ian Pratt. Xen and the Art of Virtualization. Ottawa Linux Symposium 2006, 2006.

[80] Innotek. VirtualBox. `http://www.virtualbox.org/`.

[81] Intel. Intel®Virtualization Technology. `http://www.intel.com/technology/virtualization/index.htm`.

[82] International Computer Science Institute, Berkeley, California. XORP Open Source IP Router. `http://www.xorp.org/`.

[83] Internet Systems Consortium. Internet Routing Registry Toolset. `http://www.isc.org/index.pl?/sw/IRRToolSet/`.

[84] Ivan Santarelli and Alexandra Bellogini. NetML – Network Markup Language. RIPE Meeting 47, Feb 2004.

[85] J. Moy. OSPF Version 2. RFC 2328, Apr 1998.

[86] Jay Chen, Diwaker Gupta, Kashi V. Vishwanath, Alex C. Snoeren, and Amin Vahdat. Routing in an Internet-Scale Network Emulator. In *Proc. 12th IEEE Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2004)*, pages 275–283. IEEE Computer Society, 2004.

[87] Jay Lepreau. Emulab: Recent Work, Ongoing Work. Talk at DETER Lab Community Meeting, Jan 2006.

[88] Jeff Dike. A User-Mode Port of the Linux Kernel. `http://user-mode-linux.sourceforge.net/als2000/index.html`.

[89] Jeff Dike. A User-Mode Port of the Linux Kernel. In *Proc. 4th Annual Linux Showcase & Conference*, pages 63–72, Oct 2000.

[90] Jeff Dike. User-Mode Linux. Talk at the 2001 Ottawa Linux Symposium, Jul 2001.

[91] Jeff Dike. Double your Fun with User-Mode Linux, Nov 2004.

[92] Jeff Dike. *User Mode Linux*. Prentice Hall, Apr 2006.

[93] Jeroen van der Ham, Freek Dijkstra, Franco Travostino, Hubertus Andree, and Cees de Laat. Using RDF to Describe Networks. *Future Generation Computer Systems, Feature topic iGrid 2005*, 2006. http://staff.science.uva.nl/~vdham/research/publications/0510-NetworkDescripti

[94] Jeroen van der Ham, Paola Grosso, and Cees de Laat. Semantics for Hybrid Networks Using the Network Description Language. Poster at the International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2006), Mar 2006. http://staff.science.uva.nl/~vdham/research/publications/0603-NetworkDescripti

[95] Jeroen van der Ham, Paola Grosso, Ronald van der Pol, Andree Toonk, and Cees de Laat. Using the Network Description Language in Optical Networks. May 2006. Accepted at the IEEE Integrated Management Conference 2007.

[96] Jianhong Xia. Weird ASPATH in update message. RIPE Routing Working Group list archive, Jun 2002. http://www.ripe.net/maillists/ncc-archives/ris-users/2002/msg00044.html.

[97] João Luís Sobrinho. Network Routing with Path Vector Protocols: Theory and Applications. In *Proc. ACM SIGCOMM 2003*, pages 49–60. ACM Press, 2003.

[98] João Luís Sobrinho. An Algebraic Theory of Dynamic Network Routing. *IEEE/ACM Transactions on Networking*, 13(5):1160–1173, 2005.

[99] Joachim Schmitz, Engin Gunduz, Shane Kerr, Andrei Robachevsky, and Joao Luis Silva Damas. Routing Registry Consistency Check. RIPE Document 201, Dec 2001.

[100] John W. Stewart. *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley, Reading, MA, 1999.

[101] Jorge A. Cobb, Mohamed G. Gouda, and Ravi Musunuri. A Stabilizing Solution to the Stable Path Problem. *LNCS*, 2704:169–183, 2003.

[102] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent Route Oscillations in Inter-domain Routing. *Elsevier Computer Networks*, 32(1):1–16, 2000.

[103] kc claffy. Internet Measurement and Data Analysis: Topology, Workload, Performance and Routing Statistics. National Academy of Engineering (NAE) Workshop, Mar 1999.

