Mechanisms for interdomain Traffic Engineering with LISP

Damien Saucez

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

September 7, 2011

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Mechanisms for Interdomain Traffic Engineering with LISP
by Damien Saucez

ICTEAM
Université catholique de Louvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work was partially supported by the European-funded 027609 AGAVE and 223936 ECODE projects and a Cisco URP grant.

To my grand father

Preamble

Internet is composed of the inter-connection of tens of thousands of independent networks. These networks are called Autonomous Systems (AS) or domains. Some of these ASes are only providing connectivity to the other ASes and are then called Internet Service Providers (ISP). The vast majority of ASes in the Internet are multi-homed and multi-connected [ACK03]. On the one hand, the ASes are connected to the Internet with several ISPs (i.e., multi-homed) in order to preserve their connectivity in case of the failure of one ISP and to reduce their cost by playing the concurrence game. On the other hand, the ASes are connected to the ISPs with several links (i.e., multi-connected) in order to increase their total bandwidth capacity with the Internet. Being multi-homed and multi-connected increases the path diversity with the Internet. However, measurements show that these different paths offer disparate performance [APS04, AAS03, GDZ06, AMS$^+$08, DD06]. Therefore, multi-homing and multi-connectivity enable Autonomous Systems to chose better quality paths over the Internet. Unfortunately, the current Internet design makes difficult to optimize routing to follow the paths that offer the best performance. Indeed, the Border Gateway Protocol (BGP) is the de-facto inter-domain routing protocol in the Internet [RLH06]. BGP is a path vector policy based routing protocol. In BGP, the routes are propagated gradually from AS to AS. If an AS has several routes for a given destination, it can only advertise and use the best one. The choice of the best route follows the BGP Decision Process [RLH06]. The decision process is use to implement the AS policies. Therefore, the originating AS of a route only has a limited control on the way its own routes are propagated in the Internet and path diversity is lost [dLQB06]. In addition, the path-vector nature of BGP and the fact that the Internet is composed of hundreds of thousands routes make BGP suffers from scalability issues. As a result, it is recommended to avoid changing the advertised routes too frequently (not more frequently than once every few days or weeks) [KKK07]. For that reasons, ASes lack of control on the traffic entering them and it is hard to control traffic such that it is optimized for performance.

Despite the inherent weakness of the Internet, we propose techniques to enable *performance based inter-domain incoming traffic engineering* in this thesis. In this thesis, we define the inter-domain incoming traffic of an AS A as any traffic originated by another AS B for which the destination belongs to AS A. Performing inter-domain incoming traffic engineering thus corresponds to being able to control how and where such traffic enters the AS. That being the case, an AS performs performance based inter-domain traffic engineering if it manages to make its inter-domain traffic entering via the paths that optimize its efficiency. Optimizing the efficiency corresponds to ensuring that the traffic performs at its best (e.g., minimize the delay or maximize the bandwidth). However, as BGP does not offer such a capability, we need to introduce new concepts to achieve our goal. For that, we first introduce a system that allows an AS to control its incoming traffic in a scalable way. This system relies on the Locator/ID Separation Protocol (LISP) [FFML10a, Mey08]. However, the current LISP control-plane does not scale well. This is why we propose LISP-Tree, a new scalable and efficient control-plane for LISP. Afterward, we present a system that is able to rank the paths in order to determine which is the best according to any arbitrary performance criterion. We call this system IDIPS for ISP-Driven Informed Path Selection. Finally, we show how to combine LISP, LISP-Tree and IDIPS to achieve performance based inter-domain incoming traffic engineering. At that stage, it is important to notice that our thesis mostly ignores intra-domain traffic engineering or inter-domain outgoing traffic engineering. Indeed, on the one hand, intra-domain traffic engineering is a well known problem with many solutions [FRT02, FT02, FT00]. On the other hand, inter-domain outgoing traffic engineering is already understood, is well supported by BGP and depends on intra-domain traffic engineering [OBFR09, QUP^{+}03, Quo06, UBQ03, CL05, GCLC04].

Road map

To develop our performance based inter-domain incoming traffic engineering techniques, we structure this thesis as follow. We first provide in Chapter 1 some background on how Internet works today and its limitations in term of traffic control.

In Chapter 2 we detail the LISP protocol and its design choices. LISP is a core/edge separation protocol proposed by the industry that separates the IP space in two distinct spaces. On the one hand, the Endpoint IDentifiers (EID) are assigned to the end-hosts and the routers at the edge. The EIDs are not globally routable but hosts within the same edge network belong to the same EID prefix. On the other hand, the Routing LOcators (RLOC) are attached to the core router network interfaces. The interfaces between the core and the edge are also in the RLOC space. The RLOCs are globally routable in the Internet core but not in the edge networks. Because the addressing space is split in two independent parts, a glue must be provided to allow end-hosts that belong to separated networks to communicate. To do so, the routers at the border between the edge and the core run the LISP protocol. LISP is a map-and-encap mechanism. On the one hand, the control plane maps a list of RLOCs to each EID prefix. The list of RLOCs for an EID prefix is the list of IP addresses of the LISP enabled border routers for that prefix. This association is called a mapping and the control-plane is called the mapping system. In Chapter. 3 we show that by manipulating its mappings an AS can easily control its incoming traffic. Indeed, a priority and a weight are associated to each RLOC in the mapping and we show how LISP can provide per-requester mappings. Unfortunately, the current LISP control-plane does not scale well. We propose LISP-Tree in Chapter 4, a new scalable mapping system that is tolerant to faults and configuration errors. LISP-Tree can be troubleshooted and secured easily.

LISP provides path diversity and allows one to control its incoming traffic in an elegant and scalable way when combined with LISP-Tree. With LISP and LISP-Tree it is thus possible to do inter-domain incoming traffic engineering. In the reminder of the thesis we provide a platform called IDIPS that can be used to determine the path performance in a scalable way. Chapter 5, presents IDIPS (ISP-Driven Informed Path Selection). IDIPS assumes that the relative path performance (i.e., path P_1 is faster than path P_2) are more important than the "absolute" performance of the paths. As a result, IDIPS is a request/response service that ranks the paths to highlight the most efficient ones. To do so, operator's high-level policies are translated into IDIPS cost functions. These cost functions determine the cost of each path based on routing information and path performance (like delay or monetary cost). The lower the cost, the better the path. To hide the topology and computation details, IDIPS translates the costs computed from the cost functions into ranks. In Chapter 6 we show that by clustering the destinations that are topologically close and by only measuring one destination in each group, it is possible to reduce the number of measurements without loosing too much accuracy. We propose a new clustering technique that provides a good trade-off between the number of measurements and the measurement accuracy. In addition, we show that if IDIPS performs the measurements on behalf of the clients the number of measurements can be reduced significantly. Finally, in Chapter. 7, we show how to combine LISP, LISP-Tree and IDIPS to obtain performance based inter-domain incoming traffic engineering.

Implementation and standardization efforts

All along this thesis, we made efforts to propose systems that are simple in their concept and their implementation. In addition, we ensured that all the proposed solutions can be implemented and operated in today's Internet.

To understand the network operators needs, we participated actively in the Internet Engineering Task Force (IETF). We were involved in the creation of the LISP working group[1] as an experimental working group at the IETF since the early discussion at the Routing Research Group (RRG) of the IRTF in 2008. To understand LISP and its limits, we have participated to the implementation effort. We were first involved in the OpenLISP project[2] that proposes an open source implementation of LISP. OpenLISP implements the LISP data-plane in the FreeBSD kernel [ISB11, SDIB08, SIB09a, SIB09b]. OpenLISP is used by researchers to validate new LISP related ideas.

We have also implemented a basic LISP data plane in the Click Modular Router [SN09]. This implementation is interoperable with the OpenLISP implementation and the Cisco NX-OS ones. The implementation is briefly described in Appendix A. It has been deployed in the BOWL infrastructure at the Technical University Berlin.

In addition we have developed a LISP mapping server and a LISP mapping resolver. This implementation and the notes related to its design are available on `https://scm.info.ucl.ac.be/trac/lisp-click/`. We are frequently using this implementation in our testbeds and this implementation is being adapted to become the control plane of the OpenLISP project.

Some of our work at the IETF has been adopted as working group documents and are on the way to become RFCs **[DSA:[?]]**. In addition, LISP-Tree is finally considered by Cisco as a valuable alternative to the LISP+ALT mapping system. We are now in the process of implementing LISP-Tree and standardizing it in order to offer an open source LISP control-plane for the community.

Our work at the IETF is not limited to LISP. Indeed, we have participated at the creation and the early days of the ALTO Working Group (Application Layer Traffic Optimization)[3]. Our main contribution in this working group is the introduction in the protocol recommendation to be able to consider any kind of address in the protocol as well as the support of source addresses [AFP$^+$09]. Indeed, at its early stage ALTO was only considering destination addresses. We argued in favor of accepting IP prefixes or any other form of address aggregation.

To understand the potential technical limitations of IDIPS, we carefully worked on its architecture and implementation. We have shown that IDIPS is scalable. Our implementation is freely available and is part of the final deliverable material for the European Project ECODE. Moreover, IDIPS can be used to implement the ALTO Endpoint Cost Service specified in the ALTO specifications [APY11, SDIB08, SB11].

[1]`http://tools.ietf.org/wg/lisp/`
[2]`http://www.openlisp.org`
[3]`http://tools.ietf.org/wg/alto/`

Bibliographic Notes

Most of the work presented in this thesis has already appeared in conference proceedings, journals and IETF contributions. The list of related publications is shown hereafter:

Journals

- LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System with L. Jakab, A. Cabellos-Aparicio, F. Coras and O. Bonaventure, IEEE Journal on Selected Areas in Communications, 28(8):1332 – 1343, October 2010
- Implementing the Locator/ID Separation Protocol: Design and Experience, with L. Iannone and O. Bonaventure Computer Networks, 55(4):948 – 958, March 2011

Conferences and Workshops

- LISP-Click: A Click implementation of the Locator/ID Separation Protocol, with V. Nguyen 1st Symposium on Click Modular Router, November 2009
- How to mitigate the effect of scans on mapping systems, with L. Iannone Trilogy Future Internet Summer School Poster Session, August 2009
- On the Impact of Clustering on Measurement Reduction, with B. Donnet and O. Bonaventure Proc. IFIP Networking 2009, May 2009
- Interdomain Traffic Engineering in a Locator/Identifier Separation Context, with B. Donnet, L. Iannone and O. Bonaventure, Proc. Internet Network Management Workshop 2008 (INM), October 2008
- Implementation and Preliminary Evaluation of an ISP-Driven Informed Path Selection, with B. Donnet and O. Bonaventure Proc. ACM CoNEXT Student workshop, December 2007

All along the thesis period, we have also actively contributed to the standardization efforts within the LISP working group at the IETF. Some of our contributions are now part of the LISP protocol and are on the way to become RFCs.

IETF working group documents and individual drafts

- LISP Map-Versioning, with L. Iannone and O. Bonaventure IETF draft, work in progress, draft-ietf-lisp-map-versioning-02, July 2011

- LISP-Security (LISP-SEC), with F. Maino, V. Ermagan, A. Cabellos and O. Bonaventure IETF draft, work in progress, draft-ietf-lisp-sec-00, July 2011
- LISP Threats Analysis, with L. Iannone and O. Bonaventure IETF draft, work in progress, draft-ietf-lisp-threats-00, July 2011
- Preserving the reachability of LISP ETRs in case of failures, with O. Bonaventure and P. FrancÌ§ois IETF draft, work in progress, draft-bonaventure-lisp-preserve-00, July 2009
- The PROXIDOR Service, with O. Akonjang, A. Feldmann, S. Previdi and B. Davie IETF draft, work in progress, draft-akonjang-alto-proxidor-00, March 2009
- OpenLISP Implementation Report, with L. Iannone and O. Bonaventure IETF draft, work in progress, draft-iannone-openlisp-implementation-01, July 2008
- Why should the Traffic Optimization not be restricted to the Application-Layer?, with D. Papadimitriou IETF draft, work in progress, draft-saucez-alto-generalized-alto-00, July 2008
- IDIPS: ISP-Driven Informed Path Selection, with B. Donnet and O. Bonaventure IETF draft, work in progress, draft-saucez-idips-00, February 2008
- The case for an informed path selection service, with O. Bonaventure and B. Donnet IETF draft, work in progress, draft-bonaventure-informed-path-selection-00, February 2008

Acronyms

ADS	Anomaly Detection System
ALTO	Application-Layer Traffic Optimization
API	Application Programming Interface
AS	Autonomous System
ALT	Alternative Topology
BGP	Border Gateway Protocol
CDN	Content Delivery Network
CESCA	Catalan Research Network
CF	Cost Function
CGA	Cryptographically Generated Addresses
CIDR	Classless Inter-domain Routing
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DFZ	Default-Free Zone
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
DoS	Denial of Service
ECMP	Equal Cost Multi-Path
EID	Endpoint IDentifier
ETR	Egress Tunnel Router
FIB	Forwarding Information Base
FQDN	Fully Qualified Domain Names
FIRMS	Future InteRnet Mapping System
FTP	File Transfer Protocol
GRE	Generic Routing Encapsulation
HBA	Hash-Based Addresses
HIP	Host Identity Protocol
HIT	Host Identity Tag
HTTP	HyperText Transfer Protocol
IAB	Internet Architecture Board
ICMP	Internet Control Message Protocol
IDIPS	ISP-Driven Informed Path Selection
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
ILNP	Identifier-Locator Network Protocol

IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRTF	Internet Research Task Force
IS-IS	Intermediate System to Intermediate System
ISP	Internet Service Providers
ITR	Ingress Tunnel Router
L3	Layer 3
L4	Layer 4
LAN	Local Area Network
LCAF	LISP Canonical Address Format
LER	Local EID Registries
LIR	Local Internet Registries
LISP	Locator/Identifier Separation Protocol
LSB	Locator Status Bits
LTS	LISP-Tree Server
MAC	Message Authentication Code
MED	Multi-Exit Discriminator
MPTCP	Multi-Path TCP
MPLS	Multi-Protocol Label Switching
MR	Map-Resolver
MS	Map-Server
NAROS	Name, Address and ROute System
NAT	Network Address Translation
NERD	Not-so-novel EID to RLOC Database
OPEX	OPerational EXpenses
OSPF	Open Shortest Path First
P2P	Peer-to-Peer
PA	Provider Aggregatable
PI	Provider Independent
PETR	Proxy ETR
PITR	Proxy ITR
PoP	Point of Presence
QoS	Quality of Service
RER	Regional EID Registries
RFC	Request For Comment
RIB	Routing Information Base
RIR	Regional Internet Registries
RLOC	Routing LOCator
RPSL	Routing Policy Specification Language
RRG	Routing Research Group
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SMR	Solicit Map-Request
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TE	Traffic Engineering

TLD	Top-Level Domain
TQ	Traffic Qualification
TTL	Time To Live
UDP	User Datagram Protocol
VLAN	Virtual LAN
VPN	Virtual Private Network
XORP	eXtensible Open Router Platform
XRL	XORP Resource Locator

Contents

Preamble **i**

Table of Content **xi**

List of Figures **xv**

List of Tables **xix**

1 Background **1**
1.1 Intradomain Routing 5
1.2 Interdomain Routing Protocols 7

2 Locator/Identifier Separation **11**
2.1 Introduction 11
2.2 Locator/Identifier Separation at the host level 14
2.3 The Locator/Identifier Separation Protocol (LISP) 17
2.3.1 LISP Data Plane 18
2.3.2 LISP Control Plane 19
2.3.3 On the EID-to-RLOC Cache size 24
2.4 LISP as an evolutionary approach 27
2.4.1 Core Internet Traversal 27
2.4.2 Incremental Deployment 28
2.5 Reachability Problem 28
2.6 Virtualization with LISP 31
2.7 Deployment lessons 32
2.8 Conclusion 33

3 Interdomain traffic engineering with LISP **35**
3.1 Introduction 35
3.2 Traffic engineering with LISP 35
3.2.1 Outgoing traffic engineering 36
3.2.2 Incoming traffic engineering 37
3.2.3 Mapping differentiation 39
3.3 Conclusion 41

4 LISP-Tree **43**
4.1 Introduction . . . 43
4.2 LISP-TREE . . . 44
4.2.1 Why a DNS like mapping system? . . . 44
4.2.2 LISP-TREE Overview . . . 46
4.2.3 LISP-TREE Model . . . 46
4.2.4 LISP-TREE Modes . . . 48
4.2.5 LISP-TREE Deployment Scenario . . . 50
4.3 Simulation Model . . . 51
4.3.1 Topology Model . . . 51
4.3.2 ITR Model . . . 52
4.3.3 Mapping System Models . . . 52
4.4 Mapping System Comparison . . . 54
4.4.1 Experimental Datasets . . . 54
4.4.2 Cache Miss Rate . . . 55
4.4.3 Mapping Lookup Latency . . . 56
4.4.4 Node Load . . . 58
4.4.5 Operational considerations . . . 62
4.5 Implementing LISP-Tree with LISP Iterable Mappings . . . 63
4.5.1 Map-Request Extension for LISP-Tree . . . 64
4.5.2 Functional Specifications . . . 66
4.6 Related Work . . . 69
4.7 Conclusion . . . 70

5 IDIPS **73**
5.1 Introduction . . . 73
5.2 An Informed Path Selection Service . . . 75
5.2.1 Overview . . . 75
5.2.2 Use Cases . . . 77
5.3 Service Construction . . . 79
5.3.1 Architecture . . . 79
5.3.2 XORP Implementation . . . 87
5.3.3 High Level Cost Functions Implementation . . . 99
5.4 Evaluation . . . 103
5.4.1 Methodology . . . 103
5.4.2 Results . . . 104
5.5 Related Work . . . 107
5.6 Conclusion . . . 110

6 Measurement Reduction with Clustering **113**
6.1 Introduction . . . 113
6.2 Theoretical Background . . . 114
6.2.1 Measurement reduction . . . 114
6.2.2 Clustering Accuracy . . . 116

6.3 Clustering Techniques 117
6.4 Evaluation 119
6.4.1 Methodology 119
6.4.2 Measurement Accuracy 119
6.4.3 Measurement Reduction 121
6.5 Conclusion 126

7 Incoming traffic engineering with LISP and IDIPS 127
7.1 Introduction 127
7.2 LISP and IDIPS combination 129
7.2.1 Proof of Concept 131
7.3 Conclusion 133

8 Conclusion 135

Conclusion 135
8.1 Further Work 137

A LISP-Click: A Click implementation of LISP 139
A.1 Introduction 139
A.2 LISP-Click architecture 139
A.2.1 LISPCache Element 140
A.2.2 LISPDecap Element 142
A.2.3 LISPEncap Element 143
A.2.4 LISPMissManager Element 144

References 147

List of Figures

1.1 Example of Internet architecture. Arrowed lines with a $ sign: customer-provider links, doubled lines: shared-cost links, single lines: intradomain links, blue clouds: transit ASes, green clouds: stub ASes, gray clouds: CDNs 3
1.2 IGP Example 6

2.1 History of the number of active BGP entries (FIB size) (source potaroo.net) 12
2.2 Hourly average of per-second updated and withdrawn prefix rate at the hinge between 2010 and 2011 (source potaroo.net) 12
2.3 LISP general behavior 18
2.4 LISP Data-Plane header format 19
2.5 EID-to-RLOC mapping Example 19
2.6 EID-to-RLOC association example 25
2.7 RLOC distribution 26
2.8 LISP reachability problem example 29
2.9 EID vs RLOC unreachability rate 29
2.10 L3 VPN implementation with LISP instance ID. Both red and purple VPNs use the same EID prefix, the differentiation is done at the ETR based on the instance ID. 32

3.1 Example of source-based routing with LISP. For 192.0.2.0/25, the source locator is 1.0.0.1 and for 192.0.2.128/25, the source locator is 2.0.0.2. Both networks have the 3.0.0.3 RLOC for backup. 37

4.1 Global overview of LISP-TREE. 1. The requester asks the discovery part to have the locator of some authoritative ETRs for an EID *e*. 2. The discovery part provides this list. 3. The Map-Resolver sends a Map-Request to one of these ETRs. 4. The ETR sends a Map-Reply back with a mapping for *e*. . . 47
4.2 LISP-TREE works with different mode: (*a*) recursive and (*b*) iterative. 49
4.3 LISP+ALT architecture 53

4.4 CDF of map-stretch ratio. Use of caching makes LISP-TREE have a constant value for the majority of the lookups. 57
4.5 CDF of hop count. For hierarchical mapping systems it is almost constant, for LISP-DHT we have a wide array of different values. 59
4.6 LISP-DHT load distribution. Fingers of the ITR, represented with the vertical lines, cluster on the right, having a considerable load. 61
4.7 The cumulative distribution of the amount of state in the level 2 LISP-TREE and LISP+ALT nodes 62
4.8 Map-Request extension to support the Iterable Mapping extension 65
4.9 Iterable reply message format 66
4.10 Example of Iterable mapping in iterative and recursive mode. `N`, `D`, `E` and `F` run in iterative mode while `B` and `C` are in recursive mode. 68

5.1 A network service rank paths for the clients 75
5.2 Idips server API for synchronous mode clients 82
5.3 Idips server API for asynchronous mode clients 83
5.4 One-by-one path ranking retrieval algorithm 84
5.5 Measurement module API 85
5.6 Prediction module API 86
5.7 Idips within Xorp 88
5.8 Example of modules interactions in Idips 89
5.9 Xorp Querying module XRL interface 92
5.10 Xorp Measurement module XRL interface 93
5.11 Xorp Prediction module XRL interface 94
5.12 Measurement module `loop` method pseudo-code 97
5.13 UDP ICMP port unreachable management 97
5.14 Delay precision comparison between standard UDP ping, ICMP ping and XORP UDP ping 98
5.15 Evaluation testbed 103
5.16 Idips service time as perceived by the client 105
5.17 Proportion of the service time split - median over the ten runs 106
5.18 Load on Idips without XRL 107

6.1 Illustration of measurements duplication and reduction 114
6.2 Estimation error applied to RTT 120
6.3 Estimation error applied to hop 122
6.4 25^{th}, 50^{th} and 75^{th} percentile of estimation error 123
6.5 Popularity in the NetFlow dataset 124
6.6 Cluster size distribution 124
6.7 Measurement reduction in the NetFlow dataset 125

7.1 Relative distance between the locators and the locator with the shortest delay in the mappings 128
7.2 Case study testbed 132
7.3 Evolution of the different flows bandwidth for the different network events. 133

A.1 LISP-Click Architecture 140

List of Tables

4.1 Node load in levels 1 and 2 60

Chapter 1

Background

The Internet is a world-wide data network composed of tens of thousands interconnected networks, themselves constituted of a huge number of smaller networks connecting a plethora of end systems including personal computers, servers, smart-phones or even sensors. The end systems exchange data together by sending packets over the network. All the packets are built according to the specifications of the *Internet Protocol (IP)* protocol [Pos81, DH98]. The networks are connected together by *routers*. Routers are intermediate devices that can relay packets between networks. In general, a packet is generated by an end system and is transmitted to a router that transmits it to another router and so on until the packet reaches the final destination end system. IP specifies that every device on the Internet must have an IP address. This address is unique to the device. A packet is identified by its source address, i.e., the IP address of the source end system, and its destination address, i.e., the IP address of the destination end system. When a packet arrives at a router, the router determines the network to which belongs the destination address. If the destination end system is directly connected to the router, the packet is directly transmitted to the end system. Otherwise, the packet is transmitted to another router and so on until the packet reaches the destination end system.

The networks composing the Internet are operated by different institutions. A group of routers operated by a single administration is called an *Autonomous System (AS)*. The routing at the level of an Autonomous System is called *intradomain* routing. Routing between Autonomous Systems is called *interdomain* routing.

The relationship between two interconnected Autonomous Systems can be of two types: (*i*) customer-provider or (*ii*) shared-cost [DD08, Gao01, DKF^{+}07, DFD10, ZCB96]. This relationship classification usually depends on the flow of money between the two ASes. With a customer-provider relationship, the customer AS pays money to the provider to exchange packets with the provider. In a shared-cost relationship, the traffic is not charged by any of the two ASes in the relationship. However, the cost of operating the equipments that interconnect the two ASes is identically supported by the two ASes. Two ASes that are directly connected are said to be *neighbors*.

The Internet is a graph that can be split into the core and the edge. The Autonomous Systems are hence classified as *transit* or *stub* ASes [ZCB96, CDN+97]. Transit ASes are forming the core and are used to transmit packets from one neighbor AS to another. The stub ASes are at the edge and contain most of the end systems. Example of transit ASes are *Internet Service Providers (ISP)* like Sprint, Level 3, France Telecom, Deutsche Telekom or Belnet. Companies or universities are stub ASes in general. In general, transit are connected with several other transit ASes with customer-provider or shared-cost relationships and have many customer stub ASes. On the contrary, stub ASes only have few neighboring ASes, most of them being their providers and some other maintaining a shared-cost relationship. In addition to the transit and stubs, some ASes are called *Content Delivery Networks (CDN)*. CDNs are the source of a large part of the traffic and often maintain an important number of shared-cost peering relationships with their neighbors [LIJM+10]. The best examples of CDNs are Google, Facebook, Akamai and Limelight. Some CDNs are so huge that they virtually have only shared-cost relationship links with their neighbor ASes.

Fig. 1.1 illustrates a synthetic Internet where each cloud represents an Autonomous System. The ASes connected by an arrowed line and a $ sign have a customer-provider relationship where the provider is pointed by the arrow. Doubled lines between ASes mean that the two ASes have a shared-cost relationship. The green clouds are for stub ASes, the blue clouds are for the transit ASes. Finally, the gray cloud is for a CDN. For example, the hypothetical exchange of packets from the UCL AS to the Google AS in Fig. 1.1 will follow the path UCL - Belnet - Level 3 - Google. UCL must pay Belnet for such traffic. Belnet will pay Level 3 for the traffic. Finally, neither Level 3 nor Google pays for the traffic because they have a shared-cost relationship [GR00].