[104] kc claffy. CAIDA: Visualizing the Internet. Internet Computing Online, Jan 2001.

[105] kc claffy and Sean McCreary. Internet Measurement and Data Analysis: Passive and Active Measurement. American Statistical Association, Aug 1999.

[106] kc claffy, Tracie E. Monk, and Daniel McRobb. Internet Tomography. Nature, Web Matters column, Jan 1999.

[107] Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfield, and Mark Williamson. Safe Hardware Access with the Xen Virtual Machine Monitor. In *Proc. 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS 2004)*, Oct 2004.

[108] Kunihiro Ishiguro. GNU Zebra Routing Software. `http://www.zebra.org/`.

[109] Kuthonuzo Luruo and Shashank Khanvilkar. Virtual Networking with User-Mode Linux. Linux for You – Pro, Mar 2005.

[110] Kyron. Linux Virtual Server. `http://www.kyron.it/virtual_server.asp`.

[111] L. Blunk, J. Damas, F. Parent, and A. Robachevsky. Routing Policy Specification Language next generation (RPSLng). RFC 4012, Mar 2005.

[112] L. Daigle. WHOIS Protocol Specification. RFC 3912, Sep 2004.

[113] Lakshminarayanan Subramanian, Matthew Caesar, Cheng Tien Ee, Mark Handley, Zhuoqing Morley Mao, Scott Shenker, and Ion Stoica. Towards a Next Generation Inter-domain Routing Protocol. In *Proc. 3rd Workshop on Hot Topics in Networks (HOTNETS-III)*, Nov 2004.

[114] Lakshminarayanan Subramanian, Matthew Caesar, Cheng Tien Ee, Mark Handley, Zhuoqing Morley Mao, Scott Shenker, and Ion Stoica. HLP: A Next Generation Inter-domain Routing Protocol. In *Proc. ACM SIGCOMM 2005*, pages 13–24. ACM Press, 2005.

[115] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. In *Proc. IEEE INFOCOM 2002*, Jun 2002.

[116] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points, Data Sets [discontinued], 2002. `http://www.cs.berkeley.edu/~sagarwal/research/BGP-hierarchy/`.

[117] Larry Peterson, Steve Muir, Timothy Roscoe, and Aaron Klingaman. PlanetLab Architecture: An Overview. Technical Report PDN–06–031, PlanetLab Consortium, May 2006.

[118] Larry Peterson and Timothy Roscoe. The Design Principles of PlanetLab. *ACM SIGOPS Operating Systems Review*, 40(1):11–16, 2006.

[119] Lili Qiu, Yang Richard Yang, Yin Zhang, and Scott Shenker. On Selfish Routing in Internet-Like Environments. *IEEE/ACM Transactions on Networking*, 14(4), Aug 2006.

[120] Linode.com. Virtual Private Server. `http://linode.com/`.

[121] Lixin Gao. On Inferring Autonomous System Relationships in the Internet. In *Proc. IEEE Global Internet Symposium 2000*, Nov 2000.

[122] Lixin Gao. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, Dec 2001.

[123] Lixin Gao and Feng Wang. The Extent of AS Path Inflation by Routing Policies. In *Proc. IEEE Global Internet Symposium 2002*, Nov 2002.

[124] Lixin Gao and Jennifer Rexford. Stable Internet Routing without Global Coordination. In *Proc. ACM SIGMETRICS 2000*, pages 307–317. ACM Press, 2000.

[125] Lixin Gao, Timothy G. Griffin, and Jennifer Rexford. Inherently Safe Backup Routing with BGP. In *Proc. IEEE INFOCOM 2001*, pages 547–556, Apr 2001.

[126] Lorenzo Colitti. "AS-set stuffing" experiments using RIS beacons. RIPE Routing Working Group mailing list, Feb 2005. http://www.ripe.net/ripe/maillists/archives/routing-wg/2005/msg00021.html

[127] Lorenzo Colitti. Heads up: Long AS-sets announced in the next few days. North American Network Operators Group (NANOG) mailing list, Mar 2005. http://www.merit.edu/mail.archives/nanog/2005-03/msg00029.html.

[128] Lorenzo Colitti. Heads up: Long AS-sets announced in the next few days. RIPE NCC RIS Users mailing list, Mar 2005. https://www.ripe.net/maillists/ncc-archives/ris-users/2005/msg00017.html

[129] Lorenzo Colitti, Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Visualizing interdomain routing with BGPlay. *Journal of Graph Algorithms and Applications*, 9(1):117–148, Nov 2005.

[130] Lorenzo Colitti, Giuseppe Di Battista, Maurizio Patrignani, Maurizio Pizzonia, and Massimo Rimondini. Active BGP Probing. Technical Report RT-DIA-102-2005, University of Roma Tre, Dept. of Computer Science and Automation, Nov 2005. http://dipartimento.dia.uniroma3.it/ricerca/rapporti/rapporti.php.

[131] Lorenzo Colitti, Giuseppe Di Battista, Maurizio Patrignani, Maurizio Pizzonia, and Massimo Rimondini. Investigating Prefix Propagation through Active BGP Probing. In *Proc. IEEE Symposium on Computers and Communications (ISCC 2006)*, pages 497–504. IEEE Computer Society, 2006.

[132] Louis Swinnen, Sébastien Tandel, S. Uhlig, Bruno Quoitin, and Olivier Bonaventure. An Evaluation of BGP-based Traffic Engineering Techniques. Technical Report Infonet-2002-10, Dec 2002.

[133] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, Apr 1999.

[134] Mark Handley. Proposal to Develop an Extensible Open Router Platform. Initial project proposal, Nov 2000.

[135] Mark Handley. New BGP decision process implementation, Xorp-hackers mailing list, Oct 2003. http://mailman.icsi.berkeley.edu/pipermail/xorp-hackers/2003-October/000

[136] Mark Handley, Eddie Kohler, Atanu Ghosh, Orion Hodson, and Pavlin Radoslavov. Designing Extensible IP Router Software. In *Proc. 2ns USENIX Symposium on Networked Systems Design and Implementation (NSDI 2005)*, May 2005.

[137] Mark Handley, Orion Hodson, and Eddie Kohler. XORP: An Open Platform for Network Research. *ACM SIGCOMM Computer Communication Review*, 33(1):53–57, 2003.

[138] Marko Zec. Implementing a Clonable Network Stack in the FreeBSD Kernel. In *Proc. 2003 USENIX Annual Technical Conference*, Jun 2003.

[139] Massimo Rimondini. Emulating Computer Networks with Netkit. Tutorial at the 4th International Workshop on Internet Performance, Simulation, Monitoring and Measurement (IPS-MoMe 2006), Feb 2006.

[140] Massimo Rimondini. Emulation of Computer Networks with Netkit. Technical Report RT-DIA-113-2007, University of Roma Tre, Jan 2007.

[141] Massimo Rimondini, Maurizio Pizzonia, Giuseppe Di Battista, and Maurizio Patrignani. Algorithms for the Inference of the Commercial Relationships between Autonomous Systems: Results Analysis and Model Validation. In *Proc. 2nd International Workshop on Inter-Domain Performance and Simulation (IPS 2004)*, Mar 2004.

[142] Merit Network Inc. List of Routing Registries. `http://www.irr.net/docs/list.html`.

[143] Merit Network, Inc. Overview of the IRR. `http://www.irr.net/docs/overview.html`.

[144] Merit Network, Inc. Routing Assets Database (RADB). `ftp://ftp.radb.net/radb/dbase/`.

[145] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-Law Relationships of the Internet Topology. In *Proc. ACM SIGCOMM 1999*, pages 251–262. ACM Press, 1999.

[146] Microsoft. Virtual PC 2004. `http://www.microsoft.com/italy/windows/virtualpc/default.mspx`.

[147] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP Topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

[148] Neil Spring, Ratul Mahajan, and Thomas Anderson. Quantifying the Causes of Path Inflation. In *Proc. ACM SIGCOMM 2003*, Aug 2003.