Every device on the Internet is allocated one unique IP address per interface that connects it to another device on the Internet. IP addresses are grouped in prefixes. The addresses sharing the same high-order bits are said to belong to the same prefix. Packets are routed in the Internet based on the prefix of their destination address. An address is then logically divided in two parts. The *network* and the *host*. The prefix of the address determines the network it belongs to and the following of the address specifies the device the address belongs to inside the network.

When the Internet has been created, the prefixes were divided into three main classes. The A-class for large networks where the prefix is encoded with 8 bits. The B-class for medium size networks, where the prefix is encoded with 16 bits. And finally, the C-class for small size networks with a prefix encoded with 24 bits. The IP addresses, at that time were encoded

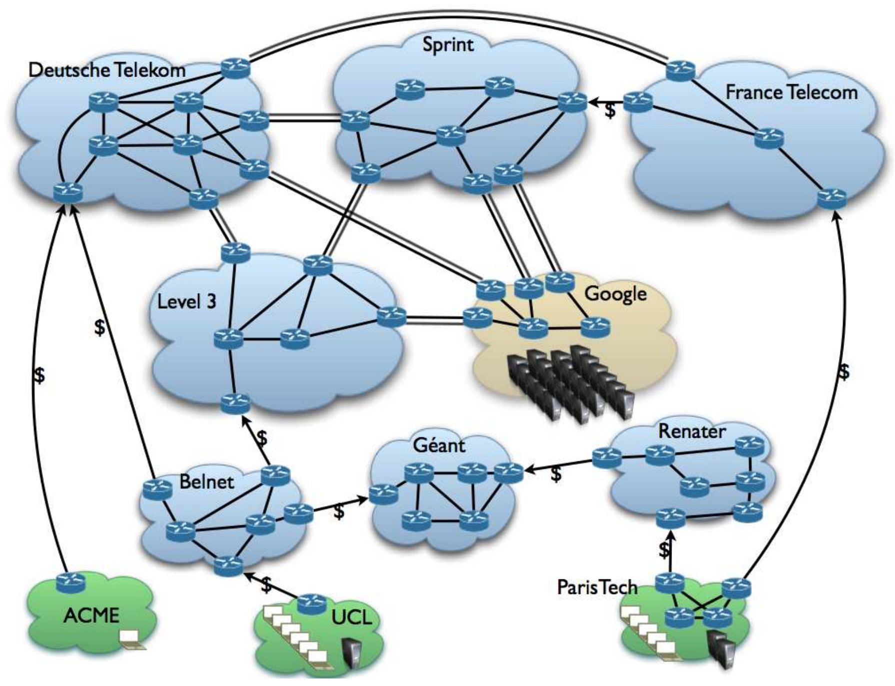


Figure 1.1: Example of Internet architecture. Arrowed lines with a $ sign: customer-provider links, doubled lines: shared-cost links, single lines: intradomain links, blue clouds: transit ASes, green clouds: stub ASes, gray clouds: CDNs

with 32 bits, meaning that most of the networks connected to the Internet had to be B-classes. However, this definition of prefixes led to an important waste of addresses. For example, our university has about 20,000 students and employees and then obtained a B-class prefix providing 65,534 addresses or 3 addresses per member of our institution. Classful routing is simple and was adequate in the early days of Internet when routers had very limited resources. However, the waste of IP addresses rapidly became a problem with the increase of the number of networks connected to the Internet. *Classless Inter-domain Routing (CIDR)* [FLYV93, FL06] has then been proposed to overcome the limitations of classful routing. With CIDR, the size of the prefix is not defined. Therefore, it is possible to assign the right size for a prefix according to the number of hosts in the network. For example, our campus could use a prefix of size 17 instead of size 16. The CIDR notation is `p/n` where `p` is the IP address where the host bits are set to 0 and `n` is the size of the prefix. For example, the `192.0.2.0/24` prefix means that the network can contain up to 256 addresses. The first possible address being `192.0.2.0` and the last possible one is `192.0.2.255`. Using classless routing increases the forwarding complexity at the router. Indeed, the router has no information about the size of the prefix of the destination address of a packet. For example, if the destination address is `192.0.2.129`, the address can belong to `0.0.0.0/0`, `192.0.2.0/24` or `192.0.2.128/25` or any other prefix that covers this address. To determine the outgoing interface to forward a packet to, a CIDR router must then check all the possible prefixes. If the router has the two prefixes `192.0.2.0/24` and `192.0.2.128/25` in its forwarding table, it will conclude that `192.0.2.129` belongs to the `192.0.2.128/25` prefix because this prefix has the longest prefix matching. CIDR routing relies on longest prefix matching to determine the outgoing interface. Longest prefix matching can be achieved with a trie like a radix tree.

With the growing importance of Internet and data networks, *Traffic Engineering (TE)* became a major preoccupation for network operators [Mor02, ACE$^+$02, Uhl04, dL05, Quo06, Pel06]. Traffic engineering is used to enhance the performance of a network. On the one hand, TE aims at improving the traffic performances. On the other hand, traffic engineering is used to optimize the resource usage [ACE$^+$02]. However, the most important TE objective is probably the reliability. Based on [ACE$^+$02] we can say that TE relies on three fundamentals components:

network robustness: a network must be robust to failure to minimize the risk of connectivity loss,

network efficiency: the resources inside the network must be use efficiently to eventually reduce the operational expenses (OPEX),

traffic performance: the network must be such that the traffic it transports fulfills all the performance requirements.

The network robustness consists in maximizing the number of distinct paths that can be used inside the network to avoid congestion (and thus packet loss) and to recover from a failure inside the network as quickly as possible. To increase their robustness, networks tend to be *multihomed*. A network is multihomed if it is connected to the Internet via several ISPs. Multihoming protects the network against the loss of one of its ISPs, either because of a technical issue or because of the bankruptcy of the provider. In the context of robustness, TE is the art of minimizing the risk for a packet to be lost and to reduce the impact of a failure.

The network efficiency is related to the network resource usage. Resources are limited and using them has a cost. Optimizing the use of each resource to minimize the operational expenses is a must in any company or ISP. Moreover, the ISPs often charge their clients based on their link usage. To minimize the cost of using Internet in a company, it is then important for a network to be able to control the ISP that it uses to send packets to the Internet but also the ISPs that it uses to receive packets from the Internet.

Finally, traffic performance is probably the ultimate goal of a network operator. However, optimizing the traffic performance does not especially mean that the traffic must follow the best possible path (e.g., the one that minimizes the delay). It rather means that every flow must respect the conditions that makes it working properly. The Internet has been designed with the *best-effort* perspective. Best-effort only guarantees that the traffic will be forwarded as best as it can, without any other guarantee. However, Internet is now used for phone communications or to assist medical operations. Such traffic cannot be satisfied by a simple best-effort approach. For example, a phone communication does not need a high bandwidth but needs low delay between the two phone terminals and stable traffic conditions. If a network is designed to carry such traffic, this traffic will be prioritized over traffic with less requirements.

1.1 Intradomain Routing

The role of intradomain routing is to compute the routes towards the prefixes inside a domain. Theses routes are computed with an *Interior Gateway Protocol (IGP)* like OSPF [Moy98, CFM99] or IS-IS [Ora90]. Most deployed IGPs rely on the link-state paradigm. In link-state routing, every router floods the state of its links. Therefore, every router can build a full map of the network in the domain. A weight is associated to each direction of the links. The network then constitutes a weighted directed graph. The objective of the routing protocol is to find the shortest paths between the routers. The cost of a path between two routers being the sum of the cost of the links composing the path between the routers. Each router computes the shortest paths with all the routers in the domain by applying the *Dijkstra* algorithm on the abstracted graph that represents the network [Dij59]. The Dijkstra algorithm computes a minimum spanning tree for the graph. The spanning tree is used to determine the link to which a packet must be forwarded to reach a destination. Because every router has a full view of the network and applies the same shortest path computation algorithm, the path followed by a packet to reach a destination is the shortest one.

Fig. 1.2 shows an example of intradomain network. The links between the routers are presented by plain lines and the minimum spanning tree is defined by the dashed-lines. In Fig. 1.2, when a packet arrives at the router B, it will be sent to the East if the destination is inside the prefix 4.0.0.0/16 and to the North if the packet destination is in the 3.0.0.0/16 prefix. If the packet is for 5.0.0.0/16, it is forwarded to the South link.

Intradomain traffic engineering is an optimization problem where the weight of each link must be computed to optimize the way the traffic is forwarded in the network [FRT02, FT00, FT02]. The input of the opti-

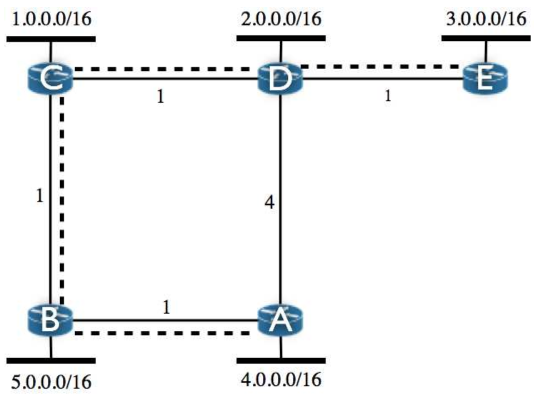


Figure 1.2: IGP Example

mization system is the domain *traffic matrix* and the output is the weights of the links such that the links are used accordingly to the domain policies. In general, link weight allocation is an optimization problem. First, every link is assigned a unit weight, then the weights are assigned locally or globally [FRT02]. For local weight allocation, the system only changes the weight of the congested links in order to reduce their load. On the contrary, when the optimization problem is seen globally, the weights are assigned such that a link never have to carry more unit of traffic than it can. The traffic demand is estimated with the traffic matrices. For example, if every link can only support one unit of traffic, the weights are computed such that a link never carries more than one unit of traffic. Linear programming can be used to solve this problem [FT00]. The traffic matrix of a domain is a matrix reflecting the volume of traffic that is exchanged between all the possible <source,destination> pairs in the network [MTS+02]. Several techniques exist to construct the best possible traffic matrix estimates [MTS+02, FGL+00, UQLB06]. These techniques follow either an optimization approach or a statistical approach. The optimization approach takes the information from the topology and traffic counters in the network and apply constraints on the obtained system to compute the traffic matrix. On the other hand, statistical inferences takes information from the topology and traffic counters and determine the distributions that fit with the observations [MTS+02].

Pure IP based TE is limited by the fact that packets are forwarded by the routers based on the destination address. The *Multi-Protocol Label Switching (MPLS)* protocol can be used to circumvent this limitation [RVC01]. With MPLS, the forwarding of a packet is based on its attached label. The label is attached to the packet when it enters the MPLS part of the network. Traffic engineering with MPLS is more flexible than with pure IP as any label can be attached to the packets. MPLS traffic engineering is an optimization problem where the labels must be distributed between the routers with a PCE or with LDP such that the link usage is optimized [EJLW01, AmW99, AJ02, AMA^{+}99].

1.2 Interdomain Routing Protocols

The principle of interdomain routing protocols is simple, at the border of each autonomous system, one or several routers are placed and connected to one or several border routers of the neighboring AS. A router at the border of an AS announces to its neighbors the prefixes of its internal network (i.e., IGP) and the routes announced by its neighboring ASes.

The *Border Gateway Protocol (BGP)* [RLH06, GW99] is the de facto interdomain routing protocol in the Internet. BGP is a *path-vector* routing protocol where a BGP router announces to its BGP neighbor routers the prefixes it can reach. A prefix is always announced with its AS path. The AS path is the list of ASes that the prefix crosses to reach the prefix originator AS. The AS path is provided to avoid forwarding loops.

However, BGP is more than just a path-vector routing protocol. Indeed, BGP is designed to allow ASes to enforce their *routing policies*. That is, if a router receives an announcement for a prefix from several neighboring ASes, it only transmits to its neighboring ASes the best announcement it receives. The choice of the best announcement is preformed by a decision process that allows policies to be taken into account for the selection of the best path. For instance, in the example of Fig. 1.1 the Belnet AS will receive a route for a network in ParisTech from its three providers[1]. However, Belnet will only announce the route via Géant to the UCL AS because Géant is the provider that will charge Belnet the less for the traffic. Again from the example of Fig. 1.1, Level 3 will not announce the routes for networks in Google to Deutsche Telekom because it has a shared cost peering relationship with both ASes and thus would not earn money by announcing these routes to Deutsche Telekom. On the contrary, the routes to Google will be announced to Belnet because doing so will attract traffic from its Belnet customer that will pay for sending traffic to Google while Level 3 does not have to pay anything for this traffic: Level 3 then maximizes its profit.

As opposed to link-state routing protocols where all the routers know the whole topology, the path-vector nature of BGP limits the selection of interdomain routes. Indeed, for any prefix P/p, a BGP router only redistributes to its neighbors the single best route for P/p. Even though the router knows several routes for P/p. This behavior reduces the path diversity but increases the scalability of the protocol. Although, BGP supports traffic engineering, but the traffic management is local per AS and no solution exists to fulfill a global TE objective. The policy based nature of

[1]The names and relationships provided in the example are pure fiction.

BGP helps an AS to control its incoming and outgoing interdomain traffic. The BGP policies applied to the BGP path selection process constraint the selection of the best route for a prefix. The first solution for preferring a route over another is to use the *Local-Pref* attribute. The Local-Pref gives an ordering to the routes for a prefix. The best route for a P/p is always a route that is among the routes with the highest Local-Pref value for the prefix. For example, to prefer the routes learned from a customer over a route learned from a peer over a route learned from a provider, a Local-Pref of 1000 can be assigned to all the routes learned for a customer, a value of 100 for the routes learned from a peer and a value of 10 for the routes learned from a provider. The best route will then be the route with the highest Local-Pref and the shortest AS path.

The Local-Pref is only used locally inside an AS to control the selection of its best routes. To control its incoming traffic, an AS can use different techniques. *AS path prepending* uses the path-vector nature of BGP to control the incoming traffic [GDZ05, QUP$^+$03]. The technique consists of artificially increasing the AS path of the route that should not be used. For example, the Belnet AS from Fig. 1.1 can use AS Path prepending when it announces a prefix for the UCL AS. Assuming that Belnet prefers to receive traffic from Géant, then from Level 3 and only if no other solution exists from Deutsche Telekom, Belnet can announce the prefix for UCL normally to Géant. When the prefix is announced to Level 3, the AS path is increased by 4 hops, simply by repeating the AS number of Belnet 4 times in a row in the exported route. Finally, when the route is announced to Deutsche Telekom, the AS path can be increased by 6 hops by repeating 6 times in a row the Belnet AS number. By doing so, if a distant AS receives the three routes and if these three routes have the same Local-Pref then the route via Géant will be used.

If an AS is connected with several routers to its neighboring AS, it can use the *Multi-Exit Discriminator (MED)* to attract the traffic to a particular router [GSW99]. If two routes for a prefix P/p are received from the same AS and if the Local-Pref and the AS path length are identical, the route with the smallest MED attribute value will be selected. The MED allows an AS to signal the peering link it prefers to be used. The MED attribute is non transitive, meaning that the value is not propagated by the AS that received the route. In is worth noticing that the MED is in general accepted only under financial compensations or at least prior specific agreements between the two neighboring ASes.

The announcement of the single best route per prefix and the path-vector nature of BGP limit the traffic control for the ASes. For example, there exist situations where the AS Path prepending has no effect [Pel06]. For example, in Fig. 1.1, the traffic from an end system in the ACME AS to a UCL end system will always enter the Belnet AS via Deutsche Telekom because ACME and Belnet are both customers of Deutsche Telekom. Similarly, the traffic from Sprint to UCL will not depend on the AS path but on the preference of Sprint for Level 3 or Deutsche Telekom. To overcome this limitation, an AS can do *prefix deaggregation*. The principle of prefix deaggregation is to announce more specific prefixes to the preferred neighbors and less specific prefixes to the least preferred neighbors. Because routing in the Internet is made on a longest prefix matching basis, a more specific prefix is always preferred to a less specific one. If we take the same example as previously with the announcement of the UCL routes by Belnet in Fig. 1.1 and if we assume that the prefix that UCL announces to Belnet is `130.104.0.0/16`, the prefix deaggregation can be done as follows to fulfill the incoming traffic engineering requirements. Instead of announcing the `130.104.0.0/16` prefix to Géant, Belnet announces the `130.104.0.0/18`, `130.104.64.0/18`, `130.104.128.0/18` and `130.104.192.0/18` prefixes. Similarly, the `130.104.0.0/17` and `130.104.128.0/17` prefixes are announced to Level 3 instead of `130.104.0.0/16`. Finally, Belnet only announces the `130.104.0.0/16` prefix to Deutsche Telekom. Therefore, any traffic generated by Deutsche Telekom to UCL will go through Level 3 because more specific prefixes are announced to Level 3 than to Deutsche Telekom by Belnet and because Deutsche Telekom receives routes from Level 3. Prefix deaggregation overcomes some BGP limitations but is limited by the maximum deaggregation authorized in BGP (i.e., /24). In addition, prefix deaggregation increases the amount of prefixes announced to the Internet and potentially increases the BGP churn [BNC02, QIdLB07].

Finally, *communities* can be used to add information to the routes announced by a BGP router [DB08, QUP$^+$03]. BGP communities are optional attributes that can be added to any route. The communities are used by the BGP routers to filter the routes and to apply special policies. For example, the `NO_EXPORT` community indicates that the BGP router that receives a route with this community can use it for its local forwarding but cannot redistribute the route to its BGP neighbors. Some communities are transitive and others are not. Transitive communities are transmitted from neighbors to neighbors while non transitive communities are only used by the first BGP router that receives the route.

Notwithstanding that BGP offers AS path prepending, MED or the communities, it does not allow a precise control on the incoming traffic: the more distant the traffic destination AS is from the traffic source AS, the least it controls the path followed by the incoming traffic. In this thesis, we present new mechanisms that enable any network to have control over its incoming traffic and to optimize it for performance.

Chapter 2

Locator/Identifier Separation

2.1 Introduction

During the last twenty years, the Internet grew at a rapid pace [MXZ+05]. Thirty years ago, the ARPANet interconnected only about 200 hosts. As of this writing, there are more than 700,000,000 hosts connected to the Internet and we will probably quickly reach more than one billion hosts. These hosts are grouped in networks that are themselves grouped in Autonomous Systems (AS). An AS is usually an Internet Service Provider or a campus/enterprise network. There are today about 35,000 different ASes. Sustaining the Internet growth causes some scalability problem on routers. In addition to having to sustain higher link bandwidth, router must also be able to support a growing number of active destination prefixes. In 2001, there were about 100,000 prefixes in the routing tables of the core Internet routers [MXZ+05]. In 2006, this number had already doubled and the Internet Architecture Board (IAB) sponsored a workshop to discuss the scalability of the Internet routing and addressing systems. Fig. 2.1 obtained from the BGP monitoring site potaroo.net shows the evolution of the number of active BGP entries since the late eighties. The active entries are installed in the *Forwarding Information Base, FIB* which needs fast memory to support forwarding at line rate. This memory is expensive and consumes a lot of energy. It is worth to notice that the number of BGP prefixes in the *Routing Information Base (RIB)* can exceed one million entries, but these prefixes are only used to compute the routes and can be stored in a cheap and slow memory. However, Fig. 2.2 also coming from the potaroo.net site illustrates a more sever problem, the *BGP churn*. When a prefix becomes unreachable or has attributes that are changing, the router announces to its neighbors that the route has been withdrawn or changed and this information is propagated to all the Internet. Fig. 2.2 shows the hourly averaged per-second rate of update and withdraw received on a typical BGP router. Fig. 2.2 shows that the router are facing peaks of more than 1,000 updates/withdraws per

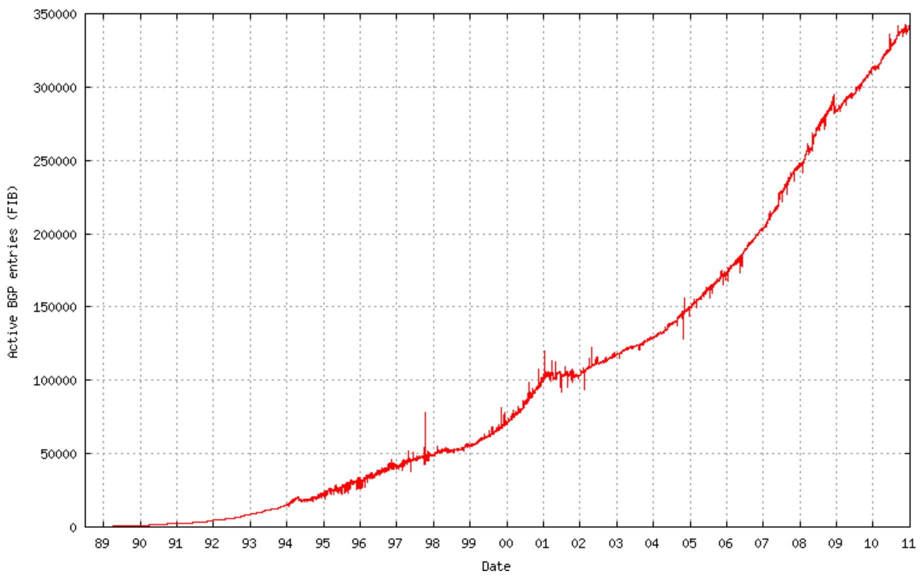


Figure 2.1: History of the number of active BGP entries (FIB size) (source potaroo.net)

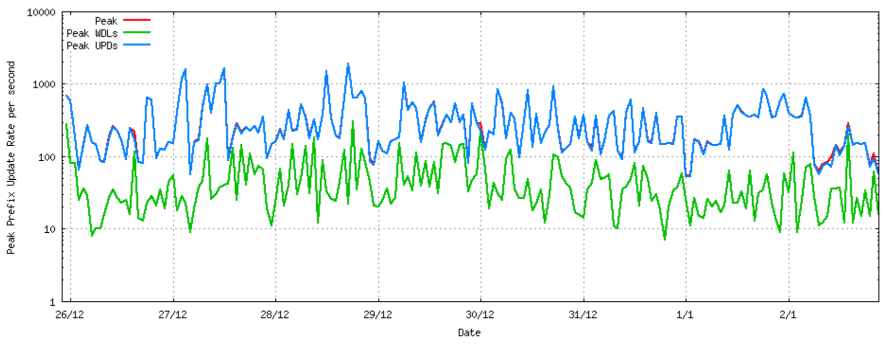


Figure 2.2: Hourly average of per-second updated and withdrawn prefix rate at the hinge between 2010 and 2011 (source potaroo.net)

second which impact the CPU utilization on the core routers.

To deal with this scalability problem, the Internet Research Task Force (IRTF) chartered a Routing Research Group (RRG) to provide recommendations on the evolution of the Internet routing and addressing systems [Li10]. The RRG discussed a large number of proposals. The Locator/Identifier Separation Protocol (LISP) was a proposal that attracted a lot of interest. A working group was created within the Internet Engineering Task Force to develop the experimental specifications for the LISP protocol. LISP, contrary to the other solutions like ILNP [ABH10], Shim6 [NB09] or

HIP [MN06], does not require the hosts to be modified.

LISP solves the scalability problems of the current Internet routing and addressing systems by changing the semantics of IP addresses [QIdLB07]. In the current Internet, an IP address has two roles. First, it is used by the applications running on a host to identify (this is the *identifier* role of an IP address) the endpoint of the transport layer flows. Second, it is used to identify the datalink layer interface used to attach the host to the Internet. This is the *locator* role of an IP address. LISP decouples these two roles.

With LISP, identifiers are attached to endhosts. These identifiers are called *Endpoint Identifiers (EIDs)*. Usually, an EID prefix is assigned to each site that assigns one identifier to each host within the site. In contrast with the current Internet, EID prefixes are not advertised in the global routing system and thus core routers do not need to maintain routes towards all EID prefixes. With LISP, locators are assigned only to the routers that are either part of the core of the network or at the border of sites, these locators are called *Routing Locators (RLOCs)*. This principle explains why LISP limits the scalability issues of the Internet [QIdLB07].

Among the 335,000 IP prefixes that are advertised in today's Internet, 41% of them come from stub ASes like campus or enterprise networks. These number are computed from the University of Oregon Route Views Project [Uni]. If LISP was in use today on all campus and enterprise networks, those prefixes would not have to be advertised in the global routing system. Hence, the routing table would be reduced accordingly [QIdLB07]. Quoitin et al. show in [QIdLB07] that if the locators are carefully allocated, the number of FIB entries can be one order of magnitude smalled than the number of ASes. Remember that in the current Internet, the number of entries in the FIB is one order of magnitude larger than the number of ASes.

This chapter is organized as follows. First, Sec. 2.2 briefly discusses the principle of the locator/identifier separation at the host level. Then, we present in details how the locator/identifier separation works at the network level. To do so, Sec. 2.3 describes the Locator/Identifier Separation Protocol with its data plane in Sec. 2.3.1 and control plane in Sec. 2.3.2. LISP is positioned as an evolutionary approach in Sec. 2.4 with the need to be compatible with the current Internet (Sec. 2.4.1) and to be incrementally deployable (Sec. 2.4.2). Afterward, we analyze the reachability problem that is inherent to the tunnel-based solutions in Sec. 2.5. Thereafter, Sec. 2.6 presents how network virtualization and segmentation is performed with LISP and discuss experiences from LISP deployments in Sec. 2.7. Finally, Sec. 2.8 concludes this chapter.

The original contribution of this chapter is to provide a tutorial description of LISP and historical feedback about the LISP protocol.