[149] University of Cambridge Networks and Operating Systems Group. XEN. `http://www.cl.cam.ac.uk/research/srg/netos/xen/`.

[150] Nick Feamster, Hari Balakrishnan, and Jennifer Rexford. Some Foundational Problems in Interdomain Routing. In *Proc. 3rd Workshop on Hot Topics in Networks (HOTNETS-III)*, Nov 2004.

[151] Nick Feamster, Jay Borkenhagen, and Jennifer Rexford. Guidelines for Interdomain Traffic Engineering. *ACM SIGCOMM Computer Communication Review*, 33(5):19–30, Oct 2003.

[152] Olivier Bonaventure. Internet Traffic Engineering Techniques. Tutorial at ICNP 2002, Nov 2002. `http://suraj.lums.edu.pk/~te/mpls/icnp2002-notes.pdf`.

[153] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022, Jan 2001.

[154] Paolo Giarrusso. SKAS patches and UML updates. `http://www.user-mode-linux.org/~blaisorblade/`.

[155] Paolo Giarrusso. UML Utilities. `http://www.user-mode-linux.org/~blaisorblade/uml-utilities/`.

[156] Parallels. Parallels Workstation. `http://www.parallels.com/`.

[157] Paul Albitz. *DNS and BIND*. O'Reilly & Associates, Inc., 2001.

[158] Paul Barford, Azer Bestavros, John Byers, and Mark Crovella. On the Marginal Utility of Network Topology Measurements. In *Proc. 1st ACM SIGCOMM Workshop on Internet Measurement (IMW 2001)*, pages 5–17. ACM Press, 2001.

[159] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating System Principles (SOSP 2003)*, Oct 2003.

[160] Pio Baake and Thorsten Wichmann. On the Economics of Internet Peering. *Netnomics*, 1(1):89–105, 1999.

[161] PlanetLab Consortium. PlanetLab. `http://www.planet-lab.org/`.

[162] Priya Mahadevan, Adolfo Rodriguez, David Becker, and Amin Vahdat. MobiNet: A Scalable Emulation Infrastructure for Ad hoc and Wireless Networks. In *Proc. 2005 Workshop on Wireless Traffic Measurements and Modeling (WiTMeMo 2005)*, pages 7–12. USENIX Association, 2005.

[163] Priya Mahadevan, Dmitri Krioukov, Marina Fomenkov, Xenofontas Dimitropoulos, kc claffy, and Amin Vahdat. The Internet AS-Level Topology: Three Data Sources and One Definitive Metric. *ACM SIGCOMM Computer Communication Review*, 36(1):17–26, 2006.

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

[165] Ratul Mahajan, Neil Spring, David Wetherall, and Thomas Anderson. Inferring Link Weights using End-to-End Measurements. In *Proc. Internet Measurement Workshop (IMW 2002)*, Nov 2002.

[166] Renzo Davoli. VDE: Virtual Distributed Ethernet. `http://sourceforge.net/projects/vde/`.

[167] Renzo Davoli. Virtual Square. `http://www.virtualsquare.org/`.

[168] Renzo Davoli. VDE: Virtual Distributed Ethernet. Technical Report UBLCS-2004-12, University of Bologna, Jun 2004.

[169] Renzo Davoli. VDE: Virtual Distributed Ethernet. In *Proc. 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2005)*, pages 213–220. IEEE Computer Society, 2005.

[170] RIPE NCC. General Routing Registry Consistency Check Report. `http://rrcc.ripe.net/RRCC_general_report.html`.

[171] RIPE NCC. RIPE Routing Registry database. `ftp://ftp.ripe.net/ripe/dbase/ripe.db.gz`.

[172] RIPE NCC. RIS Routing Beacons. http://www.ripe.net/ris/docs/beaconlist.html.

[173] RIPE NCC. Routing Information Service. http://www.ripe.net/projects/ris/.

[174] RIPE NCC. Routing Registry Consistency Check (RRCC) Project. http://www.ripe.net/projects/rrcc/.

[175] Rocky K. C. Chang and Michael Lo. Inbound Traffic Engineering for Multihomed ASes Using AS Path Prepending. In *Proc. Network Operations and Management Symposium (NOMS 2004)*, Apr 2004.

[176] S. Y. Wang. Using the NCTUns 2.0 Network Simulator and Emulator to Facilitate Network Researches. In *Proc. 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2006)*, Mar 2006.

[177] S. Y. Wang, C. L. Chou, C. H. Huang, C. C. Hwang, Z. M. Yang, C. C. Chiou, and C. C. Lin. The Design and Implementation of the NCTUns 1.0 Network Simulator. *Computer Networks*, 42(2):175–197, 2003.

[178] S. Y. Wang and H. T. Kung. A New Methodology for Easily Constructing Extensible and High-fidelity TCP/IP Network Simulators. *Computer Networks*, 40(2):257–278, 2002.

[179] Sandro Doro. Netkit4TIC Virtual Laboratory. http://www.tic.fdns.net/tic/html/lab.html.

[180] Shane Kerr. RIPE Database Inconsistencies. RIPE Meeting 43, Sep 2002.

[181] Shie-Yuan Wang and Kuo-Chiang Liao. Innovative Network Emulations Using the NCTUns Tool. *Computer Networking and Networks*, pages 157–187, 2006.

[182] SimReal Inc. NCTUns Network Simulator and Emulator. http://nsl10.csie.nctu.edu.tw/.

[183] Suman Banerjee, Timothy G. Griffin, and Marcelo Pias. The Interdoman Connectivity of PlanetLab Nodes. In *Proc. Passive & Active Measurement Workshop (PAM 2004)*, Apr 2004.

[184] T. Bates, E. Gerich, L. Joncheray, J-M. Jouanigot, D. Karrenberg, M. Terpstra, and J. Yu. Representation of IP Routing Policies in a Routing Registry (ripe-81++). RFC 1786, Mar 1995.

[185] T. Griffin and G. Huston. BGP Wedgies. RFC 4264, Nov 2005.

[186] Technical University of Madrid (UPM) Telematics Engineering Department. VNUML. `http://jungla.dit.upm.es/~vnuml/`.

[187] Thomas Erlebach, Alexander Hall, and Thomas Schank. Classifying Customer-Provider Relationships in the Internet. In *Proc. International Conference on Communications and Computer Networks (CCN 2002)*, pages 538–545. IASTED, Nov 2002.

[188] Thomas Kernen. A comprehensive list of Looking Glasses. `http://www.traceroute.org/`.

[189] Timothy G. Griffin, Aaron D. Jaggard, and Vijay Ramachandran. Design Principles of Policy Languages for Path Vector Protocols. In *Proc. ACM SIGCOMM 2003*, pages 61–72. ACM Press, 2003.

[190] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. Policy Disputes in Path-Vector Protocols. In *Proc. 7th International Conference on Network Protocols (ICNP 1999)*, page 21. IEEE Computer Society, 1999.

[191] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Transactions on Networking*, 10(2):232–243, 2002.

[192] Timothy G. Griffin and Gordon Wilfong. An Analysis of BGP Convergence Properties. In *Proc. ACM SIGCOMM 1999*, pages 277–288. ACM Press, 1999.

[193] Timothy G. Griffin and João Luís Sobrinho. Metarouting. *ACM SIGCOMM Computer Communication Review*, 35(4):1–12, 2005.

[194] Timothy Griffin and Gordon T. Wilfong. A Safe Path Vector Protocol. In *Proc. IEEE INFOCOM 2000*, pages 490–499, Mar 2000.

[195] Tony Bates, Elise Gerich, Laurent Joncheray, Jean-Michel Jouanigot, Daniel Karrenberg, Marten Terpstra, and Jessica Yu. Representation of IP Routing Policies in a Routing Registry. RIPE Document: RIPE-181, Oct 1994.