2.2 Locator/Identifier Separation at the host level

In this thesis, we consider how networks can handle path diversity and control their incoming traffic. However, in this section, we take time to study how path diversity can be handled at the host level. Indeed, separating the locator and the identifier role of IP at the network level offers two major advantages. First, the end-hosts do not need to be changed. Second, the network operator have more cost-effective multihoming and a better control over its traffic engineering. However, it is likely that if an end-host can access two different networks simultaneously, e.g., 3G and WiFi, the networks are belonging to different operators. With a network based locator/identifier separation, the end-host cannot have benefits in this situation. To let the hosts dynamically control the paths they are using when they are multihomed, separating the two roles of IP at the host level is an advantage. Host based loc/id split follows a principle of not exhibiting to the upper network stack levels the IP address effectively used to transfer or receive packets. To do so, a shim layer is added between the layer i in the stack where the locators are managed and the layer $i + 1$ in the stack where only identifiers are of interest. The role of the shim layer is to associate identifiers and locators and to send the packets by using the appropriate locators.

The host can separate the identifier and locator roles at the layer 3 or the layer 4 of the network stack.

Layer 3 protocols ensure that whatever the locator used, the IP address appearing at the layer 4 will remain the same for the life of a given flow. Three classes of solution exist, illustrated by the three following protocols.

Host Identity Protocol (HIP) defined in [MN06] generates an identifier for the hosts based on cryptographic methods. The identifier can be a key. However, because the key can be generated from different techniques and that the key can be of any length, a *Host Identity Tag (HIT)* must be deduced from the key to generate an number that is the length of an IP address. The HIT is used as the IP address presented at the layer 4 in the stack. When an HIP packet is sent over the network, its source and destination addresses are locators (e.g., IP addresses assigned to the network interfaces). The packets are tagged with the HIT to allow the hosts to map the packets to layer 4 flows.

Identifier-Locator Network Protocol (ILNP) proposed in [ABH10] separates the IP address in two. The 64 high order bits have the locator role while the 64 low order bits have the identifier role. The 64-bit prefix is considered for routing. On the contrary, only the 64-bit postfix is considered by the layer 4. When a segment must be send with ILNP, ILNP determines which locator to use. The locator is the prefix of the network interface. It then replaces the 64-bit prefix of the IP as specified by the layer 4 by the prefix of the interface used to send the packet. The destination address is replaced by using the same system, where the destination corresponds to a locator associated with the destination identifier. When a packet arrives at an ILNP host, the 64-bit prefix of both the source and destination is replaced by the 0x0000000000000000 value. Then whatever the locator used, the layer 4 always sees the same addresses for the flow. The identifier is constructed such that it is unique within the network. The same identifier part of the IP address is assigned to all the addresses of all the interfaces of the host.[1]

Shim6 defined in [NB09] use IP addresses at the same time as locators and identifiers. When a flow starts, it selects an IP address for the source and for the destination. For the life of the flow, these two addresses will be the identifiers for the flow. When a segment must be sent via shim6, it determines the most appropriate IP addresses to use for the sending of the packet. The selected locators can be the addresses used by as identifiers. To determine the shim6 context used to determine the identifier from the locator, the packet also carries a Shim6 context identifier.

Shim6 and HIP have been designed with IPv6 and security in mind. From a security point of view, the 128-bit of IPv6 address allow more flexibility. The addresses are long enough to embed information that authenticate the origin of a packet. In addition, IPv6 allows the hosts to generate as many addresses as they need. For example, in Shim6, the addresses are generated with HBA [Bag09] or CGA [BA06]. With CGA and HBA, the addresses are generated such that they prove that the host that generated the address is the owner of a given certificate. In HIP, the locator addresses are normally generated addresses, but the identifier is a key generated from a certificate owned by the hosts. The identifier address is derived from the key to generate the HIT. In both Shim6 and HIP, the packets are tagged with an information that allows the receiver to associate a packet with a flow and to check its authenticity. Both HIP and Shim6 need a sort of control protocol to determine the certificate and the identifier of the hosts to exchange packets with. ILNP simply relies on IPv6 routing and does not embed such advanced security features. In all the protocols, a mechanism is defined to let the two end-hosts learning the possible locators for the identifier. This information can be retrieve directly at the protocol level (e.g., Shim6) or can rely on DNS (e.g., ILNP).

The locator/identifier separation can also be done at the transport layer (layer 4) of the stack. In this case, the transport protocol is aware of the different addresses it can use and ensures the segments of a flow to be carried over theses addresses. We consider two layer-4 protocol that perform a

[1]ILNP also exists for IPv4.

loc/id split, the *Stream Control Transmission Protocol (SCTP)* [Ste07] and the *MultiPath Transport Control Protocol (MPTCP)* [FRH$^+$11]. The major difference between these two protocols is that MPTCP is transparent for the applications (seen as TCP) while they have to be changed to use SCTP.

SCTP works with the notion of streams and a stream is always between two hosts. An SCTP segment contains information about the stream it belongs to. When an SCTP segment is received it is transmitted as is from the lower layer to the layer 4. At the layer 4, the SCTP instances maintains the association between the stream and the application that uses it and can transmit the service data unit to the correct service that uses SCTP. SCTP segments are normally directly transported over IP but in some situation (e.g., firewall or NAT) they can be carried over UDP.

MPTCP is designed to be compatible with TCP. Therefore, an MPTCP host is able to exchange segments with a TCP host. In this situation, the traffic is the vanilla TCP. In addition, to ensure the protocol to be transported transparently over the Internet, it is seen as TCP with specific TCP options. When two MPTCP hosts establish a MPTCP flow, several TCP sub-flows are open. MPTCP is then in charge of determining to which flow belongs a segment received from a sub-flow. MPTCP must also ensure the scheduling of the data unit among the sub-flows. Indeed, the sub-flows can follow different paths with different performances. MPTCP must minimize the re-ordering by sending the data via the most appropriate sub-flow. Finally, MPTCP must manage the congestion over the different sub-flows. MPTCP segments are TCP segments with particular TCP options.

In both MPTCP and SCTP, data can be sent over different paths, independently of the identifier of the flow used at the upper layer. In both protocol a signaling mechanism is used between the two communication end-points. Each end-point signals the other end the addresses that can be used for the communication. In MPTCP, two solution exists. Either on option is added to the segments to signal the other possible addresses to use. Or the end-point spontaneously sends a new SYN from its to the other end-point to open a new sub-flow. MPTCP can determine that the SYN is related with a flow because a special TCP option is added to indicate to which MPTCP flow the SYN is related.

In general, MPTCP and SCTP are considered as multipath technologies more than loc/id split mechanisms. Indeed, the very purpose of MPTCP and SCTP is to provide high performance for large data transfers. To reach this objective, the two protocols have been designed to ensure loc/id split.

With IPv6, it is possible to benefit from the separation at the host level even if the host is connected only with one interface to the network. Indeed, if the network is in PA mode and is multihomed, it receives one IPv6 prefix per provider, the prefixes are propagated to the hosts and the hosts have thus one address per provider. The hosts are then free to chose the IP addresses they want to use and thus control themselves the path to send and receive traffic. Unfortunately, IPv6 does not provide any mechanism to deal with multiple addresses hence the existence of ILNP and shim6.

2.3 The Locator/Identifier Separation Protocol (LISP)

The *Locator/Identifier Separation Protocol (LISP)* [FFML10a, Mey08] relies on three simple principles: *address role separation*, *encapsulation* and *mapping*.

Address are separated in two roles: the *Routing Locators (RLOCs)* and the *Endpoint Identifiers (EIDs)*. The RLOCs are assigned from the *RLOC Space* to the Internet Service Providers. The EIDs are attributed by block extracted from the *EID Space* to the *stubs*. A stub is a network that only carries traffic from and to itself, e.g. an enterprise or campus network. To limit the scalability problem of today's Internet, only the routes towards the RLOCs are announced on the Internet while EIDs are also propagated today.

LISP-enabled routers are used at the boundary between the EID and the RLOC spaces. Routers used to exit the EID space are called *Ingress Tunnel Router (ITRs)* and those used to enter the EID space the *Egress Tunnel Routers (ETRs)*. When an host sends a packet to a remote destination, it sends it as in today Internet. The packet eventually arrives at the border of its site at an ITR. Because EIDs are not routable on the Internet, the packet is encapsulated with the source address set to the ITR RLOC and the destination address set to the ETR RLOC. The encapsulated packet is then forwarded in the RLOC space until it reaches the ETR. The ETR then decapsulates the packet and forwards it to the destination. The acronym xTR for Ingress/Egress Tunnel Router is used a router playing these two roles.

The correspondence between EIDs and RLOCs is given by the *mappings*. When an ITR needs to find the ETR RLOCs that serve an EID it queries the *mapping system*.

The encapsulation part of LISP is commonly called the *data plane* and the mapping part is called the *control plane*.

Fig. 2.3 shows an example of LISP deployment. In this example, the host with EID 153.16.35.6 sends packets to the host on the right. The source host generates an IP packet with source address 153.16.35.6 and destination address 153.16.36.9 and sends it on its local network. Because the destination address is not within the local network, the packet eventually reaches a network's border router. The border router identifies that the destination is an EID. It then runs into the ITR mode. It first queries the mapping system to retrieve the locators associated with this EID. The possible locators are 192.0.2.5, 192.0.2.254 and 2001:bd8:beef::7. The ITR chooses the 2001:db8:beef::7 locator. It then encapsulates the native IP

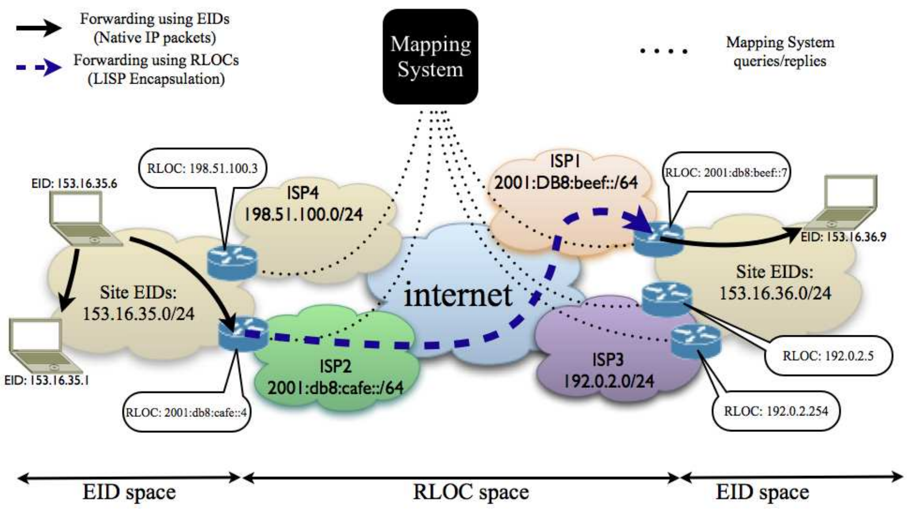


Figure 2.3: LISP general behavior

packet it received from the source host, into a LISP packet whose destination IP address is the chosen locator, i.e., 2001:db8:beef::7. The source address of the LISP packet is set to the ITR's locator, i.e., 2001:bb8:cafe::4. The LISP packet is then sent over the Internet and eventually arrives the ETR with locator 2001:db8::beef:7. When this ETR receives the LISP packet, it decaspulates it and forwards the inner packet to the destination host 153.16.36.9 as it belongs to the EID prefix 153.16.36.0/24 it is responsible for. At the opposite, the host with EID 153.16.35.6 uses only native IP forwarding to send packets to host with EID 153.16.35.1 as the two hosts are within the same LISP site.

Sec. 2.3.1 describes the LISP data plane and Sec. 2.3.2 explains the LISP control plane.

2.3.1 LISP Data Plane

LISP xTRs exchange encapsulated packets. A LISP-encapsulated packet is an IP packet whose source and destination addresses are EIDs and that is placed inside an IP packet whose source and destination addresses are RLOCs. The outer header (containing the RLOCs) is added by the source ITR and is used by the core routers to forward the packet. This header is finally removed by the destination ETR. The inner header (containing the EIDs) is processed by the ETR and is forwarded inside the site.

The LISP working group concluded that the best solution was to carry the LISP header over UDP. This designed choice is explained in Sec. 2.4.1

Fig. 2.4 shows the LISP data plane header. This header is followed by the inner header and its associated payload. The N, L, V and I bits

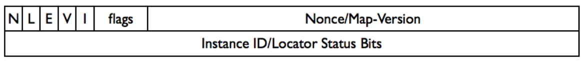


Figure 2.4: LISP Data-Plane header format

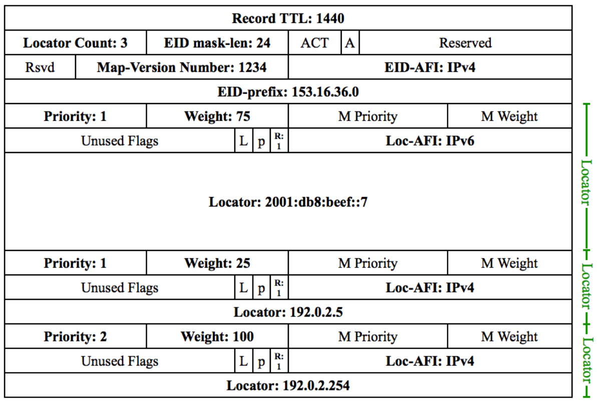


Figure 2.5: EID-to-RLOC mapping Example

determine the LISP header interpretation. They respectively indicates the presence of the Nonce, Locator Status Bits, Map-Version and Instance ID fields. Finally, The E bit is used in the Echo Nonce Algorithm. The N, L and E bits as much as Nonce and Locator Status Bits fields are described in Sec. 2.5. The N and V bits are mutually exclusive while both L and I bits can be set simultaneously.

The Map-Version field is used by the ITR to inform the ETR about EID-to-RLOC mapping version it used for the encapsulation. More details are given in Sec. 2.3.2.

The Instance ID is used for virtualization and segmentation, a detailed description of its usage is given in Sec. 2.6.

2.3.2 LISP Control Plane

The main component of the LISP control plane is the *EID-to-RLOC Database* that contains the EID prefix to RLOC mappings. Fig. 2.5 shows how this mapping is constructed with an example from Fig. 2.3 right-hand network.

An EID-to-RLOC mapping associates an EID prefix, i.e. a block of EID addresses, to a list of RLOCs. Each RLOC being associated with a priority, a weight and a reachability.

A priority is used to determine which RLOC must be preferred to reach the EID prefix. The RLOC with the lowest priority value is preferred. If several RLOCs have the same lowest priority value, a weight value determines how ITRs should balance the load among them. ITRs must use a load balancing algorithm that ensures that packets from the same EID-to-EID flow are sent towards the same ETR. Furthermore, the reachability bit (R bit) indicates whether the RLOC is reachable. However, this information is a hint an the effective reachability should be tested (see Sec. 2.5 for more details). Unreachable RLOCs are removed from the set of RLOCs to use for the encapsulation. Priority and weight are used for performing incoming traffic engineering.

LISP defines multicast mappings (M Priority and M Weight), but they are still at their early development stage and are not discussed here.

The LISP control plane allows the ITRs to obtain the EID-to-RLOC mappings needed to encapsulate packets towards the appropriate destination ETR. ITRs interact with the control plane with the *Map-Request* and *Map-Reply* messages. ITRs send Map-Requests to the mapping system to get the mappings. The control plane answers with a Map-Reply containing the EID-to-RLOC mapping. The Map-Request mostly contains a unique identifier (the *nonce*) and the requested EID. The Map-Reply contains the EID-to-RLOC mapping for the requested EID and echoed the nonce from the Map-Request. In general, mappings are related to prefixes, not to addresses. It is then common to Map-Reply a Map-Request for an EID address with a mapping for the EID prefix that covers the requested address. The Map-Reply can carry several EID-to-RLOC mappings in one message. For example, if the Map-Request is for the EID 153.16.36.1 and that the destination site has the mappings for both 153.16.36.0/24 and 153.16.36.128/28, the two EID-to-RLOC mappings have to be sent. Indeed, if the reply only returns 153.16.36.0/24 as it is the only prefix that covers 153.16.36.1, this entry would also be used by the ITR for the EID prefixes that belong to the 153.16.36.128/28 EID prefix which would lead to inconsistencies. This problem is very similar to what is faced with FIB compression [FFM$^+$07]. It is possible to avoid this by sending prefixes that are not overlapping others. In the example, the Map-Reply could be for the 153.16.36.0/25.

Mapping are time limited with a TTL expressed in minutes, in contrast with DNS TTL in seconds. This TTL helps ITRs to cache the mappings in their EID-to-RLOC cache. This cache avoids the ITR to send a Map-Request for each packet to encapsulate. A TTL of zero minute means that the mapping cannot be cached. On the contrary, a mapping with an infinite TTL (i.e., 0xffffffff) can be store forever.

However, an EID-to-RLOC mapping owner may change it before the provisioned expiration (e.g., after a network failure). To this aim, mappings are labeled with a version number, the *Map-Version*. Upon a change, the mapping's Map-Version is incremented by one. When an ITR sends an encapsulated packet, it can label the data plane packet with Map-Version of the mapping used for the encapsulation. If the ETR detects an outdated version, it can inform the ITR of the change by sending it a Map-Request with the *Solicit Map-Request (SMR)* bit set. The SMR Map-Request contains the EID prefix that has a new Map-Version. Upon SMR Map-Request reception, the ITR sends a Map-Request for the SMRed EID prefix. The Map-Request can be sent directly to the SMRing ETR or via the Mapping-System.

Mapping System

The mapping system is a major component of the Locator/Identifier Separation Protocol as it provides the association between an identifier and its locators.

From an architectural standpoint there are two possible ways in which a mapping system could supply mapping information. It can either provide individual answers to specific requests (*pull*), or distribute (*push*) all the mappings onto listeners. In pull-based mapping systems, the ITR sends queries to the mapping system every time it needs to contact a remote EID and has no mapping for it. The mapping system then returns a mapping for a prefix that contains this EID. Pull-based mapping systems have thus similarities with today's DNS. Proposed pull-based mapping systems include LISP+ALT [D. 09], LISP-Tree [JCAC$^+$10], LISP-CONS [BCF$^+$08], FIRMS [MHH10] and LISP-DHT [MI08]. Pull-based mapping systems suffer from *mapping lookup latency.* The ITR needs to have the mapping for the destination EID stored in its EID-to-RLOC cache to LISP-encapsulate a packet to an ETR. If the mapping is not in the EID-to-RLOC cache, the ITR queries the mapping system to get the mapping. As long as the mapping is not retrieved, the encapsulation cannot be done. To limit the impact of mapping lookup latency, *Data-Probe* has been proposed. Data-probing assumes that EIDs are routable on the mapping system. With Data-Probe, the packet is LISP-encapsulated but the outer destination address is the destination EID. The encapsulated packet is sent over the mapping system. When such a packet arrives at and authoritative ETR, it is decapsulated and transmitted to the destination endpoint and, at the same time, the ETR sends a Map-Reply the the ITR that encapsulated the packet. Once the mapping is known by the ITR, encapsulation is done normally and data-probing is not used anymore. This technique avoids the ITR to have to wait for a mapping but it mixes the control plane and the data plane and could cause the control plane to collapse if not used with care. In push-based mapping systems, the ITR receives and stores all the mappings for all EID prefixes even if it does not contact them. Push-based mapping systems have thus similarities with today's BGP. To the best of our knowledge, NERD [Lea08] is the only proposed push-based mapping system. Push-based mapping systems do not suffer from mapping lookup latency, but suffer from churn: the ITR does not control the amount of received mappings neither the rate at which these mappings are received. To limit the churn, push-based mapping system can use a *publish/subscribe* mechanism. To receive the mappings for an EID prefix, the ITR has first to subscribe for it. However, this solution combines the drawbacks from both push-based and pull-based solutions.

Two special devices can be used to simplify the operation of some mapping systems:

Map-Resolver (MR) is contacted by the ITRs to get the EID-to-RLOC mappings [FFML10a, FF09]. In this case, the MR does the resolution on the mapping system on behalf of the ITR. The ITR sends Map-Requests to its MR and receives Map-Replies either from the MR or directly from an authoritative ETR.

Map-Server (MS) announces the EID prefixes on behalf of the ETRs that are registered to them meaning that these ETRs are not part of the mapping system [FFML10a, FF09].

When a Map-Resolver is used, the Map-Requests are LISP-Encapsulated to the Map-Resolver. The encapsulation is done with the special *Encapsulated Control Message* LISP control plane message [FFML10a, FF09]. When such a message arrives the MR, it is first decapsulated and then transferred to the mapping system by the Map-Resolver.

The Map-Server functionality introduces the *Map-Register* control plane message [FFML10a, FF09]. A Map-Register message is composed of the EID prefix to register to the mapping system and is authenticated with a Message Authentication Code (MAC) based on a secret key shared between the ETR and the Map-Server.

The following of this section presents the major proposed LISP mapping systems: LISP+ALT, LISP-Tree, LISP-DHT, LISP-CONS and NERD.

LISP+ALT As of today, the LISP community is focused on the *LISP+ALT* mapping system [FFML10b]. LISP+ALT relies on a BGP overlay. In LISP+ALT, the ETR stores the mappings for which they are authoritative for. The overlay is constructed by connecting ETRs together via tunnels, for example GRE tunnels. This ETRs overlay is called the *Alternative Logical Topology (ALT)* where routers are called *ALT routers*. ALT routers maintain a BGP session with their neighbors and announce the EID prefixes they are authoritative for, making the EIDs routable in the ALT. At this point it is worth to notice that BGP is used to build the ALT, not to announce the EID-to-RLOC mappings. To get a mapping, an ITR sends a Map-Request for the EID on the ALT topology. The source address for the Map-Request is the ITR RLOC and the destination the EID. The Map-Request eventually reaches an originator ETR for the EID prefix that matches the destination EID. This ETR resolves the EID and sends a Map-Reply directly to the ITR RLOC. Map-Replies are not sent on the ALT.

LISP-Tree LISP-Tree is a pull-based hierarchical mapping system that we are proposing. A detailed LISP-Tree description and study is presented in Chapter 4. LISP-Tree is a DNS-like system with a hierarchy of LISP-Tree Servers, *LTS*, and Mappings Servers. The hierarchy is maintained as a tree where each node is responsible of a part of the EID space. The child of a node is responsible of a part of the EID space its parent is responsible for, recursively. Mapping information are only stored at the leaves of the tree. Intermediate nodes only maintaining pointers to their children. When a mapping has to be retrieved for an EID prefix. A root server is queried first. By definition, a root server is responsible for the all EID space and has a children responsible for the EID to resolve. The root replies with a pointer to one of its children that is responsible for the EID prefix to resolve. The process is repeated with the returned child considered as the root of the sub-tree where a mapping can be retrieved for the EID prefix. This process is repeated until a leaf is reached. In LISP-Tree, leaves are Mapping Servers (MS). MS maintain the list of ETR authoritative for the EID prefix registered to them. The tree browsing stops and the mapping is retrieved by sending a Map-Request to one of the ETR authoritative for the EID prefix. To speed up the process, the mapping resolution is not performed directly by the ITRs but by a Map-Resolver. MR does the mapping resolution for the ITRs and can cache intermediate steps in the resolutions. Like in DNS, the resolution can be performed either recursively or iteratively.

LISP-DHT LISP-DHT [MI08] is a mapping system based on a Distributed Hash Table (DHT). The LISP-DHT mapping system uses an overlay network derived from Chord [SMLN+03]. Choosing this particular structured DHT over others (e.g., CAN, Pastry, Tapestry or Kademlia) was motivated by the algorithm used to map search keys to nodes containing the stored values. In a traditional Chord DHT, nodes choose their identifier randomly. In LISP-DHT, a node is associated to an EID prefix and its Chord identifier is chosen at bootstrap as the highest EID in that associated EID prefix. This enforces mapping locality that ensures that a mapping is always stored on a node chosen by the owner of the EID prefix, see [MI08] for details. When an ITR needs a mapping, it sends a Map-Request through the LISP-DHT overlay with its RLOC as source address. Each node routes the request according to its finger table (a table that associates a next hop to a portion of the space covered by the Chord ring). The Map-Reply is sent directly to the ITR via its RLOC.

LISP-CONS The Content distribution Overlay Network Service for LISP, LISP-CONS [BCF+08], is a hierarchical content distribution system for EID-to-RLOC mappings. It is a generalization of LISP+ALT, which does not use the BGP routing protocol. On the other hand, it adds support for caching in intermediary nodes. In this thesis we do not consider LISP-CONS as it does not seem to evolve anymore.

NERD The Not-so-novel EID to RLOC Database, NERD [Lea08] is a flat centralized mapping database, using the push-model. The NERD database is distributed to a group of well known servers with their public certificates known by all the ITRs. The certificates are used to validate the database stored on the servers. When an ITR starts, it downloads the NERD database from the NERD servers, it then periodically polls the servers to obtain the changes since the last query. To keep track of the changes, the servers increment the database version number at every change. Because any change requires a new version of the database to be downloaded by all ITRs, this approach is unlikely to scale to the needs of a future global LISP mapping system, it could however be used in controlled environments. The main advantage of NERD and its push-based like model is the absence of cache misses that could degrade traffic performance.

2.3.3 On the EID-to-RLOC Cache size

Quoitin et al. show in [QIdLB07] that the locator/identifier separation improves the routing scalability by reducing the FIB size and increases the path diversity and thus the traffic engineering capabilities. In addition, Iannone and Bonaventure show in [IB07] that the number of mapping entries that must be supported at an ITR of a campus network is limited and does not represent more that 3 to 4 Megabytes of memory. Similarly, Juhoon et al. show that the EID-to-RLOC Cache size should not exceed 14 MB for an ITR responsible of more than 20,000 residential ADSL users at a large ISP [KIF11]. [IB07, KIF11] rely on BGP and traffic traces to determine the number of entries to install on the cache. In both papers, the size of the cache is inferred from the number of entries by considering that every EID is associated with two or three locators. In this section, we confirm these results by showing the distribution of the number of locators per EID if LISP were deployed in today's Internet. To do so, we rely on three assumptions to determine the EIDs and their associated RLOCs: (*i*) as contiguous addresses tend to be used similarly [CH10], EID prefixes follow the current BGP prefixes decomposition; (*ii*) EIDs are used only at the stub ASes, not in the transit ASes; (*iii*) the RLOCs of an EID are deployed at the edge between the stubs owning the EID and the providers and locator addresses are allocated in a *Provider Aggregetable (PA)* mode. Assumptions (*i*) and (*ii*) are the same as those in [IB07, KIF11, QIdLB07]. Assumption (*iii*) is similar to [QIdLB07].