[196] Tony Bates, Jean-Michel Jouanigot, Daniel Karrenberg, Peter Lothberg, and Marten Terpstra. Representation of IP Routing Policies in the RIPE Database. RIPE Document: RIPE-81, Feb 1993.

[197] Universiteit van Amsterdam. NDL – Network Description Language. http://www.science.uva.nl/research/sne/ndl/.

[198] University of Oregon. Route Views Project. http://www.routeviews.org/.

[199] University of Utah. Emulab Network Emulation Testbed. http://www.emulab.net/.

[200] University of Washington. Rocketfuel: An ISP Topology Mapping Engine. http://www.cs.washington.edu/research/networking/rocketfuel/.

[201] W. Simpson. IP in IP Tunneling. RFC 1853, Oct 1995.

[202] Adaptive Networking Software - Avaya. http://support.avaya.com/japple/css/japple?PAGE=Product&temp.productID=2

[203] BGP Advanced Internet Routing Resources. http://www.bgp4.as/.

[204] Bochs. http://bochs.sourceforge.net/.

[205] CAIDA – AS ranking. http://as-rank.caida.org/.

[206] Cooperative Linux. http://www.colinux.org/.

[207] Debian GNU/Linux. http://www.debian.org/.

[208] DOSBox. http://dosbox.sourceforge.net/.

[209] DOSEMU. http://www.dosemu.org/.

[210] EINAR (Einar Is Not a Router) Router Simulator. http://www.isk.kth.se/proj/einar/.

[211] Flow Control Platform - Internap, Inc. http://www.internap.com/product/technology/fcp/page1533.html.

[212] Network stack cloning/virtualization extensions to the FreeBSD kernel. http://www.tel.fer.hr/zec/BSD/vimage/.

[213] GARR - The Italian Academic and Research Network. http://www.garr.it/.

[214] Graphviz Graph Visualization Software. http://www.graphviz.org/.

[215] InternetNews xSP Archives. http://www.internetnews.com/xSP/.

[216] The Internet Routing Registry: History and Purpose. http://www.ripe.net/db/irr.html.

[217] Internet Routing Registry Daemon (IRRd). http://www.irrd.net/.

[218] ISP-Planet ISP-Business Archives. http://www.isp-planet.com/business/index.html.

[219] J-Sim simulation environment - The Ohio State University. http://www.j-sim.org/.

[220] The Linux Kernel Archives. http://www.kernel.org/.

[221] KVM Kernel-based Virtual Machine. http://kvm.sourceforge.net/.

[222] The MLN Project. http://mln.sourceforge.net/.

[223] Nemecis. http://ira.cs.ucr.edu:8080/Nemecis/.

[224] netfilter – Firewalling, NAT, and Packet Mangling for Linux. http://www.netfilter.org/.

[225] Cisco Netflow. http://www.cisco.com/en/US/tech/tk812/tsd_technology_support_protocol_home.html.

[226] Packages installed in Netkit filesystem version F2.2. http://www.netkit.org/download/netkit-filesystem/installed-packages-F2.2.

[227] The Network Simulator – NS-2. http://www.isi.edu/nsnam/ns/.

[228] PearPC PowerPC Architecture Emulator. http://pearpc.sourceforge.net/.

[229] Peer Director - Radware, Inc. http://www.radware.com/content/products/pd/default.asp.

[230] PlanetLab Acceptable Use Policy. http://www.planet-lab.org/php/aup/.

[231] Plex86 x86 Virtual Machine Project. http://plex86.sourceforge.net/, http://www.plex86.org/.

[232] Q – Mac port of QEMU. http://www.kju-app.org/kju/.

[233] Quagga Routing Suite. http://www.quagga.net/.

[234] Routing Registry Consistency Check Scripts. ftp://ftp.ripe.net/ripe/dbase/software/RRCC-0.2.tar.gz.

[235] Scalable Simulation Framework (SSFNet). http://www.ssfnet.org/homePage.html.

[236] Serenity Virtual Station (formerly TwoOSTwo). http://www.serenityvirtual.com/.