Fig. 2.6 illustrates the methodology for our EID-to-RLOC association. In Fig. 2.6, the blue clouds represent transit ASes with locators. The green clouds are used for stub ASes that use LISP and contain EIDs. The routers

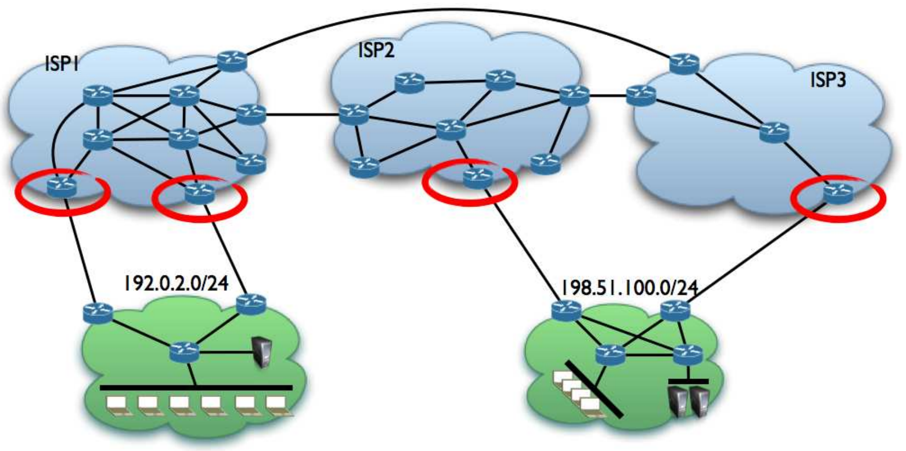


Figure 2.6: EID-to-RLOC association example

at the border between the EID space and the RLOC space are circled in red. The stub AS `192.0.2.0/24` is multiconnected to the AS `ISP1`. The stub AS `198.51.100.0/24` is multihomed to `ISP2` and `ISP3`. Both `192.0.2.0/24` and `198.51.100.0/24` appear to have two active locators.

We base our evaluation on two datasets. First, the BGP prefixes and the AS classification (stub versus transit) is extracted from the Route Views Project dataset [Uni] (labeled as *Route Views* in the remainder of this chapter). We use a BGP dump from Oregon-IX collected on August, 12th 2010. Second, the mapping between BGP prefixes and locators is done with the Archipelago dataset [cHKF09] (labeled as *Ark* in the following) collected on July, 24th 2010. For the analysis, only the completed traceroutes are considered in the Ark dataset.

Following the first two assumptions (see above), we filter the prefixes from Route Views to extract the stub prefixes using the AS ranking provided by CAIDA [CAI10]. We further take only into account the most specific prefixes. Less specific prefixes, typically used for resiliency, are thus filtered. By doing so, we have an upper bound on the number of EID prefixes. Indeed, filtering the less specific prefixes does not filter the prefixes deaggregated for load balancing reasons. After filtering out the Route Views dataset, 41.21% of the prefixes (i.e., 138,123 prefixes) are considered as EID prefixes.

To determine the active locators for each EID prefix, we rely on the Ark dataset and the third assumption. In this chapter, we consider the edge from an IP routing point of view: a router at the edge of a stub is the first router whose address is not inside the EID prefix. For each destination address in the Ark dataset, we determine its most specific BGP prefix, i.e., its EID prefix, and backtrack the traceroute until we reach an address that does not

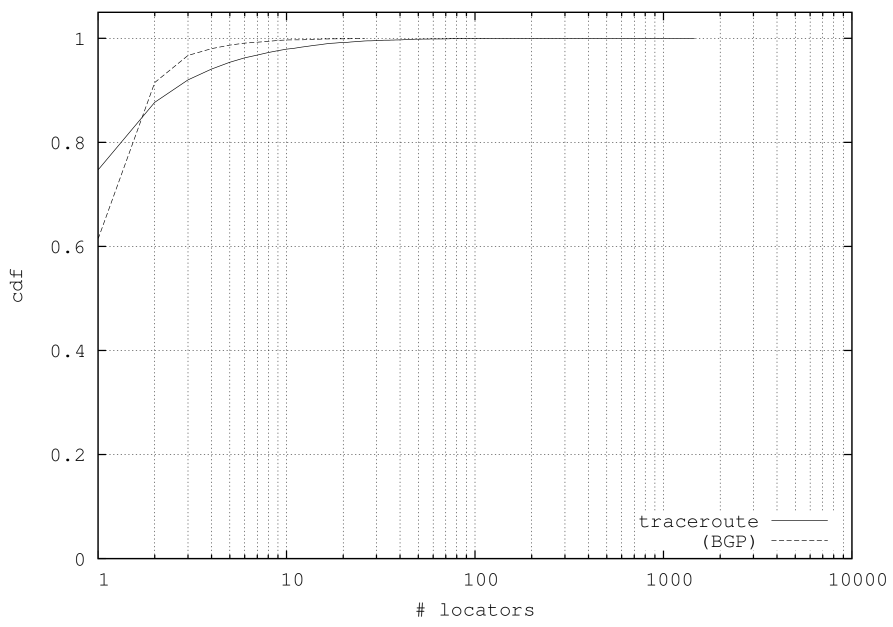


Figure 2.7: RLOC distribution

belong to this BGP prefix. This address is considered as one of the RLOCs of the EID prefix. Computing this on all the completed traceroutes from the Ark dataset gives the list of RLOCs associated to a given EID prefix. As EID prefixes determined with Route Views are not always traceroutable, we were only able to extract 15,337 EIDs.

Fig. 2.7 shows the degree of multihoming and multiconnectivity of the EIDs. An EID is multihomed if it has several neighboring ASes. Multihoming is important for the robustness of the site but from a performance viewpoint, the locators diversity, i.e., the multiconnectivity, is more important than the diversity of neighboring ASes, as long as it increases the number of locators and thus the path diversity [QIdLB07].

The curve labeled `traceroute` on Fig. 2.7 provides the distribution, as a cumulative mass, of the number of locators (horizontal axis, in log-scale) associated to each EID prefix. Among the whole set of 15,337 EIDs, 3,880 (i.e., 25%) have at least two RLOCs and 1,886 (i.e., 12%) have more than two RLOCs. The distribution of the EID multihoming degree is presented with the curve labeled `BGP` on Fig. 2.7. In the BGP case, the number of locators corresponds to the number of neighboring ASes. The comparison between the two curves shows that the multihoming degree is lower than the multiconnectivity degree for the EIDs. For example, 22.8% of the EIDs have at least three locators, while only 8.4% of the EIDs are multihomed to more than two different autonomous systems. The limited number of traceroutable EIDs does not allow us to determine the size of the cache with the exact number of locators for each EID prefix. However, the distribution of the number of locators in Fig. 2.7 shows that the assumptions made in [IB07] and [KIF11] for the number of locators is acceptable. The study of the number of locators with traceroutes does not show the inactive locators. It is then possible that the number of locators is higher than what we observed. However, the inactive locators do not have to be installed in the EID-to-RLOC Cache used for the data plane encapsulation. Indeed, these locators are not used for the forwarding except if none of the active locators is reachable. The inactive locators can then remain stored in the slow but inexpensive control plane memory of the router.

2.4 LISP as an evolutionary approach

Some people advocate that the Internet has several flaws and a *clean-slate* approach must be followed to design a new Internet. At the opposite, others are defending an *evolution* approach [RD10]. In the clean-slate approach, the future Internet is designed from a blank sheet. At the opposite, the evolutionary approach uses the Internet as it is today and modifies it to make it better.

LISP is an evolutionary approach for scaling the Internet. Because of this, it has been designed to be able to be progressively deployed. To do so, LISP does not requires any change neither at the core nor in the stub sites. Only the border between the stubs and the core has to be changed. It thus mean that LISP packet have to be able to cross the core Internet (Sec. 2.4.1) but also to be incrementally deployed (2.4.2).

2.4.1 Core Internet Traversal

LISP uses encapsulation to transport EID-to-EID packets over the Internet where EIDs are not routable. However, even if LISP has always been designed with encapsulation, the method chosen to do the encapsulation has evolved during the standardization process and it is worth discussing this evolution. The first LISP specifications used IP-in-IP encapsulation, i.e., the outer IP header is directly placed in front of the inner IP header by the ITR. Although IP-in-IP encapsultation minimizes the overhead, it causes several problems. First, many deployed middle-boxes (e.g., firewalls, Network Address Translators (NAT)) block IP packets not transporting UDP or TCP traffic. Second, thanks to the utilization of *Equal Cost MultiPath (ECMP)* [Hop00], core routers often load balance the flows over several paths by hashing the source and destination addresses as well as the UDP/TCP port numbers [AFT07]. Using IP-in-IP thus limits the effectiveness of ECMP based load balancing. Finally, IP-in-IP does not have room for carrying LISP specific information.

For these three reasons, LISP encapsulates EID-to-EID packets over a specific LISP header that is carried over UDP. UDP is transported by RLOC-to-RLOC IP packets. Using the UDP header helps to pass through the middle-boxes. When a protocol uses UDP, it can either use random port numbers or apply for a reserved port number. LISP uses two reserved UDP port numbers: 4341 and 4342. 4341 (resp. 4342) is used as the destination port number for data (resp. control) plane packets sent by a LISP node. However, the UDP header also contains a source port number. The first LISP specifications used random source port number in each encapsulated packet. However, it was not appropriate for LISP as packets belonging to the same EID flow could have different source ports and thus be forwarded over different paths because of ECMP and potentially be reordered before reaching their final destination. To ensure that all the packets from the same EID-to-EID flow follow the same path, the source port is computed as a hash of the EID-to-EID packet source and destination EIDs and ports.

2.4.2 Incremental Deployment

The transition from the current Internet to a LISP enabled Internet will imply the coexistence of *LISP sites* and *non-LISP sites*. On one hand, LISP sites are expecting encapsulated packets which cannot be created by non-LISP sites. On the other hand, non-LISP sites are not able to decapsulate LISP packets, or even to forward traffic to an EID as EIDs are not globally routable. Fortunately, the encapulation principle enables a transition mechanism. As a matter of fact, a non-LISP site can exchange packets with a LISP site as long as the traffic is LISP encapsulated somewhere in the network and a LISP site can exchange traffic to a non-LISP site as long as the LISP traffic is decapsulated before arriving at the final destination. All this transition is transparent for the traffic endpoints. This transition relies on *Proxy ITRs (PITR)* and *Proxy ETRs (PETR)*. PITR are particular ITRs that attract non-LISP traffic destined to LISP sites to encapsulate and forward it to the LISP destination. At the opposite, PETR are particular ETRs that attract LISP traffic destined to non-LISP sites to decapsulate and forward it to the non-LISP destination. The PITR/PETR transition mechanism is deployed in the LISPx.net network presented in Sec.2.7.

2.5 Reachability Problem

The advantage of flexibility and control offered by the loc/ID separation comes at the cost of increasing the complexity of the reachability detection. Indeed, identifiers are not directly routable and have to be mapped to locators. But a locator may be unreachable while others are still reachable. This is an important problem for any tunnel-based solution. To understand it, let us analyze the situation shown in Fig. 2.8.

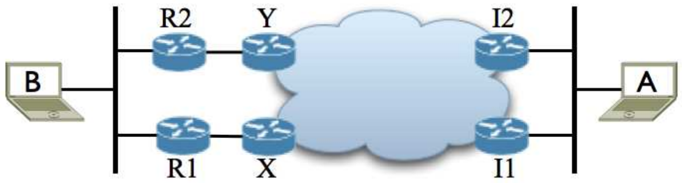


Figure 2.8: LISP reachability problem example

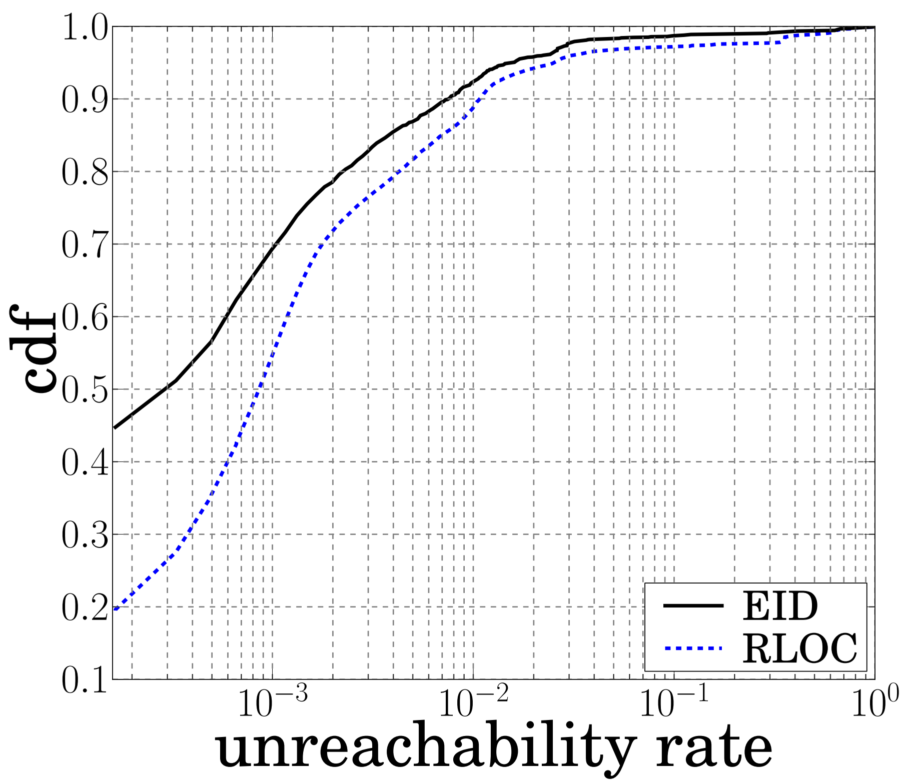


Figure 2.9: EID vs RLOC unreachability rate

In Fig. 2.8, host `A` is attached to the right-hand LAN that exchanges packets with host `B` attached to the left-hand LAN. In a normal IP network, either a campus/enterprise network using an intradomain routing protocol or the global Internet using BGP and intradomain routing, A sends its packet to its default router (e.g. I1). This router consults its forwarding table and forwards the packet along the best path towards the prefix that includes `B`'s address. Assume that `R1` is along this best path. If link `X-R1` fails, the routing protocols detect the failure and after some time `I1` learns that the only path to reach `B` is via `R2`.

Fig. 2.9 illustrates the end-to-end reachability problem. We measured the RTT with all the locators obtained in Sec. 2.3.3 and a machine in our campus. The measurement campaign started on September 3rd 2010 and lasted until September 24th 2010. The measurements have been collected in cycles, one cycle every 5 minutes, every locator being measured during each cycle. The delay is the result of the ping command. A locator is considered as not reachable if its RTT is longer than one second. The unreachability rate is defined as the ratio of the number of measurements that failed to the total number of measurements. An EID is unreachable if none of its locators is reachable. Fig. 2.9 shows that the locators are unreachable more frequently than the EIDs. It means that all the locators of an EID are seldom unreachable at the same time. This observation confirms that multihoming or multiconnectivity increases the robustness of a network. Unfortunately, in LISP, the packets for a network are sent to a particular border router of that network (an ETR). This means that the sending ITR must be able to recover from the failure of an ETR by itself. Fig. 2.9 shows that the choice of the locator for the encapsulation is of prime importance to ensure the traffic liveliness. A simple solution to solve the reachability problem is to let the ITR change its mapping upon the reception of an ICMP destination unreachable message. However, ICMP messages are often rate limited or even not sent. In addition, spoofing ICMP is easy. Therefore, several mechanisms has been defined to allow an ITR to verify the reachability of a remote ETR:

active probing consists in sending explicit probes to determine whether a locator is reachable or not. LISP uses the *RLOC Probing Algorithm* for this [FFML10a]. The ITR periodically sends a Map-Request with the probe-bit set to the ETR with the locator to test, and the corresponding EID prefix. The ETR answers with a Map-Reply with the probe-bit set and the probed locator is flagged. Both Map-Requests and Map-Replies are exchanged directly via the locators, without using the mapping system. Active probing presents scalability issues as it requires the sending of specific packets.

data plane monitoring consists of monitoring the data plane traffic to determine whether a locator is reachable or not. LISP uses the *Echo Nonce Algorithm* for this purpose [FFML10a]. When a source xTR wants to test the reachability of an RLOC, it flags packets sent to this RLOC with the Echo-Nonce-Request bit (E) and the Nonce-Present bit (N) and tags the packet with a nonce. When the destination xTR receives such a packet, it stores the nonce in a cache and, when it sends packets back to the source xTR, it flags the packet with the N bit and copies the nonce received in the echo-nonce request. If the source xTR receives the packet with the correct nonce (and the E bit reset), the RLOC is considered reachable.

control-plane feedback uses information from the control-plane to determine the locator reachability. The *Locator Status Bits (LSB)* are used for that purpose [FFML10a]. The LSB is a 32-bits vector in the data plane packets where the bit at position i indicates locator i reachability. If position i in the LSB is set, the RLOC is reachable, otherwise, it is not. The ITR builds the LSB from its local routing information. LSB information may thus be considered as an hint. When the status changes a reachability test described above can be used to check the LSB.

It is interesting to notice that LSB can be used in conjunction with the Instance ID (see Sec. 2.6). However, because they share the same 32-bit word, the first 8 bits are reserved for the LSB that is truncated to its first 8 bits. The last 24-bits being the instance ID that is always 24-bit long. Therefore, with instance ID, the LSB is limited to the first eight locators while up to 32 locators can be monitored otherwise.

The three techniques presented above can be combined to get a better understanding of the RLOC reachability. For example, the control-plane feedback can be used to rapidly filter the RLOCs that are not reachable because of global routing failures. Then, active probing can be used at low rate to determine if the backup paths are reachable or not and the data plane monitoring can be used to ensure that active paths are reachable.

2.6 Virtualization with LISP

Network virtualization allows to operate logical networks over one physical network. Provider-based *Virtual Private Networks (VPN)* are a way of virtualizing networks. Operators often use BGP/MPLS VPNs to offer VPN services to enterprises [RR06]. LISP was recently extended with the *Instance ID* [FFML10a] to be able to run several LISP instances on one xTR. The Instance ID enables LISP to provide virtualized services as well. The Instance ID is 24-bits long and can be a 802.1Q VLAN tag or VPN identifier.

Fig. 2.10 shows a simple LISP-based VPN. VPN1 and VPN2 use the same IP addresses. When ITR1 (resp. ITR2) encapsulates a packet from VPN1 (resp. VPN2), it labels it with the VPN1 (resp. VPN2) instance ID. When the ETR receives such a packet, it determines the forwarding table to use for the decapsulated packet thanks to the instance ID. The packet is then forwarded to the correct site. Without the instance ID, it would be impossible to determine the destination site to forward packets to as the same EID prefix is used by the two VPNs. Farinacci et al. [FMS10] go one step further and propose a *Canonical Address Format (LCAF)* to encode any kind of addresses in LISP control plane [FMS10]. With a canonical address representation, any address type can be used as EID (the address is

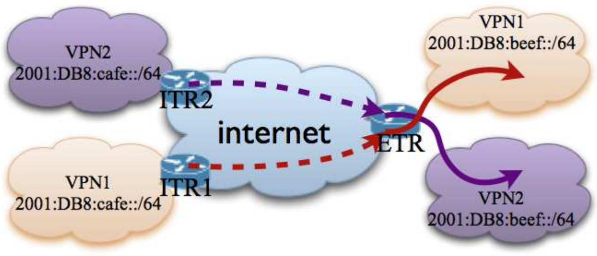


Figure 2.10: L3 VPN implementation with LISP instance ID. Both red and purple VPNs use the same EID prefix, the differentiation is done at the ETR based on the instance ID.

the key for the mapping lookup) and LISP can then transport, for example, Ethernet frames over the Internet.

2.7 Deployment lessons

The LISP effort is not limited to the specifications but also to the deployment of LISPx.net, a world wide experimental network (LISP4.net (resp. LISP6.net) for IPv4 (resp. IPv6) EIDs).

The LISPx.net is an experimental network composed of more than 80 xTRs spread over 13 countries (Latin-American, Asia/Pacific, European and US regions), operated by more than 20 company and university and powered by two different implementations (CISCO IOS and CISCO NX-OS). LISPx.net uses the LISP+ALT mapping system and most of the nodes rely on Map-Resolvers and Map-Servers. Hosts attached to LISPx.net can contact regular IP addresses thanks to the deployment of five PxTRs.

Deploying a new protocol in a real environment where any people can come and connect forces developers to improve it rapidly. The improvement ranges from debugging implementations to adding new features. The deployment also helps to refine the specifications to remove ambiguities. For example, the ITR load balancing algorithm was refined on the basis of LISPx.net experiments.

Deploying a public LISP network enables usages of LISP that were not expected. For example, Facebook experiments LISP to support IPv6 [LL10]. The Facebook infrastructure is mostly IPv4 but an IPv6 prefix is announced and plays the role of an EID. IPv6 packets arriving at the PITR are LISP-encapsulated and forwarded on the IPv4 Facebook backbone. When the ETR receives these packets it decapsulates and transmits them to a load balancer doing an HTTP IPv6-to-IPv4 translation. The translated packets are then injected in the IPv4 infrastructure.

2.8 Conclusion

Separating the locator role of IP addresses from the identifier role can help in improving the scalability of Internet routing and the traffic engineering capabilities. The Locator/ID Separation Protocol (LISP) separates these two roles at the router level. In LISP, the Internet is separated in the identifier and locator spaces. Locator (resp. identifier) addresses are routable only in the locator (resp. identifier) space. Identifiers are assigned to the stub networks while locators are used inside the core Internet. LISP uses a map-and-encap mechanism to allow stub networks to exchange packets via the core Internet although identifiers are not globally routable. LISP routers are deployed at the border between the stubs and the core Internet. When a packet leaves a stub it passes by such router and is encapsulated in a packet whose source and destination addresses are locators. When the encapsulated packet arrives the destination stub, it is decapsulated by a LISP router and forwarded in the stub until it reaches the destination. The locator to use to reach a destination stub is known thank to the mapping system that provides the mappings between the identifiers and the locators. The separation of the two roles of IP addresses can also be performed at the host level, which give the control over the path used for to packet to the hosts.

The initial goal of the locator/identifier separation paradigm was to make the Internet more scalable. However, as we see in the chapter, LISP goes beyond the scalability of the Internet and provides an elegant way of doing interdomain traffic engineering or network virtualization. LISP is still at its early stage. Multicast, mobility and security are open issues that are still being discussed.

Chapter 3

Interdomain traffic engineering with LISP

3.1 Introduction

Sec. 2.3 briefly presents the traffic engineering features embedded in LISP. However, to the best of our knowledge, no related work shows how to make traffic engineering with LISP. This probably comes from the lack of incentives for ISPs to comply with inbound recommendations of the distant networks. For that reason, we present in details how traffic engineering can be performed with LISP in this chapter. Chapter 1 shows how traffic engineering is performed in today's Internet with BGP. In this chapter we show how LISP can be used to perform traffic engineering. Sec. 3.2.1 shows how LISP can be used for outgoing traffic engineering and Sec. 3.2.2 shows that LISP can benefit incoming traffic engineering. Finally, Sec. 3.2.3 goes one step further and shows that the incoming traffic can be controlled on a per source basis with LISP. Chapter 7 even shows how traffic engineering can be automatized and how LISP can be used to make performance based traffic engineering.

The original contribution of this chapter is two-fold. On the one hand, we show how to use the mapping priorities, weights and TTL to perform incoming traffic engineering with LISP. On the other hand, we propose a solution to differentiate the mappings on a per-requester basis.

3.2 Traffic engineering with LISP

In today's Internet, stub networks are globally routable and the routing system distributes the routes to reach these stubs. On the contrary, the EID prefixes of a LISP site are not routable on the Internet and mappings are needed to determine the list of LISP routers to contact to send them packets. The difference is significant for two reasons. First, the packets are not sent to a site but to a specific ingress router. Second, a site can control the entry points for its traffic by controlling its mappings.

Mappings present in the EID-to-RLOC Cache are used to determine the best destination locator to use. In the same way, an ITR that has several RLOC uses the mappings from its EID-to-RLOC Database to determine the source locator to use for the encapsulation. From a purely technical point of view, the selection of the best source RLOC and the best destination RLOC are two instances of the same problem. Indeed, the RLOC decision process algorithm is the same for the source and the destination RLOC at an ITR. An ITR basically chooses the RLOCs with the lowest priority values as explained in Sec. 2.3.2. However, from an operational point of view, these problems are very different. On the one hand, the selection of the source locator only depends on local policies as the mappings in the EID-to-RLOC Database are built locally. But, on the other hand, the selection of the destination locator depends on external policies as the mappings in the EID-to-RLOC Cache are built by the remote site which causes a problem of incentives as show in the next sections. In this thesis, we focus on incoming traffic engineering. However, for the shake of completeness Sec. 3.2.1 briefly provides the general ideas to perform LISP outgoing traffic engineering.

The remaining of this section presents how traffic engineering can be instantiated with LISP.

3.2.1 Outgoing traffic engineering

Each ITR maintains in its EID-to-RLOC Database the mappings for the source EID prefixes it can encapsulates. If the source EID is valid, the ITR chooses one RLOC in the source mapping. The selection of the locator follows the same principle as the selection of the destination RLOC. The locator with the lowest priority value is preferred. If several RLOCs have the same priority the load is shared among them using the same per-flow hashing mechanism.

Using a mapping for selecting the source locator can be used to control the outgoing link to use. Priorities and weights can be adapted to fit the local policies. For example, if a link is overloaded, the weight attached to its RLOC can be lowered.

By nature, LISP provides source-based routing capabilities. Indeed, the source mapping associates a source EID prefix to a list of locators. In other words, the source EID determines the interfaces that are used to transmit traffic over the Internet. For example, an ITR can be responsible for a best effort and for a premium network. The two networks can be isolated from each other by assigning them different EID prefixes. The source mapping for the best effort prefix always using the cheapest outgoing link while the premium traffic uses the outgoing link that fits the service level agreements. Fig. 3.1 shows an example of source-based routing with LISP. In this ex-

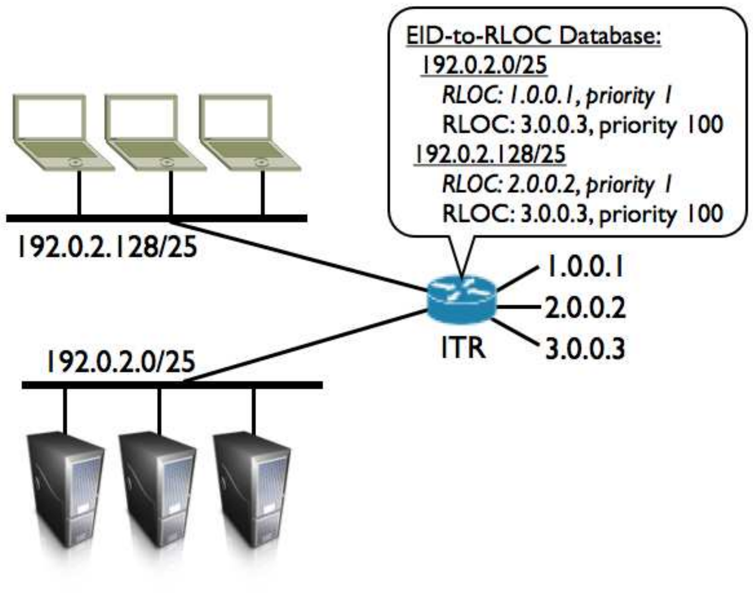

EID-to-RLOC Database:
192.0.2.0/25
RLOC: 1.0.0.1, priority 1
RLOC: 3.0.0.3, priority 100
192.0.2.128/25
RLOC: 2.0.0.2, priority 1
RLOC: 3.0.0.3, priority 100
192.0.2.128/25
192.0.2.0/25
ITR
1.0.0.1
2.0.0.2
3.0.0.3

Figure 3.1: Example of source-based routing with LISP. For 192.0.2.0/25, the source locator is 1.0.0.1 and for 192.0.2.128/25, the source locator is 2.0.0.2. Both networks have the 3.0.0.3 RLOC for backup.

ample, the ITR is responsible for the `192.0.2.0/25` and `192.0.2.128/25` networks. The two network have the same backup RLOC (i.e., 3.0.0.3) but they do not use the same primary RLOC which is `1.0.0.1` for `192.0.2.0/25` and `2.0.0.2` for `192.0.2.128/25`.

It is worth to notice that the choice of the source locator is coupled with the choice of the destination locator because some destination locators might be not routable on some links or the address families might be incompatible. Unfortunately, this problem is still open.

3.2.2 Incoming traffic engineering

Sec. 2.3 describes how an ITR selects the destination RLOC. In this section, we show that the construction of the mappings combined with the RLOC selection offers interesting perspectives for incoming traffic engineering. Throughout this section, we present the traffic engineering functionality with examples. This section considers manual mapping creation. The dynamic construction of mappings is presented in Chapter 7.

The simplest incoming traffic engineering requirement happens when a site has two ETRs including one for backup. In this situation, two priority values can be used. The lowest value, e.g., 10, being assigned to the *preferred* locator and the highest value, e.g., 100, to the *backup* locator. In this case, ITRs always use the primary locator, unless it is unreachable. Thus, the ITRs automatically use the backup RLOC as long as the primary locator is not available.

A second traffic engineering case happens when the site has two ETRs that can be used simultaneously. In such a case, the RLOCs have the same priority which enables the possibility to share the load among them. To control the proportion of flows to receive on each locator, the weight is used. ITRs load balance the flows among the locators, proportionally to their weight. If a locator has a weight of 75 and the other locator has a weight of 25, 75% of the flows will be encapsulated and forwarded to the first locator. The remaining 25% of flows will be forwarded to the other locator. The LISP specification imposes that the sum of the weights of the locators of the same priority equals 100. However this constraint makes the management of the mappings harder. For example, if a mapping has three locators a, b and c with the weights $5, 10$ and 85 respectively. If locator a is removed, the weights for b and c must be recomputed for their sum to equal 100. Without this constraint, when a is removed, the weights do not need to be recomputed but the proportion of traffic balanced to the two remaining locator is respected.

Fig. 2.5 shows an example of mapping that combines the two techniques presented above. The 2001:db8:beef::7 and 192.0.2.5 locators are primary locators and 192.0.2.254 is a backup locator. The 2001:db8:beef::7 locator has to be preferred for 75% of the encapsulated flows.

In the cases presented above, the policies are fixed in time and mapping changes are planned in advance, e.g., connectivity is evaluated every three months. However, the priority and weight are not always sufficient to control the costs. Indeed, to avoid performance degradation, load balancing is performed on a per flow basis [KKSB07] and independently of the total traffic generated by the flow. Therefore, it is possible to have a flow consuming a high bandwidth that goes through an ETR with a low weight. A solution to face this issue would to change the mappings when the load distribution among the ETRs does not follow the traffic engineering policies. For example, if a link is expected to cover 75% of the load but only represents 50% of the load, the mapping can be updated with a weight higher than 75 for the RLOC of this link to increase the probability that this link will be chosen. To do so, the mappings are time limited with a TTL. When the mapping expires, the ITR must refresh the mapping. By using the TTL, it is thus possible to tackle the per-flow load balancing issue and its possible load sharing violation. However, to avoid degrading the performance of the existing flows or moving important portions of traffic, care must be taken while changing the mappings for this reason.

Customer-provider links are often billed on a *95th percentile* basis [DHKS09]. With the 95th percentile billing, a month is divided in slots of, for instance, 5 minutes. During each slot, the amount of traffic transferred on the link is monitored and the bandwidth consumption is computed. At the end of the month, the 95^{th} percentile value of bandwidth is computed and the link is charged proportionally to this value as shown in Eq. 3.1

$$C_m = C_{fix} + percentile(95, \mathcal{B}_m) \cdot C_{var} \tag{3.1}$$

where C_m is the cost charged for the month m, C_{fix} a fixed cost, independent of the link usage, $\mathcal{B}_m$ the collection of the 5 minute bandwidths observed during the month m and C_{var} the cost charged for each Mbps. The 95^{th} percentile billing model has been proposed to reflect the bursty nature of IP traffic. Another model that is often used is the *flat fee* charging where the price of using a link is independent of its utilization. Flat fee can be seen as a 95^{th} charging model with C_{var} set to zero.

When a site is charged on a 95^{th} percentile basis, it is possible to reduce the cost of using the links when priority, weight and TTL are combined. Every few hours, a site can evaluate the expected cost of using each link and increase the weight of the links with a low expected cost. The principle is to limit the maximum bandwidth observed on the 5 minute slots such that the 95^{th} percentile remains as low as possible. If a link carried too much traffic, it is possible to stop receiving packets on it by setting it as backup with a low priority. To react to the link usage, the mapping TTL can be fixed to a value of few hours but care must be taken with low TTL values to avoid oscillations and traffic performance degradation. A month contains between 8,064 and 8,928 five minute slots meaning that each link has from 403 to 446 slots that will not be considered in the link fee. In other words, it is not required to be too reactive as the system can tolerate more than one day of arbitrary burst without impacting the link usage cost.

We have only considered primary and backup links so far. However, it is possible to have a hierarchy in RLOCs of the ETRs. For example, one site could have two links both running IPv4 and IPv6. The first link is the primary one and the second is for backup use. For internal reasons, IPv4 is preferred to IPv6 but the primary link is always preferred. In this situation, four priorities can be used, e.g., 10, 20, 100 and 110. The IPv4 locator on the primary link is attributed priority 10 and the IPv6 locator on this link priority 20. For the backup link, the IPv4 locator has priority 100 and the IPv6 locator priority 110. With these priorities, the site policies are respected. In any situation, the primary link is always preferred.

3.2.3 Mapping differentiation

It is generally considered that the same mapping is provided to all the ITRs. However, there exist situations where *differentiated mappings* can improve traffic engineering. A differentiated mapping is a mapping that is built specifically for a given ITR (or a set of ITRs). The first use of differentiated mappings is for topological separation. Depending on the origin of the Map-Request, the locators, priority and weight can be tuned (e.g., to be optimized for locality). This approach is very similar to how CDNs perform traffic engineering. In general, the CDNs control the traffic they have with their clients with DNS. Depending on the geographical or topological location of the clients, the DNS server provides different replies [PFA+10, HWLR08, CB08]. For example, a name requested by a client in Europe is likely to be resolved into addresses in Europe. CDNs can assume that the traffic is preceded by a DNS request but this assumption does not hold in general.

Differentiated mappings can also be considered if the site detects that an ITR does not respect the mapping priorities and weights. The problem with mappings is that the site explicitly provides different locators it is reachable from. It is then possible for an ITR to ignore the locator selection process and use the locator that best fits its local policy requirements. For example, an ITR can prefer to use a backup locator just because the backup locator is on a peering link while the primary locator must be forwarded via a high cost link. This situation perfectly illustrates the lack of incentives to respect the traffic engineering choices of the remote system. When a site detects that an ITR cheats, it can update the mapping provided to this ITR and remove the RLOCs for the links it does not want to be used by the ITR. The cost reduction has of course to be balanced with the resiliency reduction caused by the removal of some locators. If a LISP site wants to apply such strategy it must limit its mapping TTL to ensure that the source of the traffic will refresh the mappings frequently enough.

Mapping differentiation can be implemented either by the mapping system or independently. The cost of implementing the mapping differentiation directly in the mapping system depends on the mapping system. For example, NERD is not adapted for this purpose as one full mapping database would have to be stored on the server for each ITR. Mapping system with cache capabilities are also weakly designed to support mapping differentiation except LISP-Tree where caching is used for the intermediate nodes in the tree, not for storing the mappings. However, it is not possible to implement a pure differentiated mapping directly in LISP+ALT or LISP-Tree as they do not provide the mappings but the mechanism to find the mappings [FFML10b, JCAC+10]. However, LISP+ALT and LISP-Tree select the authoritative ETRs to send the Map-Requests to and it is possible to influence this decision to implement topological differentiation.

LISP+ALT relies on BGP to construct its topology meaning that deaggregation, AS path prepending or MED can be used to influence the path followed on the ALT. Similarly to DNS, the path in the tree in LISP-Tree depends on the server registration. For example, a LISP-Tree server in Europe can register itself only to parents in Europe while a LISP-Tree server in Australia will only register to parents in Oceania. When Map-Server are used in the mapping system, like in LISP-Tree or LISP+ALT, the ETRs can chose the MS they register their EID prefix on. As a result, a Map-Request is sent to an ETR or another, depending on the MS that is queried first. If the ETRs do not store the same mapping, this technique implements topological mapping differentiation.

In LISP-DHT [MI08], it is possible to implement differentiated mappings as the mapping system itself provides the mappings. The site servers only have to return a mapping that is a function of the requester.

It is also possible to implement differentiated mappings without the mapping system. A generic mapping is provided by the mapping system as if differentiated mapping was not used and when differentiation is required, the SMR bit is used. Using the SMR bit to implement differentiated mappings decouples the data plane, the mapping system and the incoming traffic engineering. With the SMR bit, the mapping system is used to provide the generic purpose stable mappings that respect general policies.

For complex network management, a system can be implemented to monitor the network and its traffic and enforce differentiated mappings for the critical ITRs. An interesting use case is the combination of the SMR bit with an *Anomaly Detection System (ADS)*. When the ADS detects abnormal traffic, an the SMR bit can be sent to the detected ITR to force it to change the mapping. The traffic can then be black-holed or isolated from normal traffic.

If mapping differentiation cannot be done directly by the mapping system or if the SMR bit is not applicable, the differentiation must be implemented on the ETRs. However, for site with complex mapping differentiation policies, the memory and CPU costs can be important. In such deployments, it is possible to deploy ETRs only involved in the control plane and ETRs only involved in the data plane. The ETRs involved in the data plane do never receive Map-Request because their locators are no registered to a Map-Server. On the contrary, ETRs involved in the control plane receive Map-Requests but never receive data plane packet. To do so, only the locators of the control plane ETRs are registered to the Map-Server. Likewise, the mappings only contain the locators of the data plane ETRs.

The ability to provide mapping differentiation is a major improvement in traffic engineering compared to BGP where such granularity is hard to obtain, if not impossible without extending it [WSR09].

3.3 Conclusion

One of the major drawbacks of BGP is the lack of control a site has on its incoming traffic. LISP has been proposed to alleviate some of the scalability issues of BGP. In this chapter, we show that an added value of LISP is its embedded ability to support incoming and outgoing traffic engineering.

In this chapter, we first show that static incoming and outgoing traffic engineering can be easily achieved with LISP. Then, we show that a LISP site can do incoming traffic engineering by potentially controlling differently the traffic from each different source.

We have shown in this chapter that the mappings in LISP enable flexible incoming traffic engineering. However, it is important to notice that the granularity of the control implemented in the network influences the monitoring and management overhead. Indeed, the more precise the decision, the more information must be tracked. It is potentially possible to provide one different mapping to every single ITR sending packets to the network in LISP. However, this granularity might generate an important management overhead. Therefore, the choice of the granularity is a tradeoff between the precision of the control and the management overhead for the network. To have a better traffic engineering support in LISP, we propose LISP-Tree, a new scalable mapping system, in Chapter 4.

In Chapter 5, we present IDIPS, a service that is able to rank the path between two IP addresses according to performance criteria. And in Chapter 7 we show how LISP and IDIPS can be combined to perform performance based incoming traffic engineering on the Internet.

Chapter 4

LISP-Tree

4.1 Introduction

Chapter 3 shows that LISP is designed with strong traffic engineering capabilities. Traffic engineering in LISP is performed by changing the priorities, weights and TTL of the mappings. Chapter 3 also shows that the mappings can be distributed in a per-requester basis with the mapping differentiation. Unfortunately, the mapping systems proposed so far for LISP ([D. 09, BCF$^+$08, MI08, Lea08]) are not well designed to support frequent changes in the mappings or an important number of mappings. For this reason, we propose in this chapter a new mapping system, LISP-TREE that is more adapted to a high load than the other mapping systems presented in Sec. 2.3.2. LISP-Tree is inspired by the widely used Domain Name System (DNS) [Moc87b], with a similar hierarchical organization. Blocks of EIDs are assigned to the layers of the EID hierarchy by following the current allocation rules for IP addresses. The root of the EID hierarchy is maintained by the Regional EID Registries, which allocate EID blocks to local registries. These in turn maintain delegation information for the owners of the provider independent EID prefixes. Levels can be dynamically added to the hierarchy. LISP-TREE nodes can use existing DNS implementations, and benefit from the long operational experience, but to avoid interference with the current domain name system, LISP-TREE should be deployed on a *physically* separate infrastructure. LISP-Tree can also be implemented with its own specific protocol. One of the main advantages of LISP-TREE is DNS' proven scalability track record: for instance, at the time of this writing, the *.com* top-level domain stores roughly 77 million entries [VB.09], while there are only around 350,000 entries in the default-free zone today [Hus].

In addition to its architectural qualities, the performance of mapping systems plays a key role in LISP. However, apart from a first trace-driven analysis of the mapping cache [IB07], no research effort has been devoted to this goal. The work on LISP-Tree aims to fill this gap as well, presenting a trace-driven simulation of the performance of LISP's main mapping systems. In particular we consider LISP+ALT, LISP-DHT and LISP-TREE. Our performance analysis assumes a complete deployment of LISP in today's Internet. We use topology and latency data from the iPlane [MAK+06] infrastructure and use the open source CoreSim simulator [CJC+09] developed at UPC on top of that. This setup provides estimations of LISP performance metrics.

LISP-Tree is the result of a collaboration with Loránd Jakab, Albert Cabellos-Aparicio and Florin Coras from Universitat Politècnica de Catalunya (UPC). We proposed the main LISP-Tree design taking into account deployment considerations. On the other hand the team at UPC took the lead for the simulations with CoreSim [CJC+09]. Finally, the work presented in Sec. 4.5 is an original work.

The following of the chapter is structured as follows: Sec. 4.2 describes LISP-TREE, our proposed mapping system. Next, Sec. 4.3 presents CoreSim, the simulator used to evaluate the performance of the main LISP mapping systems. The simulation results are presented and discussed in Sec. 4.4. Sec. 4.5 proposes to extend the LISP control plane for it to support LISP-Tree. Finally Sec. 4.6 includes the related work while Sec. 4.7 concludes the chapter.

This chapter provide two major original contribution. First, we propose a new scalable mapping system similar to DNS. Second, we provide the protocol that can be used to implement it.

4.2 LISP-TREE

This section presents the LISP-TREE mapping system. First, we provide arguments in favor of a DNS like mapping system, followed by a short overview of our proposal. Then, after the detailed description we propose a deployment scheme for today's Internet.

4.2.1 Why a DNS like mapping system?

On today's Internet, DNS is the system which is closest to a mapping system. More than twenty years of operational experience in implementing, testing, deploying, and operating such a system forms a solid foundation for an identifier-to-locator mapping service.

Scalability

Scalability is a key concern for any mapping system. The scalability of the current DNS system is due to several factors. First, caching is heavily used to reduce the number of queries that are sent by resolvers. Second, the domain names have a hierarchical structure, so the changes to the IP addresses and names of name servers responsible for domains are local to a few DNS servers and do not need to be propagated throughout the DNS. In contrast, the BGP-based LISP+ALT would require an intelligent organization of the topology, coupled with aggregation, to mitigate the risk of organic growth (i.e., due to non-technical reasons: social, economical) of the ALT routing table. Last, the combination of the previous two factors leads to a shortened path for a significant amount of queries, due to the fact that not only responses, but also intermediate nodes can be cached. Because queries can go directly from the resolver to servers lower on the hierarchy, not all cache misses have to traverse the tree.

Fault tolerance and troubleshooting

Network operators need to rapidly detect and fix the problems when they happen. DNS resolvers operate in iterative and recursive modes. In iterative mode, the resolver sends the queries to servers on different levels of the hierarchy, step by step, starting at the root. If one server does not reply, the resolver automatically tries another server [DOK92]. In recursive mode, the query is forwarded from one server to another, closer to the authoritative, which sends the reply back on the same path.

In this case, the location of a failing intermediate server will not be known to the resolver, which will be unable to circumvent the failing node if no extra failure discovery protocol is deployed. DNS allows both forms of operation, but today's Internet mostly uses iterative DNS [JSBM02].

Moreover, fault tolerance is a key concern in a mapping system. Because a reply from the mapping system is necessary to allow an ITR to send packets towards a destination EID, any failure of the mapping system would block communications. In the current DNS, fault tolerance is provided mainly by replicating servers. All the important zones of the DNS hierarchy are served by two or more name servers with distinct IP addresses. Furthermore, some of these IP addresses are in fact anycast addresses that correspond to several physical servers. DNS resolvers take advantage of this redundancy by using load balancing when contacting name servers [DOK92].

Security

None of the proposed LISP mapping systems have specified security features, except NERD [Lea08]. While some existing solutions could be used to secure them (e.g., applying the Secure Inter-Domain Routing architecture to LISP+ALT [BFMR10]), they have not received enough wide-scale implementation, testing and operational experience. DNSSEC adds the required security mechanisms to allow resolvers to authenticate the replies received from DNS servers. It is readily available in most DNS implementations, with some top-level domains already deploying it. It does introduce additional complexity to the mapping system, but all add-on security mechanisms do so.

For completeness sake, security was included in this section, but a detailed security analysis is left for future work.

Impact of configuration errors

DNS configuration errors impact only the domains served by the misconfigured name server and do not propagate to the whole system. With LISP+ALT, the main source of misconfiguration would be the advertisement of incorrect EID prefixes via BGP on the overlay network. Experience with BGP shows that many misconfiguration errors could cause network operators to advertise an invalid BGP prefix with different impacts on the network [MWA02]. All these misconfiguration errors could affect the ALT overlay as well.

4.2.2 LISP-TREE Overview

LISP-TREE is a hierarchical mapping system that has a clear separation of the mappings storage and their discovery. The *mapping storage* is under the responsibility of the ETRs while the *discovery* mechanism is built on top of the LISP-Tree protocol. The role of the discovery mechanism is to provide a list of ETRs that respond authoritatively for the mappings associated to the queried EID.

Fig. 4.1 presents an overview of LISP-TREE. When a requester needs to obtain a mapping for an EID, it first sends a request to the discovery part that answers with a list containing the locators of the authoritative ETRs for the requested EID. The requester then sends a Map-Request to one of these ETRs and receives a Map-Reply containing a mapping for the identifier. The mappings are provided by the ETR to let them control their traffic by setting the priorities and weights.

4.2.3 LISP-TREE Model

The binding between the EIDs and the locators is kept on the authoritative egress tunnel routers of customer domains. These ETRs manage and distribute these mappings with the aim of sinking all self owned EID-prefix mapping requests. All the mapping databases are combined to form the mapping storage. It will not be further detailed here as its functionality has already been defined in the LISP specification [FFML10a] and explained in Sec. 2.3. We assume that the mappings are more dynamic than the list of authoritative ETR providing these mappings.

ETRs respond with Map-Replies only for the EID space they are authoritative for [FFML10a]. Because of this, it is the responsibility of the inquiring node, the Map-Request originator, to find the locators of the ETRs

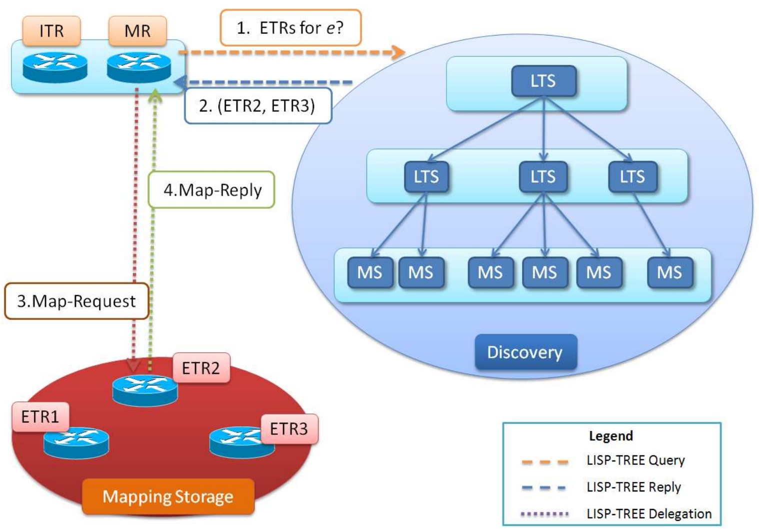


Figure 4.1: Global overview of LISP-TREE. 1. The requester asks the discovery part to have the locator of some authoritative ETRs for an EID e. 2. The discovery part provides this list. 3. The Map-Resolver sends a Map-Request to one of these ETRs. 4. The ETR sends a Map-Reply back with a mapping for e.

authoritative for the queried identifier. Such functionality is provided by the *discovery part* of LISP-TREE. It builds its tree by logically linking *LISP-TREE Servers (LTS)*.

All the LTSes are *responsible* for an EID prefix and build a hierarchy determined by their intrinsic relationship. Therefore, an LTS responsible for EID prefix p_1 is ancestor of an LTS responsible for EID prefix p_2 if and only if $p_1 \preceq p_2$[1]. Moreover, any LTS of prefix p_1 is a parent of an LTS responsible for prefix p_2 if and only if there exists no LTS of prefix p_3 such that $p_1 \prec p_3 \prec p_2$. All these strict ordering relations are stored by parent LTS as a list of child servers. Exceptions are the lowest level servers, the leaves, which store a list of ETRs that are responsible for their associated prefix. Hence, the search for any EID e ends at a leaf of prefix p that respects $p \preceq e$.

The authoritative ETRs are registered to their responsible leaf LTSes by using the Map-Register procedure defined in [FF09]. In LISP terminology, the leaf LTSes are called *Map-Servers (MS)* [FF09]. However, the Map-Server considered in LISP-Tree differ from those proposed in [FF09]. A LISP Map-Server is supposed to received encapsulated Map-Requests. When it receives such a Map-Request, it decapsulates it and send the inner Map-

[1]$p \preceq q$ if and only if prefix p is shorter (less specific) than q

Request to a registered ETR for the EID. In LISP-Tree, a Map-Server does not forward the Map-Request, instead it returns the list of authoritative ETRs to the requester.

To make LISP-TREE transparent for the ITRs, *Map-Resolvers (MR)* [FF09] are added to the system. When a Map-Resolver receives a Map-Request from an ITR, it queries the LISP-TREE discovery part to obtain the list of the authoritative ETRs for the EID in the request. Once the MR has the corresponding list of authoritative ETRs, it sends a Map-Request to one of them and subsequently forwards the received Map-Reply to the requesting ITR. Thanks to this functionality, the ITRs do not need to know anything about LISP-TREE, they just need to send a Map-Request to an MR.

To avoid circular dependencies in the addresses, every node involved in the mapping system (i.e., MRs, LTSes, MSes and xTRs) is addressed with a locator.

4.2.4 LISP-TREE Modes

Like DNS, LISP-TREE can run in two distinct modes: (*i*) *recursive* and (*ii*) *iterative*. Fig. 4.2 illustrates the difference between the two and presents a requester who wants to obtain a mapping for the EID `192.0.2.20`. To simplify the figure, only the last EID octet's is shown.