[237] TORQUE: Type Of Relationship Quality Evaluation Toolkit. http://www.dia.uniroma3.it/~compunet/torque.

[238] User-mode Linux Kernel. http://user-mode-linux.sourceforge.net/.

[239] UML Utilities. http://user-mode-linux.sourceforge.net/dl-sf.html.

[240] Università degli Studi Roma Tre. http://www.uniroma3.it/.

[241] VINI – A Virtual Network Infrastructure. http://vini-veritas.net/.

[242] Virtuozzo. http://www.swsoft.com/en/products/virtuozzo.

[243] VMware. http://www.vmware.com/.

[244] Win4Lin. http://www.win4lin.com/.

[245] Wine. http://www.winehq.com/.

[246] William B. Norton. Internet Service Providers and Peering. NANOG 19 Meeting, Jun 2000.

[247] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. Building an AS-topology Model that Captures Route Diversity. *ACM SIGCOMM Computer Communication Review*, 36(4):195–206, 2006.

[248] Xenofontas Dimitropoulos, Dmitri Krioukov, Bradley Huffaker, kc claffy, and George Riley. Inferring AS Relationships: Dead End or Lively Beginning? *LNCS*, 3503:113–125, 2005.

[249] Xenofontas Dimitropoulos, Dmitri Krioukov, Bradley Huffaker, kc claffy, and George Riley. Inferring AS Relationships: Dead End or Lively Beginning? In *Proc. 4th International Workshop on Efficient and Experimental Algorithms (WEA 2005)*, May 2005.

[250] Xenofontas Dimitropoulos, Dmitri Krioukov, and George Riley. Revisiting Internet AS-level Topology Discovery. In *Proc. Passive & Active Measurement Workshop (PAM 2005)*, Apr 2005.

[251] Xenofontas Dimitropoulos, Dmitri Krioukov, Marina Fomenkov, Bradley Huffaker, Young Hyun, kc claffy, and George Riley. AS Relationships: Inference and Validation. *ACM SIGCOMM Computer Communication Review*, 37(1), 2007. To appear.

[252] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, Jan 2006.

[253] Yang Richard Yang, Haiyong Xie, Hao Wang, Li Erran Li, Yanbin Liu, Avi Silberschatz, and Arvind Krishnamurthy. Stable Route Selection for Interdomain Traffic Engineering. *IEEE Network Magazine – Special Issue on Interdomain Routing*, Nov/Dec 2005.

[254] Zhuoqing Morley Mao. BGP Beacon Info. `http://www.psg.com/~zmao/BGPBeacon.html`.

[255] Zhuoqing Morley Mao, David Johnson, Jennifer Rexford, Jia Wang, and Randy Katz. Scalable and Accurate Identification of AS-Level Forwarding Paths. In *Proc. IEEE INFOCOM 2004*, Mar 2004.

[256] Zhuoqing Morley Mao, Jennifer Rexford, Jia Wang, and Randy H. Katz. Towards an Accurate AS-level Traceroute Tool. In *Proc. ACM SIGCOMM 2003*, pages 365–378. ACM Press, 2003.

[257] Zhuoqing Morley Mao, Ramesh Govindan, George Varghese, and Randy H. Katz. Route Flap Damping Exacerbates Internet Routing Convergence. In *Proc. ACM SIGCOMM 2002*, pages 221–233. ACM Press, Aug 2002.

[258] Zhuoqing Morley Mao, Randy Bush, Timothy G. Griffin, and Matthew Roughan. BGP Beacons. In *Proc. of the 3rd ACM SIGCOMM Conference on Internet Measurement (IMC 2003)*, pages 1–14. ACM Press, 2003.

[259] Zihui Ge, Daniel Ratton Figueiredo, and Sharad Jaiswal. Logical Relationship Inference software. `http://www-net.cs.umass.edu/~ratton/AS/`.

[260] Zihui Ge, Daniel Ratton Figueiredo, Sharad Jaiswal, and Lixin Gao. On the Hierarchical Structure of the Logical Internet Graph. In *Proc. SPIE ITCom 2001*, Aug 2001.