Recursive Mode

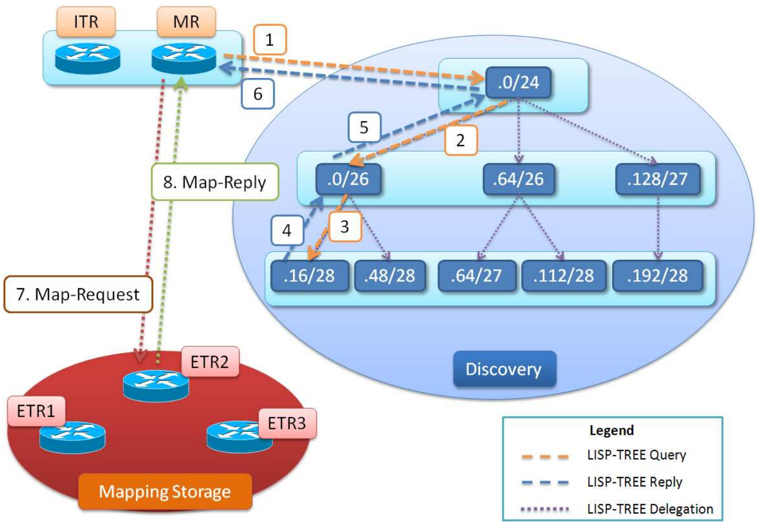
In the recursive mode (Fig.4.2(a)), the MR requests a mapping for EID e from a root LTS[2]. The root LTS sends a request to one of its children responsible for e and so on until a MS is reached. The MS then generates a list of e's authoritative ETRs locators and this list is back-walked until the answer arrives the root LTS. At that stage, the root LTS sends the reply back to the MR.

Iterative Mode

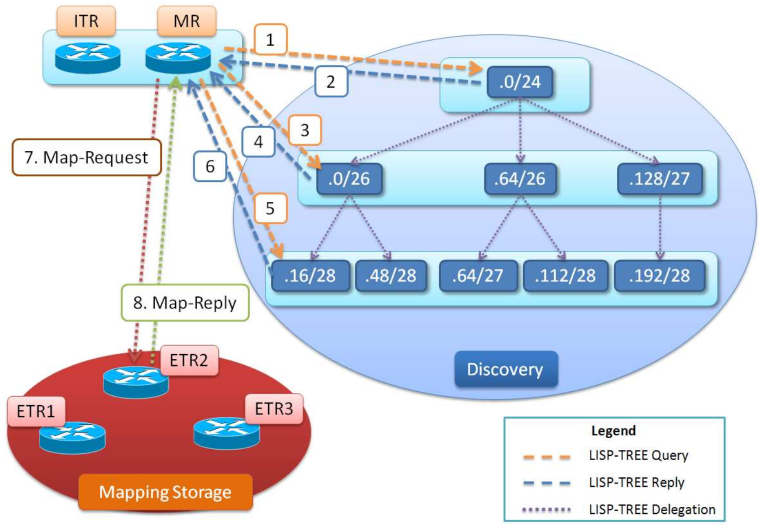
In the iterative mode (Fig.4.2(b)), the MR requests a mapping for EID e from a root LTS. The LTS then sends back the locators of its children responsible for e to the MR. The MR then sends a request for the mapping to one of those children. The child does the same and answers with a list of locators of its children responsible for e, and so on until a MS is reached. The MS then generates a list of e's authoritative ETRs locators and sends it to the MR.

In both modes, when the MR receives the ETR locators list, it sends a Map-Request to one of them to get a mapping for e and eventually receives a Map-Reply. It is possible to optimize for latency in the last step, by allowing

[2]A root LTS is an LTS that serves the root of the tree



(a) Recursive mode: queries in the discovery parts are progressively transmitted to a MS. The MS answer then back-walk the tree up to the root that sends the answer to the MR.



(b) Iterative mode: queries are moving back and forth from the MR and the LTSes, starting at a root LTS until a MR is reached. The MS then provides the answer to the MR.

Figure 4.2: LISP-TREE works with different mode: (a) recursive and (b) iterative.

the MS to send a Map-Request to a registered authoritative ETR but we do not consider this scenario in order to keep the architectural separation and with that the good troubleshooting properties of LISP-TREE. This separation also optimize the latency for retrieving mobile node mappings.

4.2.5 LISP-TREE Deployment Scenario

A key factor for the long-term success of a mapping system is its ability to scale. However, the effective deployment of a mapping system also largely depends on its ability to fit with a business model accepted by the community willing to use it.

We therefore propose a model that relies on the the *delegation* principle which is very common in the Internet. A globally approved entity (e.g., IANA, IETF. . .) defines the minimum set of rules and delegates the operations to independent service providers that can, at their turn, delegate to other providers and fix extended rules. People wishing to gain access to the system are free to contact any provider and sign agreements with them. The providers are differentiated by the extra services they propose.

The EID prefix delegation follows an approach similar to the current Internet prefix allocation. The IANA divides the EID space in large blocks and assigns them to five different *Regional EID Registries (RER)* that could be the current RIRs. The RERs are responsible for the allocation of prefixes to the *Local EID Registries (LERs)* and large networks. The LERs can then provide EID prefixes to some customers or LERs below, which, at their turn, can delegate the prefixes to other LERs or to customers. In this scenario, we consider that the EID space is independent from the currently assigned IP space. For example, it could be a subset of the IPv6 address space. This separation means that the EID space does not have to support all the legacy decomposition of prefixes observed on the Internet.

In this scenario, the tree has at least three levels. The first level (i.e., the root) is composed of LTSes maintained by the RERs. These servers replicate the same information: the list of all LERs responsible for all the large blocks of EIDs. The number of such blocks is limited (maximum 255 in an /8 decomposition perspective) meaning that the total state to maintain at the root LTSes is limited even if every prefix is maintained by tens of different servers. Level 2 of the tree is composed of the LTSes maintained by the LERs and the lowest level encompasses the map servers responsible for customer ETRs. State at level 2 depends on the deaggregation of the large block of EIDs and implicitly on the number of subscribed customer map servers. To avoid having to support a large number of deaggregated prefixes, an LTS can partially deaggregate its EID block (e.g., divide it by two) and delegate such sub-blocks. In other words, the state to maintain at an LTS can be controlled by adapting the depth of the tree.

It is important to note that the EID prefixes are independent of any given ISP, and that additional levels in the hierarchy can be defined if needed. It is worth to note that this raises the problem of registrar lock-in: once an organization gets an allocation, it cannot move the allocated prefix to a different registrar. It is also important to remark that any LTS (including the MSes) can be replicated and maintained by independent server operators and implementations. Therefore, to limit the impact of the lock-in, we would recommend the registrars to ensure enough diversity in LTS operators for their blocks. No significant lock-in is observed in DNS today thanks to this way of deploying TLDs. The load can also be controlled among the different replicas by deploying them in anycast. Finally, the use of caches, like in the DNS, can reduce the number of LTSes involved in every query.

4.3 Simulation Model

LISP is provided on some Cisco platforms and two experimental implementations are provided for the research comunity [ISB11, SN09] (see Sec. A). However, while these implementations help to validate and gain operational experience with the proposed protocols, they do not allow to estimate the behavior of LISP at a very large scale. Coras et al. developped *CoreSim* [CJC+09] for this purpose, an open source Internet-scale LISP deployment simulator[3]. CoreSim combines a packet trace and Internet latency data to simulate the behavior of an ITR and the mapping system.

CoreSim works on top of a topology built from measurements performed by the iPlane infrastructure [MIP+06], which provides Internet topology information and the latency between arbitrary IP addresses. The simulator reports mapping lookup latency, the load imposed on each node of the mapping system and cache performance statistics.

4.3.1 Topology Model

The first element to model the behavior of mapping systems in a large scale network is the network itself. For CoreSim, we chose to use the current Internet as the reference topology. The topology used in the simulator is composed of 14,340 points of presence (PoPs), one per autonomous system (AS), about which iPlane [MIP+06] knows inter-domain peering information. For the simulations we assume that each AS deploys one and only one LISP tunnel router (xTR) and that its xTR is located on its most connected PoP.

Once the xTRs are selected, they are assigned several EID prefixes. Those prefixes are the IP prefixes currently advertised in the default-free zone (DFZ) by this AS. Since the causes of prefixes deaggregation in BGP is not always clear, we removed from the iPlane dataset the more specific prefixes. These prefixes are mostly advertised for traffic engineering purposes [MZF07] and would not be advertised with LISP as it natively supports traffic engineering in the mappings. A total number of $112,233$ prefixes are assigned based on originating AS to their respective xTR[4].

[3]Available from `http://lisp.cba.upc.edu/`

[4]Data is from March 2009

CoreSim relies on the iPlane Internet latency lookup service that returns the measured latency between arbitrary IP addresses to simulate the time required to obtain a mapping in the simulated mapping system. Unfortunately, iPlane does not provide all possible delay pairs. Because of this, for 35% of the lookups, we use a latency estimator that correlates the geographical distance between IP addresses as reported by the MaxMind database[5] with the latencies based on an iPlane training dataset (see details in [CASBDP09]). This approach was found only slightly less accurate than other more complex algorithms in [AL09] using measurement data from over 3.5 million globally well distributed nodes.

An important assumption made by CoreSim is that there is no churn in the topology during a simulation run, meaning that the delay between two nodes is constant, and that nodes never fail.

4.3.2 ITR Model

CoreSim studies the behavior of only one ITR at a time, therefore, out of all xTRs, one is selected as the ITR under study. The PoP for the selected ITR is determined by manual configuration, based on the point of capture of the traffic trace that is fed to the simulator. The ITR caches mappings (i.e., resolved Map-Requests) and evicts entries after 3 minutes of inactivity.

4.3.3 Mapping System Models

In CoreSim, a LISP Mapping System is modeled as an IP overlay. The overlay is mainly composed of nodes of the topology module, but also includes some mapping system specific ones. As expected the organization of the overlay depends on the considered mapping system.

The simulator routes Map-Requests from the ITR to the node authoritative for the mapping (ETR). The path of the query and the latency of each hop are recorded in order to infer statistics and metrics of interest.

LISP+ALT model

The LISP+ALT draft [D. 09] envisages a hierarchical topology built with GRE tunnels but does not recommend any organization for the overlay. Therefore, among all the possible organizations, a hierarchical overlay structure with strong EID prefix aggregation for advertisements has been chosen.

Based on discussions on the LISP mailing list, a three-level hierarchy was decided (see Figure 4.3). In this hierarchy, the bottom leaf nodes are ALT routers belonging to a certain domain. The upper two levels are dedicated ALT routers, which may be offered as a commercial service by providers or registries. Each of these nodes is responsible for certain aggregated prefixes,

[5]Available from `http://www.maxmind.com/`.

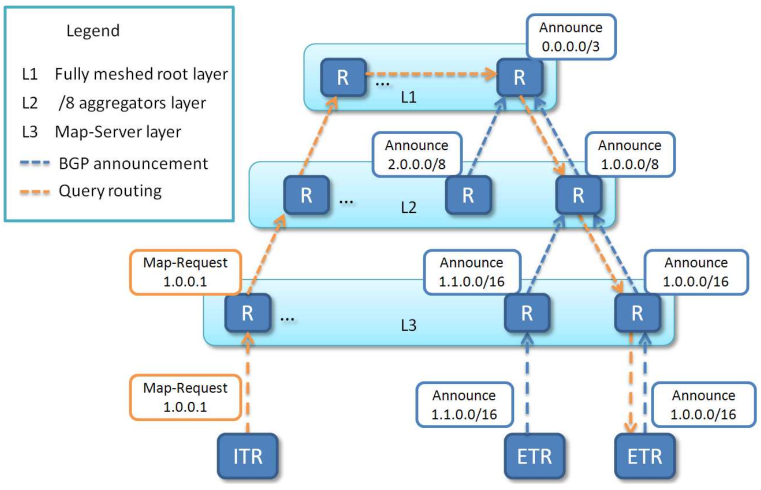


Figure 4.3: LISP+ALT architecture

and connects to all lower level nodes which advertise sub-prefixes included in those prefixes. The hierarchy consists of 16 root (first) level ALT routers, responsible for the eight `/3` prefixes, and 256 second level ALT routers, each responsible for a `/8` prefix. For each `/3` prefix two ALT routers at different locations are used and each lower level ALT router connects to the one with the lowest latency. All these 16 routers are connected to each other with a fully meshed topology. Please note that LISP+ALT can support other topologies and for instance include intra-level links.

Map-Requests are routed via the shortest path through the hierarchy starting from the bottom layer where the ITR is connected to the ALT topology.

LISP-DHT model

The simulator considers the $112,233$ filtered prefixes from iPlane and builds a static Chord ring of the same size, using a trace-based approach to compute the finger table on each node. The last EID of each prefix is used as the ChordID of the LISP-DHT nodes that is responsible for this EID prefix. EID lookups proceed according to the standard Chord protocol, which we implemented in CoreSim.

LISP-TREE model

CoreSim considers only one ITR at a time, and because the MR always selects the closest root LTS, the one selected by the MR is always the same. Therefore, the simulator considers only one root LTS for the tree. This server connects to the 256 level 2 LTSes that are each responsible for one `/8` EID prefix. In turn, these servers know about all the third level LTSes that are responsible for the prefixes included in their /8 prefix. These third level servers are the map servers the ETRs subscribe to. The simulator assumes that an ETR registers to a single MS, and that MS is located in the same PoP as the ETR. Since the simulator assigns them to the same PoP, the resulting latency between them is 0. This deployment scenario is an instantiation of the one presented in Sec. 4.2.5 which is congruent with the current Internet.

Additional implementation details on the simulator can be found in [Cor09].

4.4 Mapping System Comparison

Section 4.2 described in detail the advantages of our proposed mapping system from an architectural point of view. This section complements that with a qualitative analysis, comparing several performance metrics of three mapping systems: lookup latency, hop count, node load and the amount of state stored in mapping system nodes. Low lookup latency improves user experience for new flows, while node load and state affect the scalability of the system. We begin by describing the packet traces that support our evaluation.

4.4.1 Experimental Datasets

In order to evaluate the performance of the mapping systems presented above we used traffic traces collected at the border routers of two university campuses, because border routers are the most likely place to deploy a LISP tunnel router. The first trace was captured at Université catholique de Louvain (UCL) in NetFlow format, and the second is a packet trace from Universitat Politècnica de Catalunya (UPC).

UCL

The UCL campus is connected to the Internet with a 1 Gbps link via the Belgian national research network (Belnet). This trace consists of a one day full NetFlow trace collected on March 23, 2009. For this chapter, only the outgoing traffic is considered, representing 752 GB of traffic and 1,200 million packets for an average bandwidth of 69 Mbps. A total number of 4.3 million different IP addresses in 58,204 different BGP prefixes have been contacted by 8,769 different UCL hosts during the 24 hours of the trace. The UCL campus is accessible to more than 26,000 users.

NetFlow generates transport layer flow traces, where each entry is defined as a five-tuple consisting of the source and destination addresses and ports, and the transport layer protocol. The simulator however requires packet traces. This is why the NetFlow trace collected at UCL has been converted into a packet trace: for each flow, we generated the number of packets specified in the NetFlow record, distributed evenly across the flow duration and the size of the flow. Throughout the rest of the chapter, the term *UCL trace* corresponds to the packet trace obtained from the NetFlow trace collected at UCL.

UPC

The second unidirectional trace we used was captured at the 2 Gbps link connecting several campus networks of UPC to the Catalan Research Network (CESCA) with the help of the CoMo infrastructure [BRISC$^+$07]. It consists of the egress traffic on May 26, 2009 between 08:00-11:49 local time, and contains about 1,200 million packets accounting for 463 GB of traffic with an average bandwidth of 289 Mbps. 4.3 million distinct destination IP addresses from 56,387 BGP prefixes were observed in the trace. UPC Campus has more than 36,000 users.

4.4.2 Cache Miss Rate

The average packet rates at the UCL and UPC border routers are 13 Kpkts/s and 87 Kpkts/s, respectively. However, due to the nature of the traffic, a mapping is already cached by the ITR for most of the packets, with only 0.31% and 0.1% of cache misses for the mentioned vantage points. A cache miss occurs when no mapping is known for an EID to encapsulate. On cache miss, a Map-Request is sent to obtain a mapping, resulting in 2,350,000 Map-Requests sent for UCL and 560,000 for UPC during the 24h and 4h periods of the traces, which are shorter than the default TTL value (1 day) for mappings. As a result, all cache evictions were due to lack of activity to a particular prefix for 3 minutes, rather than expired TTL. These values for Map-Requests have to be compared with the total of 1,200 million packets observed for both traces. The difference between UCL and UPC can be explained by the higher average packet rate of the UPC trace, which keeps the cache more active, resulting in less timed out entries.

During our simulations the maximum number of mapping cache entries reached was 22,993 and 15,011 for the two traces. This is an order of magnitude less than routes in the DFZ. For a detailed mapping cache study, please refer to [IB07].

It is important to note that the results are obtained for a cache initially empty. Therefore, the miss rate is very important at the beginning of the experiment and thus influences the number of requests. The number of Map-Requests would have been dramatically smaller if we had considered the system at the steady state.

4.4.3 Mapping Lookup Latency

The mapping lookup latency is particularly important in mapping systems as it represents the time required to receive a Map-Reply after sending a Map-Request. When an ITR waits for a mapping for an EID, no packet can be sent to this EID. If this delay is too long, the traffic can be severely affected.

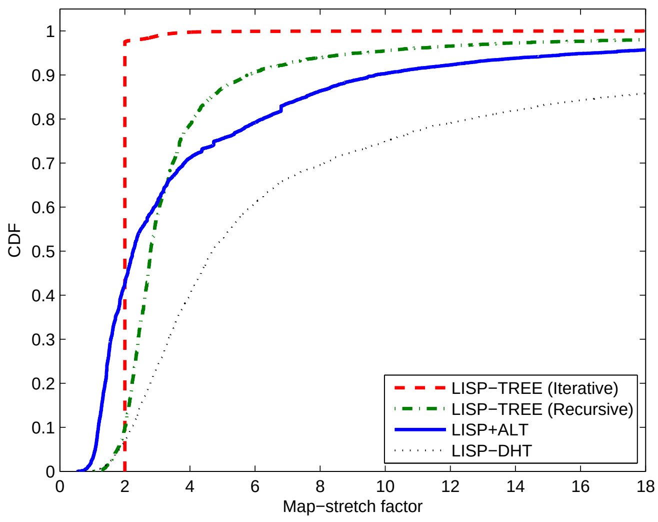
For instance, one of the Internet's most popular resources, the World Wide Web is continuously evolving, delivering web pages that are increasingly interactive. Content on these pages is often from different servers, or even different providers, and presents a multi-level dependency graph [Wis07]. This is already a challenging environment for some applications, and the introduction of a new level of indirection has the potential to introduce additional latencies. However the ATLAS study [LIJM^{+}09] shows that content is more and more comming from datacenters and CDNs.

To compare the considered mapping systems, we define the *map-stretch factor* of a lookup as the ratio between the total time required for performing it and the round-trip delay between the ITR and ETR. A low map-stretch indicates that the mapping system introduces a low delay overhead.

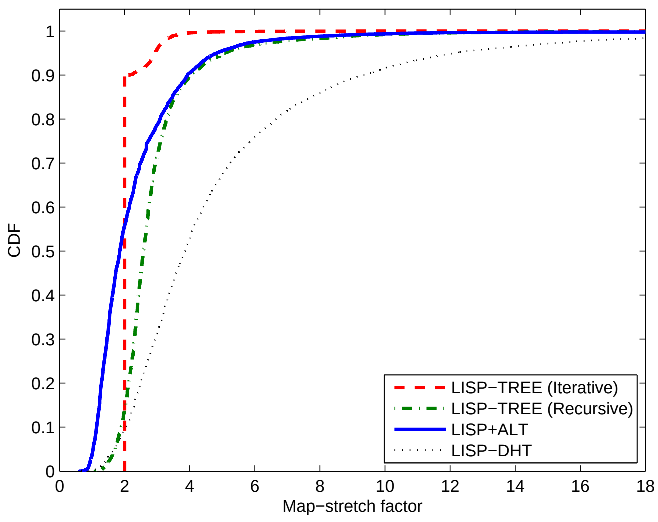
Fig. 4.4 presents the cumulative distribution function of the map-stretch factor obtained for both the UCL and UPC traces. In most of the cases (over 90%) the iterative LISP-TREE presents a stretch factor of 2. After the initial cache warm up, there is no need to contact the root and level 2 servers, their responses are already cached. Still, the discovery phase cannot be completely avoided and the level 3 MS have to be contacted for the RLOCs of the ETRs. These are not cached in the MR, because it would have a negative impact on the mapping dynamics, limiting the possibilities for supporting fast end-host mobility.

Recursive LISP-TREE is slower than iterative in over 90% of the cases. The caching limitation mentioned in the previous paragraph has particularly negative consequences for this operating mode: since the only response arriving to the MR is the list of RLOCs for the authoritative ETRs, no caching at all can be done. The 10% of the cases when recursive is better than iterative can be attributed to the path via the tree being faster than from the MR to the MS, resulting in map-stretch ratio values below 2.

Concerning LISP+ALT, its latency performance is similar to LISP-TREE iterative. This is because in LISP+ALT, the queries are routed through to the overlay topology, which is composed by core routers. According to our assumptions (see Section 4.3), these nodes are located in well connected PoPs. However, in iterative LISP-TREE the query flows in almost all the cases between the ITR and ETR, which may not be so well-connected. When comparing LISP+ALT and LISP-TREE recursive, we can see that LISP+ALT performs slightly better. In the recursive mode of LISP-TREE queries are also forwarded through a topology of well-connected nodes, but



(a) UCL



(b) UPC

Figure 4.4: CDF of map-stretch ratio. Use of caching makes LISP-TREE have a constant value for the majority of the lookups.

as we will see, they have to follow a much longer path.

As expected, LISP-DHT has the highest latency because of the longer path taken by the queries routed through the P2P topology. This is a well known problem in P2P networks, and research to tackle this issue is ongoing [XYK$^+$08]. However, it is worth to note that LISP-DHT uses a particular

Chord ring, where the peers do not choose their identifiers randomly, but as the highest EID in the EID prefix. This enforces mapping locality that ensures that a mapping is always stored on a node chosen by the owner of the EID prefix. As a consequence, it may be difficult for LISP-DHT to benefit from existing optimization techniques proposed in the literature.

Average lookup latencies were close to half a second for all mapping systems except LISP-DHT, which had values slightly larger than one second.

Fig. 4.5 helps to better understand the mapping latency difference between the mapping systems. It presents a CDF of the number of hops passed by over which a request and reply.

The iterative version of LISP-TREE has the lowest hopcount values, which can be explained, just like for the latency, by caching in the Map-Resolver. Recursive LISP-TREE not helped by caching, and queries have to traverse the full path in each case, for a maximum of 8 hops (Fig. 4.2(a)).

The topology chosen for LISP+ALT in the simulator (Fig. 4.3) limits the maximum number of hops to 6, but in 95% of the cases, this maximum number of hops is observed. In order to have a shorter path, the destination EID would have to be in one of the `/8` prefixes that doesn't have a more specific part announced separately. As we will see in the next section, this also increases the load on the nodes composing the LISP+ALT mapping system. In fact, Fig. 4.3 shows that all queries are forwarded through the root layer. This may result in scalability problems.

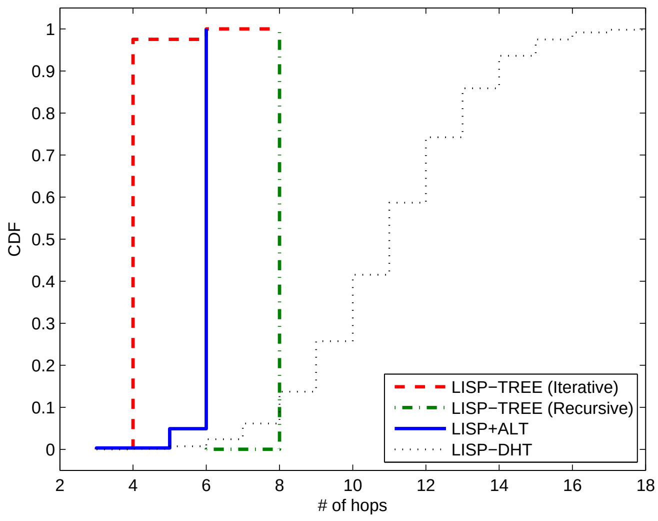
LISP-DHT has a much higher hop count with a maximum of 17 hops. This not only increases the lookup latency, it means that more nodes are involved in the resolution of a mapping than in the case of the other mapping systems, increasing the overall load on the system and the probability of failure.

Summarizing, these results reveal significant differences among the mapping systems under consideration. LISP-TREE and LISP+ALT both use a hierarchical model, where nodes on the query path tend to be congruent with the topology. In contrast, the Chord overlay used to route queries in LISP-DHT does not follow the underlying physical topology. Further, iterative LISP-TREE allows caching and this reduces its mapping latency.

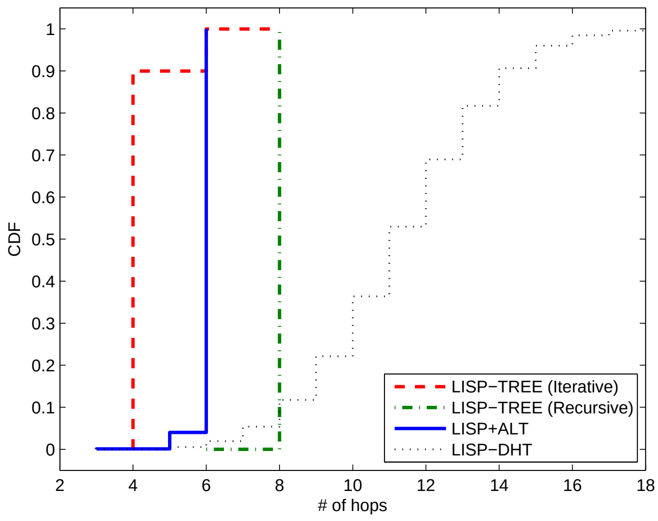
The latency introduced by the mapping system in case of cache misses will likely have a negative impact on the higher layer protocols, in particular on congestion control in the transport or application layer. These issues deserve a dedicated study, and are left for future work.

4.4.4 Node Load

We define the *node load* as the total number of Map-Request messages processed by nodes of the mapping system during the full run of the trace. For a more thorough analysis, we differentiate between the load caused by messages forwarded by the node (*transit load*) and the load due to the



(a) UCL



(b) UPC

Figure 4.5: CDF of hop count. For hierarchical mapping systems it is almost constant, for LISP-DHT we have a wide array of different values.

messages for which the node is the final destination (*mapping load*). Mapping load mainly depends on the observed traffic (distribution of destination EIDs) and is mostly the same for all studied mapping systems. On the other hand, transit load is mapping system specific and depends on how a request is sent to the holder of the mapping. Table 4.1 summarizes the load statistics

Table 4.1: Node load in levels 1 and 2

	Level 1		**Level 2**	
Mapping System	**Avg.**	**Max.**	**Avg.**	**Max.**
LISP-TREE (Itr.)	158	158	372	2,354
LISP-TREE (Rec.)	2,351,815	2,351,815	14,941	70,520
LISP+ALT	655,600	2,348,695	29,776	2,356,404
LISP-DHT	147,860	1,257,642	258	2,365,797

of the UCL trace.

The root LTS is only contacted 158 times in the case of iterative LISP-TREE, that is, the number of times necessary to look up the locators of the level 2 nodes responsible for the contacted /8 prefixes. Since these locators are cached by the Map-Resolver, they are only requested once. The load on the level 2 servers is also very low, compared to the other mapping systems, where there is no caching on intermediate nodes. In recursive LISP-TREE all Map-Requests have to pass through the root node, and then get distributed to the lower level nodes, according to the destination prefix. In LISP+ALT, level 1 consists of 7 logical nodes for the seven /3 covering the routable unicast prefixes. The nodes responsible for IP space with high prefix density sustain a considerably higher load.

For LISP-DHT, level 1 in the table refers to the transit load of the ITR's fingers, while level 2 to the transit load of all other nodes. Figure 4.6 shows the transit load in LISP-DHT. Vertical lines represent fingers of the ITR at UCL, which was originating the Map-Requests. From the 112, 233 nodes participating in the DHT, only 74% have routed or received Map-Request messages and are depicted in the figure. We can observe a sharp surge after a load value of 1000, that accounts for about 1.8% of the total number of nodes on the DHT. As expected we find many of the ITR's fingers among these hotspots. Of the 2, 365, 797 Map-Requests initiated, more than half pass via the last finger and one third via the second last finger. Among the top ten most loaded nodes 7 are not fingers.

Upon further inspection it was discovered that the node with the highest load was the last finger of the ITR. Due to the way Chord lookups work, this node is responsible for half of the key space on the Chord ring, which explains the high load. Further investigation revealed that there is one node which is last finger for 5.6% of the LISP-DHT participants: the one responsible for the prefix 4.0.0.0/8. This is the first prefix from the IPv4 space present in the iPlane dataset. Since Chord is organized as a ring (key space wraps around), this node becomes responsible for the EID space of classes D and E as well. Because this EID space is not represented in the overlay, the result is a disproportional load. The UPC trace produced similar load distribution results.

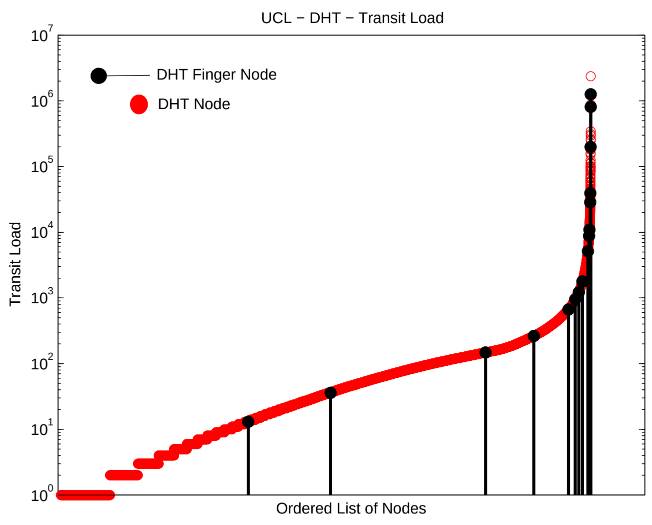


Figure 4.6: LISP-DHT load distribution. Fingers of the ITR, represented with the vertical lines, cluster on the right, having a considerable load.

LISP-DHT's transit traffic distribution characteristics may be desirable for peer-to-peer networks, but are a major disadvantage for a mapping system. Since the transit route is defined only by the Chord routing algorithm, two issues arise: lack of path control may lead to choke points at nodes belonging to small sites, and exposure of mapping traffic to unknown third parties (potentially leading eavesdropping and denial-of-service attacks). This makes LISP-DHT a poor choice as a mapping system.

Due to these architectural differences, the mapping systems under study exhibit different transit load characteristics. Indeed, in the case of LISP-DHT all participating nodes are both transit and destination nodes, while the hierarchical mapping systems have dedicated transit nodes on levels 1 and 2. The iterative version of LISP-TREE has a significantly lower load in those dedicated transit nodes, because of the caching done in the Map-Resolver.

LISP+ALT needs to route all packets through the root nodes, producing a potential hot spot. LISP-TREE on the other hand avoids using the root nodes most of the time, because it is able to cache intermediate nodes. Apart from the obvious scalability advantages, this improves reliability as well, since a total failure of the root infrastructure would still allow partial functioning of LISP-TREE, while no resolutions would be possible in LISP+ALT.

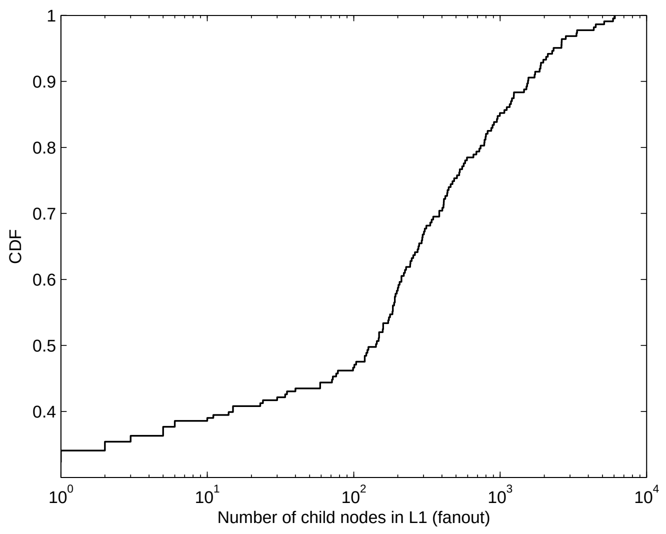


Figure 4.7: The cumulative distribution of the amount of state in the level 2 LISP-TREE and LISP+ALT nodes

4.4.5 Operational considerations

The mapping requests are transmitted through the mapping system towards the queried ETR. To achieve this, every node involved in the topology has to store some *state* about its neighbors. The amount of state needed on each node is directly related to the mapping system technology. For example, in an horizontally organized system such as LISP-DHT, all nodes have the same amount of state (32 entries). On the contrary, the amount of state needed on a node in a hierarchical mapping system depends on its position in the hierarchy.

LISP-TREE and LISP+ALT are both hierarchical topologies, and in this chapter have been deployed according to the same EID prefix distribution, and thus have the same amount of state. The root nodes refer to all the disjoint /8 prefixes, which amounts to a maximum of 256 entries when all are allocated. The number of nodes that are referred to by level 2 nodes depends on how the prefixes are allocated. Fig. 4.7 shows the distribution of the state kept at each level 2 node (both LISP-TREE and LISP+ALT). Finally, the leaves store a small amount of entries, equal to the number of announced prefixes.

Fig. 4.7 shows that 15% of the nodes have more than 1000 children and the most connected node has 6072 children. For LISP-TREE this is not an issue, as the nodes only need to keep a database that relates a prefix with the list of locators. It is well known that currently deployed DNS servers scale to much more than thousands of records [VB.09]. However, in the case of LISP+ALT, a BGP session has to be maintained for each child, as well as a tunnel between the two nodes. The costs in terms of configuration, memory, and processing are much higher than for LISP-TREE. A way to alleviate this would be to increase the depth of the tree and thus reduce the number of children of any one node. Unfortunately, this solution stretches the path length within the tree and is likely to increase the mapping lookup latency. Another solution could be to follow an organic deployment, but in that case, the mapping system would eventually have scalability issues because of the lack of aggregability.

4.5 Implementing LISP-Tree with LISP Iterable Mappings

Different technologies can be used to implement the LISP-Tree protocol. For instance, LISP-Tree is a DNS like system, so that, the DNS protocol could be used directly. However, there is a major difference between DNS and LISP-Tree: DNS is designed to work with *Fully Qualified Domain Names (FQDN)* while LISP-Tree is designed to work with EID prefixes. While the hierarchy is explicit in FQDN with the dot separator, it is implicit in EID prefixes. For instance, the FQDN `acme.example.com.` shows that `acme` is a child of the `example` domain that belongs the `com` top level domain. EID prefixes are fundamentally different as they are following a longest-match prefix pattern. Reverse DNS supports longest prefix matching but DNS is far from being optimized to work with IP prefixes [EdGV98, Moc89, Moc87a, THKS03]. A major concern with DNS is the way IP addresses are encoded and configured. We therefore propose not to use DNS directly to implement LISP-Tree but to extend the LISP control plane. However, LISP-TREE can be built over DNS [MD95] but then it must be deployed independently to keep the separation between the DNS names and the identifier resolutions.

The LISP control plane contains four types of message. The Map-Request, Map-Reply, Map-Register and the Encapsulated Control Message that we have discussed in Sec. 2.3. Unfortunately, these messages are not sufficient to implement LISP-Tree. LISP-Tree requires a message to provide a list of authoritative ETR that can be Map-Requested to retrieve a mapping but Map-Replies are used to provide the list of locators to use for encapsulating packets to an EID prefix. A mapping that contains a list of authoritative ETRs is called an *Iterable mapping*.

A first solution could be to use the Map-Reply as-is and interpret them differently depending on the context. When an LTS receives a Map-Reply, the list is considered as the list of authoritative ETR. When an ITR receives a Map-Reply, the Map-Reply corresponds to the requested mapping. This approach is simple and do no require any change in the protocol but this implicit information means that the system must maintain information about the context to remove any ambiguity. Managing this context may become complex if a node is at the same time a Map-Resolver, an ITR and an LTS[6]. Therefore, we propose to add a new LISP control plane message, the *LISP Iterable-Reply* to support Iterable mappings.

LISP Iterable-Reply

The LISP Iterable-Reply can be considered as a lightened Map-Reply that only contains a list of LTS or ETR RLOCs and the EIDs these entities are authoritative for. Fig. 4.5.1 shows the LISP Iterable-Reply message format. An Iterable-Reply message is composed of records. Each record corresponds to an EID prefix and its list of authoritative LTS/ETR locators. The records are time-limited by using a TTL. As for a Map-Reply, the nonce is echoed from the request that generated the Iterable-Reply. The Iterable-Reply message also contains the T-bit (terminal-bit). This flag is set if the progression in the hierarchy ends. When the T-bit is set, the locators in the Iterable-Reply are all referencing authoritative ETRs, not LTS. When a node operates in recursive mode, the Map-Reply that it receives must have the T-bit set.

By construction, the Iterable-Reply is a backward compatible extension to LISP as it is a new message type that only needs to be implemented by the LISP-Tree nodes.

4.5.1 Map-Request Extension for LISP-Tree

An Iterable-Reply is an answer to a request for an Iterable mapping. Two solutions are possible. On the one hand, a new control plane message can be introduced for this purpose. On the other hand, Map-Request can be used. Proposing a new message type is probably the cleanest solution from an architectural point of view. However, the semantic of this new message would not be significantly different than the semantic of a Map-Request. Nonetheless, the Map-Request message does not contain a field that could be used to inform the receiver that an Iterable-Reply is expected instead of a Map-Reply. Consequently, we propose to extend the Map-Request format with the addition of a flag in the reserved space. The I-bit – for Iterable-bit – flags indicates whether the requester is expecting an Iterable-Reply or a Map-Reply. If this bit is set, the requested node must return an Iterable-Reply. A second flag must be added to the Map-Request to indicate if the requester operates in recursive mode or not. This flag, the R-bit, for recursive bit, is added in the reserved space. This bit is set only if the requester is in the recursive mode. When this bit is set, the returned

[6]The context ID of a message can be its nonce but the nonce must remain random to avoid security problems

Iterable-Reply must have the T-bit set. If the T-bit is not set, the requester must fall back to the iterative mode or consider that the resolution is not possible. Fig. 4.5.1 and Fig. 4.5.1 show the message format.

```
        0                   1                   2                   3
        0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |Type=1 |A|M|P|S|I|     Reserved      |   IRC   | Record Count  |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                         Nonce . . .                           |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                         . . . Nonce                           |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |         Source-EID-AFI        |  Source EID Address  ...      |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |         ITR-RLOC-AFI 1        |    ITR-RLOC Address 1  ...    |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                              ...                              |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |         ITR-RLOC-AFI n        |    ITR-RLOC Address n  ...    |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     / |   Reserved    | EID mask-len  |        EID-prefix-AFI         |
   Rec +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     \ |                       EID-prefix  ...                         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                   Map-Reply Record  ...                       |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                     Mapping Protocol Data                     |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.8: Map-Request extension to support the Iterable Mapping extension

Inasmuch as these flags are in the current reserved block of bits, the legacy implementations always set them to 0, the extension is thus backward compatible.

One could argue that similar flags could be used in the Map-Reply to implement the iterable mappings. The idea is rational but suffers from two drawbacks. First, the records in Map-Replies contain many fields that are not relevant for the iterable mappings causing an unnecessary overhead. Second, the semantic of a Map-Reply differ from the semantic of an Iterable-Reply. While a Map-Reply carries mappings and their locators, the Iterable-Reply carries the authoritative ETR locators for a given EID prefix. We then conclude that separating these two roles is cleaner from an architectural standpoint.

```
      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |Type=5 |              Reserved                 | Record Count  |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                         Nonce . . .                           |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                         . . . Nonce                           |
+--> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    |                          Record  TTL                          |
|    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
R    | Locator Count | EID mask-len  |            EID-AFI            |
e    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
c    |                          EID-prefix                           |
o    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
r L--|          Unused Flags         |            Loc-AFI            |
d o  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| c--|                            Locator                            |
+--> +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4.9: Iterable reply message format

4.5.2 Functional Specifications

This section describes the actions performed on the entities involved in the LISP-Tree hierarchy. We assume that Map-Resolvers and Map-Servers are connected to the LISP-Tree mapping system. We also assume that ITRs are configured with a Map-Resolver and that ETRs are registering to Map-Servers.

When an ITR must retrieve a mapping for an EID, it sends a Map-Request to its Map-Resolver as specified in the LISP specification [FFML10a, FF09]. If the Map-Resolver is caching a mapping for the requested EID, it replies to the ITR with a Map-Reply containing the mapping for the requesting EID. If the Map-Resolver has no mapping for the requested EID in its cache, it sends an extended Map-Request to a root LTS. The Map-Request is always sent with the I-bit set. Once the Map-Request has been sent to the LTS, the Map-Resolver waits for an Iterable-Reply. If the received Iterable-Reply does not have the T-bit set, the Map-Resolver sends an extended Map-Request with the I-bit set to a locator found in the Iterable-Reply and waits for an Iterable-Reply. This procedure is repeated until the received Iterable-Mapping is flagged with the T-bit. Upon reception of a T-bit flagged Iterable-Reply, the Map-Resolver sends a Map-Request without the I-bit set to one of the locators in the Iterable-Reply. A Map-Reply is eventually received by the MR. Upon reception of the Map-Reply, the MR sends a Map-Reply to the ITR with the mapping contained in the received

Map-Reply. The Map-Resolver can cache the intermediate locators in the tree to speed up any further resolution. In this case, the Map-Resolver does not sends the first Map-Request to a LISP-Tree root but to a locator that belongs to the most specific EID prefix stored in the cache. This operation is similar to what is done in DNS. When a resolver receives an NS record, it caches it for future use. For instance, if a DNS resolver has resolved `example.net.`, and has to resolve `lisp4.net.`, it does not need to contact a root DNS server first to determine a `net.` name server. This speeds-up the resolution and reduces the overall load on the system. If the Map-Resolver has already resolved EID prefix 192.0.2.0/25 and if it exists an LTS authoritative for 192.0.2.0/24, when it will have to resolve 192.0.2.128/25, it can send the Map-Request to the LTS authoritative for 192.0.2.0/24 directly instead of requesting a LISP-Tree root server, as long as the LTS for 192.0.2.0/24 is in the Map-Resolver cache.

When a Map-Request with the I-bit set arrives at a Map-Server, it answers the request with the list of authoritative ETRs for the EID. The answer is provided by an Iterable-Reply whose T-bit is set to 1.

The functional specification given above is for Map-Resolver running in iterative mode. When a Map-Resolver is running in recursive mode, the first Map-Request sent by the Map-Resolver must be flagged with the R-bit. When an LTS receives a Map-Request with both the I and R bits set, it must generate a new Map-Request it is the originator for. The Map-Request is also flagged with the I and R bits. This procedure is repeated until the Map-Request arrives at a Map-Server. The Map-Server then sends back an Iterable-Reply with the T-bit. The mapping contained in this reply is then sent back to the LTS that made the request and so on until the Map-Resolver receives the Iterable-Reply. As long as the mapping backtracks the hierarchy of LTS, the T-bit is set. The nodes can cache the reply to avoid traversing the hierarchy. It is important to notice that all the nodes involved in the recursive resolution must maintain state for the request as explained in Sec. 4.2.4. If a node receives an extended Map-Request with the R-bit set but does not accept to run in recursive mode, it must return the same result as it would have returned in iterative mode. If an LTS receives an Iterable-Reply without the T-bit while running in recursive mode, the reply is back-tracked. If a Map-Resolver receives an Iterable-Reply without the T-bit while running in recursive mode, it has two options. Either it considers that the resolution is impossible and returns a negative Map-Reply to the ITR with the ACT field set to 3, i.e., drops all the packets for this mapping. The MR can also fall back into the iterative mode and sends an extended Map-Request to a locator present in the Iterable-Reply.

A LISP-Tree server can run in iterative or recursive mode. When running in iterative mode, the operation at the LTS only consists of a looking the local children database. As opposed to that, the operation of an LTS in recursive mode is more complex and involves per-request state. When a

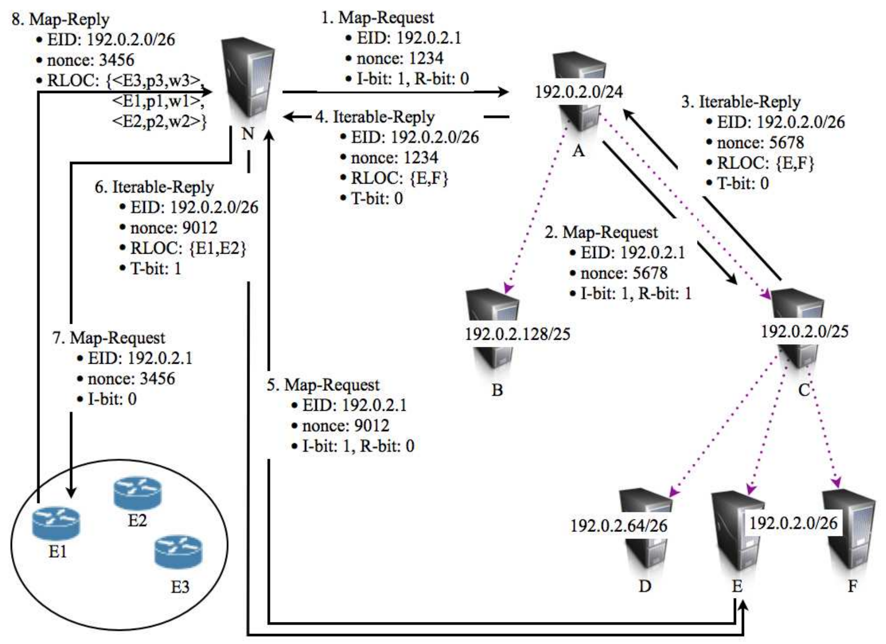


Figure 4.10: Example of Iterable mapping in iterative and recursive mode. `N`, `D`, `E` and `F` run in iterative mode while `B` and `C` are in recursive mode.

Map-Request arrives at the LTS, the EID is extracted from the Map-Request and a new Map-Request is created by the LTS with a new nonce. This new Map-Request is sent to a child LTS that is responsible for the requested EID. The LTS then waits for an Iterable-Reply from its child. When the Iterable-Reply is received from the children, the LTS replies the original requester with an Iterable Mapping. To avoid memory leakage, the LTS must then implement a garbage collector mechanism to purge Map-Requests that have not been replies within an acceptable time frame.

Fig. 4.10 shows an example that mixes iterative and recursive modes. Node `N` runs in the iterative mode. It first sends a Map-Request to the `A` LTS. Node `A` runs in recursive mode. It then sends a Map-Request for the EID to one of its children, the `C` LTS, responsible for the EID 192.0.2.1. As `C` runs in iterative mode. It replies `A` with an Iterable-Reply. The mapping is then returned to `N` by `A`. `N` then sends a Map-Request to a locator, e.g., `E`, from the Iterable mapping. `E` replies with an Iterable-Reply containing ETR locators authoritative for 192.0.2.0/24, the most specific prefix covering 192.0.2.1. `N` then sends a Map-Request to one of the ETRs and finally receives the mapping for 192.0.2.1.

With the combination of Map-Request, Map-Reply and Iterable-Reply, it is not necessary to have an overlay deployed to build the mapping system, as long as the LISP-Tree Servers, Map-Servers and Map-Resovlers are reachable via locators. The deployment of LISP-Tree is thus flexible, simple and cost effective.

4.6 Related Work

On today's Internet, the DNS [MD95] is the system which is closest to a mapping system. Several researchers have proposed to extend it to store identity-to-location bindings, and provide mobility support in the Internet (e.g., [SB00, ABH08, Ahm05]). In this chapter we extend the DNS to support the LISP mapping system. We also propose to follow the same address allocation policy as in today's Internet. Further, by means of measurement-driven simulations, we show its feasibility along with performance and operational benefits.

Concerning the mapping systems, several architectures have been proposed for LISP [D. 09, BCF$^+$08, Lea08, MI08], but to our knowledge none of these systems have been evaluated in detail. Iannone and Bonaventure have analyzed in [IB07] the mapping cache used by ITRs. Using trace-driven simulation, they showed that the number of entries in this cache grows with the TTL of the cache and that the mapping system should provide mappings for EID prefixes and not individual EIDs. CoreSim also models the mapping cache on the ITR, but furthermore it models the entire mapping system and provides detailed information about its performance in terms of delay and load on mapping nodes. Other researchers [KCGR09, BFCW09] have analyzed the utilization of caches or similar techniques to reduce the size of FIB tables on routers.

Luo et al. proposed in [LQZ09] another LISP mapping system that relies on the CAN DHT. This mapping system provides mappings for individual EIDs instead of EID prefixes as [D. 09, BCF$^+$08, Lea08, MI08]. Using such flat EIDs is unlikely to scale. A trace-driven evaluation of this mapping system is provided in [LQZ09]. The evaluation mainly focuses on the size of the mapping cache and the number of hops through the CAN DHT. CoreSim models delays and is not dedicated to a single mapping system.

Menth et al. propose the *Future InteRnet Mapping System (FIRMS)* in [MHH10]. FIRMS is a mapping system for LISP where EIDs are assigned by prefixes to the sites. Each site maintains a map-base (MB) storing all the site's mappings. Meta information about the MB are put in a map-base pointer (MBP). A global table of MBP is maintained in a global distribution network. Every Map-Resolver copies the global MBP locally that it retrieves from the global MBP distribution network. To resolve a mapping for an EID, the MR first determines the address of the MB related to the EID by looking in it local copy of the global MBP. The MR then sends a Map-Request to that MB. Upon the reception of the Map-Request, the MB sends a Map-Reply to the requester with the mapping for the requested EID. Menth et al. also define the global MBP distribution network which consists of a hierarchical interconnection of map-base pointer exchange nodes (MBPXs) [MHH10]. The major difference between FIRMS and LISP-Tree is that in FIRMS, every Map-Resolver must store the global MBP table. This means that a MR in FIRMS must scale with the number of MBs and the resulting churn. In LISP-Tree, the MR only maintains information about the root LTSes. To optimize its operation, a MR can cache information about the LTSes on the path to the destination EID mappings that it resolves, however, this optimization is not mandatory and the MR scales with the traffic pattern of its clients. This difference implies that the Map-Requests are never forwarded between the MBPXs in FIRMS while they are potentially forwarded between the LTSes with LISP-Tree.

4.7 Conclusion

This thesis presents the LISP protocol and its ability to support traffic engineering. LISP is so flexible that it makes incoming performance based traffic engineering possible. However, the LISP traffic engineering capabilities rely on mappings. The mappings are used to distribute the IP addresses (i.e., the locators) to use to reach a destination. Changing the way the traffic is entering a LISP site thus simply consists of changing its mapping. With performance based traffic engineering, the mapping changes can be frequent. However, the mappings are distributed with the control plane and no mapping system has been proposed to scale with frequent mapping changes. In this chapter, we propose LISP-Tree, a DNS-Like mapping system that is designed to scale well. Like DNS, LISP-Tree relies on a hierarchy. The hirarchy forms a tree where the leaves store the mappings and the inner nodes are used to browse the hirarchy. When a xTR device must obtain a mapping, it queries the root of the hirarchy. The root determines which of its children potentially stores the mapping for the requested EID. The request is then redirected to this particular child. If the child is a leaf, it replies with the mapping. Otherwise, the node determines which of its children potentially stores the mapping and the request is redirected to this child. This operation is repeated until the mapping is found.

LISP-Tree supports two modes of operation. With the recursive mode, when a node receives a request, it sends the request to its child and the child to one of its children and so on until a leaf is reached. The reply then backtracks the tree and is finally returned to the requester by the root. The iterative mode is different. When a node receives a request, instead of sending it to its best children, it sends a message back to the requester that contains the address of the children that potentially know the mapping. The requester then sends the request to the nodes until the message is sent to a leaf. The advantage of the recursive more is that the tree traversal is transparent for the requester. However, all the requests always start from the root and the intermediate nodes in the hirarchy have to keep state about the requests they sent to their children. On the contrary, with the iterative mode, the requester sees the hirarchy. Because the requester knows the hirarchy, it can cache information about the intermediate nodes and it is then possible to avoid to start a mapping resolution from the root. This design has been proved to be scalable with DNS as the request will rapidly discover most of the hirarchy.

To ensure that a request will eventually reach a leaf that knows the mapping and scales, LISP-Tree must be deployed with care. The tree hirarchy is designed with the idea that each node is responsible for a prefix. The prefix covered by a node is always a subprefix of the prefix covered by its parent in the hirarchy. Each node keeps the information about the prefix it covers and the prefixes covered by all its children. The tree hirarchy is then a trie. In this chapter, we present a possible deployment of LISP-Tree in the current Internet and the expected performance of the lookups and the load distribution on this hirarchy. Based on the simulations performed with the CoreSIM simulator developed by UPC and real traffic traces, we show that LISP-Tree outperforms the LISP+ALT and LISP-DHT mapping systems.

LISP-Tree is designed such that it could be implemented in the DNS. However, DNS is not the most efficient protocol to process IP prefixes. We thus propose a simple extension to the LISP protocol to support LISP-Tree. Actually, the LISP protocol could be used as-is for LISP-Tree but overloading the semantic of the current LISP control-plane message is not a nice design on the long run.

Chapter 5

IDIPS

5.1 Introduction

We are seeing the emergence of technologies severely challenging some assumptions that have driven the development of many Internet protocols and mechanisms. A first assumption is that (usually) one address is associated to each host. Also, the forwarding of packets is often exclusively based on the destination address. For this reason, there is usually a single path between one source (or client) and one destination (or server). Finally, the Internet was designed with the client-server model in mind assuming that many clients receive information from (a smaller number of) servers. During the last years, these assumptions have been severely challenged.

The client-server model does not correspond to the current operation of many applications. First, large servers are usually replicated and different types of content distribution networks are used to efficiently distribute content [FFM03, Lim, Aka]. Second, the proliferation of peer-to-peer applications implies that most clients also act as servers. This is currently creating several problems in many Internet Service Provider (ISP) networks [KRP05]. Client-server asymmetry does not hold anymore.

Due to the transition from IPv4 to IPv6 many hosts will be dual-stack for the foreseeable future [CLH03]. Furthermore, measurements show that IPv4 and IPv6 do not always provide the same performances, even for a single source-destination pair [ZJUVM07]. This implies that to reach a destination supporting both IPv4 and IPv6, a source host can achieve better performance by selecting the stack that provides the best performance. However, today, this selection is based on simple heuristics. For instance, as highlighted by Matsumoto et al. [MFHK08], IPv6 is chosen prior to IPv4 in most of the dual-stack configurations although IPv4 still remains the best choice from a performance point of view in many environments [ZJUVM07].

We are thus heading towards an Internet that provides a set of potential paths between a source and a destination or a content. Obviously, any path in this set differs from the others as each path has its own performances, i.e., bandwidth, delay, loss, etc. In such a context, it is important for any application to select their paths in a way meeting their requirements (i.e., not in random way). For instance, bulk data transfer peer-to-peer clients will favor paths with the largest bandwidth so that the targeted file will be downloaded faster. Such a situation is currently discussed within the IETF *Application-Layer Traffic Optimization (ALTO)* Working Group [SB09].

A way to enable efficient path selection for applications would be to allow the network to cooperate with them. Such a cooperation would also give the opportunity to operators for managing incoming and outgoing traffic on their networks. Indeed, according to their traffic engineering needs, operators could balance traffic from one link to another and ensure that some are only used as backup.

In this chapter, we propose a generic informed path selection service called *ISP-Driven Informed Path Selection* (Idips). Idips is generic as it can be used in many networking contexts without changing anything to its behavior (see Sec. 5.2.2 for a description of networking use cases). It further does not require fundamental changes in the current Internet architecture and implementation (only the service clients need to implement a library for contacting the service), making its deployment very easy. Idips is scalable, lightweight, and designed to be easily deployed in ISP, corporate, or campus networks.

Idips is designed as a request/response service. The network operators deploy servers that are configured with policies and that collect routing information (e.g., OSPF/ISIS, BGP) and measurements towards popular destinations. The clients that need to select a path send requests to a server. A request contains a list of sources, a list of destinations and a traffic qualification that determines the rule for ranking the paths to use. The technique used by the client to discover the multiple paths it needs is not related to Idips. In this chapter, we assume that the client already knows the paths it want to get a rank for. The server replies with an ordered list of $< source, destination, rank >$ tuples to the client. The reply gives an indication of the ranking lifetime. This ranking is based on the current network state and policies. The client will then use the first pairs of the list and potentially switch to the next one(s) in case of problems or if it wants to use several paths in parallel.

We implemented Idips within the Xorp [HHK03] open source routing platform. We describe this implementation in the chapter and discuss cost functions, i.e., functions returning a cost for a given path allowing later ranking. We explain how to construct simple cost functions, such as maximizing the available bandwidth, and demonstrate how to combine them to reflect more complex ranking strategies.

Our evaluation of Idips focuses on the whole ranking process, from the request sent by the client to the use of cost function. We demonstrate that

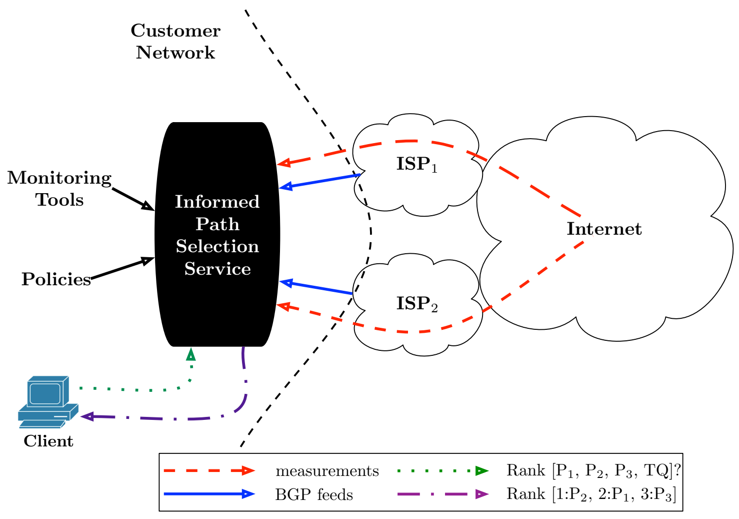

Customer Network
Monitoring Tools
Policies
Informed Path Selection Service
ISP$_1$
ISP$_2$
Internet
Client
measurements
BGP feeds
Rank [P$_1$, P$_2$, P$_3$, TQ]?
Rank [1:P$_2$, 2:P$_1$, 3:P$_3$]

Figure 5.1: A network service rank paths for the clients

our implementation is robust as IDIPS is able to process a large quantity of requests per second while providing a stable response time to the clients.

This chapter is organized as follows: Sec. 5.2 describes how an informed path selection service should work; Sec. 5.3 explains how the IDIPS service can be implemented; Sec. 5.4 evaluates our IDIPS implementation; Sec. 5.5 positions our work regarding the state of the art; finally, Sec. 5.6 concludes by summarizing our main achievements.

The very contributions of this chapter are (*i*) the proposition a of request/response service to rank paths in a scalable way. (*ii*) an architecture and an implementation for this service. Finally, (*iii*) an evaluation of the performance of this service with the implementation.

5.2 An Informed Path Selection Service

In this section, we provide a high level description of our informed path selection service. We first explain the behavior of this service (Sec. 5.2.1) before discussing several use cases that could benefit form the service (Sec. 5.2.2).

5.2.1 Overview

As illustrated in Fig. 5.1, an informed path selection service is a request/response service allowing to rank paths. The service is intended to be deployed at strategic points of the network by the network operator inside a domain, a campus, or a corporate network. Clients (see Sec. 5.2.2 for example of applications that can benefit from such a service) have to implement a library to send requests to the server and use the more preferable paths.

Clients see the service as a black-box. First, a client sends a request to the server (dotted arrow on Fig. 5.1). This request contains a list of sources and destinations, forming a set of paths to rank, and a traffic qualification (TQ on Fig. 5.1). The latter argument is a ranking criterion provided by the client. It might be, for instance, "maximize the bandwidth", "minimize the delay", or "maximize locality". The server replies to the request with a ranked list of paths (e.g., source-destination pairs - dashed-dotted arrow on Fig. 5.1). The way the ranking is computed by the server remains hidden for the client.

The presence of multiple sources in the request comes from particular situations, such as multihomed hosts (for instance a smartphone with a 3G and a standard WiFi connection) or IPv4/v6 dual-stack hosts, where hosts own several addresses.

The server ranks the paths with the help of information from the network:

- Routing information (i.e., BGP, OSPF/ISIS) allowing the informed path selection service to compare different paths based on their routing metrics (e.g., BGP local preference or AS path, IGP cost, etc). This is illustrated in Fig. 5.1 with the plain arrows.

- Active or passive measurements (i.e., delay, bandwidth, loss, etc) allowing the informed path selection service to compare different paths based on quantitative performance metrics. Note that the server does not necessarily perform the measurements itself. It can request a third party to do the job and retrieve the required performance metrics. This is illustrated with dashed arrows on Fig. 5.1.

- Policies configured by the network administrator that indicate preferences for some paths over others.

The information that the server can access depend on its location and how it is operated. If IDIPS is deployed directly in the operator network, it can access routing information and policies. However, if IDIPS is operated by a third party, it is unlikely that the network will share all these information making the decisions less precise.

Upon reception of a request, the informed path selection service builds a list of all the possible paths between the source(s) and the destination(s). Then, it removes from consideration the paths that are invalid due to routing (e.g., one destination is not reachable from a given source) or policies. The remaining paths are then ranked according to a set of criteria and the reply sent back to the client contains the following information:

- the best path (source, destination),

- the second best path (source, destination),
- ...
- the Nth best path (source, destination)
- the lifetime for the ranked paths.

and for each path its associated rank.

The number of paths returned by the service may be lower, as indicated above, than the total number of possible paths. An evaluation of the overhead of returning the paths is provided in Sec. 5.4.

Ranking is valid for some time and the client is encouraged to cache the ordered list for the lifetime indicated in the response.

Source and destinations can be IP addresses (either IPv4 or IPv6 or both), IP prefixes, AS numbers, names or of any other type, as long as the client and the server agree on a meaning for them. This possibility to use IP prefixes is motivated by the fact that contiguous IP addresses tend to be used similarly and present similar performances [CH10, SDB09].

IDIPS is designed to be as generic as possible and to prevent an operator to reveal critical information about its topology. This is the reason why the only information returned to the client is the path rank. The rank is an abstraction of information known by the server that defines a partial order among the paths. The order is valid only within one reply and the ranking relationship among any two paths in the reply holds the reflexivity, antisymmetry and transitivity properties.

Path rank is the information revealed to the clients and is computed at the server side with the help of *cost functions*. A cost function determines the cost of using a path. The higher the cost, the less interesting the path. The cost of a path is computed from its *attributes*. The path attributes are the information related to the path. Example of path attributes are its predicted round-trip delay, its reachability or even the AS path from the RIB.

Sec. 5.2.2 discusses different use cases where IDIPS can be used.

5.2.2 Use Cases

A client of the informed path selection service refers to any entity that has the possibility to select a path to reach a destination or get a content. IDIPS is thus not limited to the user level applications but can also be used directly to improve the routing with the control plane requesting information from IDIPS to make better choices. In this section, we explore several networking scenarios in which the service might find a suitable usage, demonstrating so the general purpose of the service. Additional use cases described in [SB09] can be applied to our informed path selection service.

Peer-to-Peer

Peer-to-peer applications are clear candidate users for such an informed path selection service. Each time a peer-to-peer client wants to fetch a given content, it can use the service to rank the various peers sharing the content and, then, select the peer based on the service reply instead of randomly selecting a peer. Ranking, from the client perspective, might be done in order to maximize the throughput, so that the best path will allow a faster download of the content.

Furthermore, not only peer-to-peer clients can benefit from the service but also the ISPs. Indeed, an ISP or a campus network running the informed path selection service could influence providers used by packets sent/received by hosts of its networks. One can imagine an ISP giving priority to peer-to-peer clients in its own network or in those with who it has a shared-cost peering relationship, avoiding so to pay traffic to its own provider.

The rank is the information abstraction revealed to the client. Internally, an IDIPS server relies on path attributes and cost functions.

Solutions allowing ISPs to help peer-to-peer clients have already been proposed [XYK+08, AFS07, PM08, KPS+11]. See Sec. 5.5 for more discussion about the related work and how our informed path service differs from those solutions.

IPv4/IPv6 Transition

Due to the transition from IPv4 to IPv6 many hosts will be dual-stack for the foreseeable future [CLH03]. Furthermore, measurements show that, in today's Internet, IPv4 and IPv6 do not provide the same performances, even for a single source-destination pair [ZJUVM07]. This implies that to reach a destination supporting both IPv4 and IPv6, a source can use the informed path service selection to achieve better performance by selecting the stack that provides the best performance.

Today, this selection is based on simple heuristics. For instance, as highlighted by Matsumoto et al. [MFHK08], IPv6 is chosen prior to IPv4 in most of the dual-stack configuration although IPv4 still remains the best choice in most of the environments [ZJUVM07].

Multihoming

An increasing number of ISPs, but also campus and corporate networks have chosen to become multihomed by being attached to two or more ISPs [ACK03]. For these networks, multihoming offers two main benefits: technical and economical redundancy, i.e., they remain connected to the Internet even if the link that attaches them to one of their ISPs fails or if one of their ISPs becomes bankrupt. Another important benefit shown by several studies [APS04, AAS03, AMS+08, GDZ06, DD06] is that multihoming allows sites to choose better quality paths over the Internet.

An informed path selection service can find here an usage, for the ISP, in selecting the best provider to reach a given destination/content.

Local Network Optimization

The use cases presented above are mostly considering interdomain traffic. Nevertheless, Idips can be used for intradomain application. For example, a network optimizing its IGP with OSPF-TE [KKY03] could obtain traffic engineering information regarding its local links via Idips. In this case, the Idips server would monitor internal path and use information from the IGP to determine the best paths to use. Although this application is promising, we have not investigated more.

Traffic Engineering with LISP

LISP, described in Chapter 2, relies on mappings. A mapping associates a list of locators to an identifier. A priority and a weight are associated to each locator. Only the locators with the lowest priority value can be used, the others can be used only for backup. The weight is used to balance the traffic between the most preferable locators (see Chapter 3). Idips can be used by the mapping owner to determine the priority and the weight of the locators in a mapping. The priorities and weights in LISP offer traffic engineering capabilities to the LISP sites.

Idips is a service that proposes to rank paths in order to use those that offer the best performance. However, using a path might influence its performance and result in oscillation [AAS03, GDZ06]. Therefore, the ranks must be computed with care to avoid this risk. Nevertheless, this thesis does not aim at solving this well known oscillation problem [AAS03, GDZ06].

5.3 Service Construction

In this section, we describe how we implement an informed path selection service. Our implementation is called *ISP-Driven Informed Path Selection* (Idips) and it architecture is described in Sec. 5.3.1. We further discuss how our implementation is included in the Xorp framework [HHK03] (Sec. 5.3.2). Finally, we explain in details how to build simple cost functions and combine them to reflect more complex ranking strategies (Sec. 5.3.3).

5.3.1 Architecture

Idips is composed of three independent modules: the *Querying* module, the *Prediction* module, and the *Measurement* module. The Querying module is directly in relation with the client as it is in charge of receiving the requests, computing the path ranking based on traffic qualification provided by the client and the ISP traffic engineering requirements, and replying with the ranked paths. For the sake of generality, the remainder of this chapter will use the term *ranking criterion* when referring to traffic qualification. The Measurement module is in charge of measuring path performance metrics if required. Finally, the Prediction module is used to predicting paths performance (i.e., future performance metrics of a given path based on the past measurements).

The ranking criterion provided by clients in their requests might require measuring the network to obtain path performance metrics, such as delay or bandwidth estimation. One of the key advantages of IDIPS is that it avoids clients measuring themselves the network, leading to redundant traffic injected in the network (See Chapter 6). The Measurement module performs the measurements or asks a third-party to perform the measurements. Those measurements can be active (i.e., probes are sent in the network) or passive (i.e., no additional traffic is injected).

It is possible to predict the performance of a given path if it has been previously measured [YRCR04, DCKM04, PLMS06, Pap07, dLUB05, WSS05, LPS06, LGS07, LHC03, LGP+05, NZ04, FJP+99, NZ02, PCW+03, ST03, LHC05, CCRK04, RMK+08, MS04]. This prediction task is achieved by the Prediction module. Note that a given measurement can be used in several different predictions. For instance, the previous delay measurements can serve for predicting the delay, the jitter, or for determining whether the path is reachable or not.

To enable flexibility, ease of implementation and performance[1], IDIPS clearly separates the Querying, Measurement and Prediction modules. Each instance module communicates with the other modules thanks to a standardized interface. Therefore, the handling of requests from the clients is strictly separated from the prediction of path performance and path performance prediction is separated from path measurements.

The Querying module receives the ranking requests from the clients and computes the rank for these requested paths based on their predicted future performance. Future paths performance are estimated by the Prediction module that relies on the measurements performed by the Measurement modules.

All along this chapter, we are using the terms measurements and predictions however, they have to be understood in their very generic meaning. For IDIPS, a measurement corresponds to any information grabbed from the network. This definition encompasses active measurements like pings, passive measurements like Netflow information [Cla04] or even routing information like BGP feeds. Likewise, a prediction in IDIPS is an information

[1]IDIPS must potentially handle many ranking requests simultaneously

that is likely to be valid in the coming future. Therefore, a prediction can be the result of very complex machine learning techniques but also very simple information like the originating AS of the path destination. In other word, a measurement is an information discovered in the past or just at present and a prediction is an information that is likely to be valid in the coming future.

To support as many requests per second as possible, the IDIPS modules are running independently of each others. This independence is ensured through the use of caches. Each module stores its processing results in its local cache. If another module requests a given result, a simple `get` in the appropriate cache will return it.

There may exist several instances of the Prediction and Measurement modules. For example, IDIPS can have a delay measurement module, a bandwidth measurement module, a delay prediction module, and a bandwidth prediction module. Sec. 5.3.2 gives an example of Measurement and Prediction modules implementation.

Querying module

Common applications are only able to use one path at a time, even if several exist. In this case, the client only needs to know the very best path returned by IDIPS when it has no additional information about the paths. For this reason, the list of ranked path is sorted by rank before being transmitted to the client. Then, the client can safely consider the first path of the list as the very best path (or one best path among all the best paths if several ones have the same lowest rank value). The other paths are returned only for resiliency (the best path is not valid for the client) or if the client uses the ranked list to refine a local decision. Sorting the paths simplify the operation at the client.

Paths ranking is done with the use of *Cost Function.* For a given <source, destination> pair, the cost function returns a cost, i.e., a positive integer resulting from metrics combination of a given path. The lower the path cost, the more attractive the path. We chose the cost to be represented by a positive integer for its simplicity (i.e., no complex representation to be processed) and because operators are already used to translate their policies into integers with the BGP local-pref [RLH06]. By definition, the sum of several costs is also a cost. One can for example combine cost functions with an exponentially weighted sum in order to reflect complex strategies or politics as long as the result is rounded into a positive integer. Sec. 5.3.3 explains how to construct cost functions.

To support as many requests per second as possible, the IDIPS modules are running independently of each others. This means that the Querying module never has to wait for a path performance prediction to be computed by the Prediction module to compute the path ranking. When a prediction

```
sync_rank_paths
         ? sources & destinations & criterion
         -> ranked_paths_list & ttl
```

Figure 5.2: Idips server API for synchronous mode clients

has to be retrieved by the Querying module, it calls a `get` on the Prediction module for the path attribute it is interested in. The attributes of a path are the predicted metric values as computed by the Prediction module for the path. For the sake of generality, any attribute is encoded as an integer. If an information is too complex to be represented with a single integer, it can always be represented as a set of integers. For example, an $< x, y >$ coordinates can be decomposed in the `x_coordinate` and the `y_coordinate` and a function that needs to use the coordinates just needs to retrieve the `x_coordinate` and the `y_coordinate` to reconstruct the full coordinates. Sec. 5.3.1 gives more details about the interface to retrieve path attributes from the Prediction module.

Depending on its needs, a client can query Idips in a *synchronous* or *asynchronous* way. In synchronous mode, when a request is received by the Idips server, the server sends the list of ranked paths back to the client once computed. On the contrary, in asynchronous mode, when a request is received by the Idips server, the server computes the paths ranking but does not send the list back to the client. The requester must explicitly send a special command to retrieve the list of ranked paths. The API that Idips presents to clients is depicted in Fig. 5.2 for the synchronous mode and in Fig. 5.3 for the asynchronous mode.

The commands are sent by the client to the server. When the client uses the asynchronous mode, it receives a transaction identifier (*tid*) back from the server. Every request received by a server is abstracted as a transaction. This *tid* is the identifier of that transaction on the server. This identifier is used for retrieving the list of ranked path with the `get_all_path_ranks`. If the ranking is not yet computed by the server when the `get_all_path_ranks` is received, an empty list of ranked paths and the invalid `0x0` ttl are returned. The server, in asynchronous mode, always returns immediately a result when it receives an `async_rank_paths` or a `get_all_path_ranks`. The client must then poll the server until it has retrieved the list. This behavior is used to avoid the server to maintain too much state about the clients, it only maintains ranking state (linked with *tid*). To avoid the need of client polling, signaling could be used to let the server inform the client that the transaction is ready but it thus means that the server must maintain state about the client, which is what we want to avoid while using asynchronous mode. Polling is by definition avoided in synchronous mode. It is worth to notice that a ranking call can be implemented as being blocking or non-blocking at

```
async_rank_paths
         ? sources & destinations & criterion
         -> tid

get_all_path_ranks
         ? tid
         -> ranked_paths_list & ttl

get_next_path_rank
         ? tid
         -> source & destination & rank & ttl & more

get_next_n_path_ranks
         ? tid & n
         -> ranked_paths_list & ttl & more

terminate_transaction
         ? tid
```

Figure 5.3: IDIPS server API for asynchronous mode clients

the client side, independently of the client to server communication mode. The typical use of a blocking call is when the path to exchange data cannot be changed once the flow is started. Then, the best path must be used. The client must then wait for the path ranking before being able to exchange data. On the contrary, non-blocking call is used when the client can change the path it uses while exchanging data. For example, a shim6 [NB09] host starts exchanging data with a path arbitrarily selected by following the rules of RFC3484 [Dra03]. If the data transfer is long enough, shim6 could decide to switch to the best path computed by IDIPS. In this case, the flow can start as soon as possible, even if the path used to exchange data might be sub-optimal at the beginning.

To avoid this waste of resources, IDIPS also offers the possibility to retrieve one path at a time with the `get_next_path_rank` that returns the best path that has not yet been retrieved by the client. To use the best working path, the client can use the algorithm presented in Fig. 5.4 where `handle_path` is the client function that needs the path and that returns true when no more path is needed. The `more` parameter returned by the `get_next_path_rank` indicates if there is still a path to retrieve for the transaction. Optionally, the client can explicitly ask IDIPS to terminate the transaction. If not, IDIPS should eventually terminate it automatically. Instead of considering retrieving the rankings one by one or all at a time, the more generic `get_next_n_path_ranks` is also proposed where the client specifies

```
more := true

WHILE more
DO
   (src, dst, rank, more) := get_next_path_rank(tid)
   IF handle_path(srcs, dst, rank)
   THEN
       STOP
   END
DONE

terminate_transaction(tid)
```

Figure 5.4: One-by-one path ranking retrieval algorithm

the number of paths that must be returned by IDIPS. The equivalent of GET_ALL_PATH_RANKS corresponding to a specified number higher or equal to the number of sources time the number of destinations while a value equal to one corresponds to the `get_next_path_rank`. However, in most of the cases, a client is interested by either one or all the paths.

We suggest to use UDP to exchange message between the clients and the servers. Using UDP avoids the burden of establishing and maintaining TCP connections. However, IDIPS does not preclude the use of another protocol or even several protocols at the same time. For example, a server can be requestable via UDP by lightweight clients and propose a more complex interface via HTTP/XML over TCP for more powerful clients.

Changing the paths to always use the ones with the best performance might result in oscillations [AAS03, GDZ06]. Mechanisms to avoid oscillations [AAS03, GDZ06] can be implemented in the Querying module. However, dealing with the oscillation problem is out of the scope of our study that focuses on the architectural part of the performance based traffic engineering problem.

Measurement Module

The Measurement module is in charge of measuring the paths. The measurements can be active or passive. For example, an active measurement could be a ping and a passive measurement could be the count of the number of TCP SYNs entering the network.

The Measurement module API presented in Fig. 5.5 is two fold. The `start_measurement`, `stop_measurement` and `set_interval` commands determine the targets to measure while the `get_measurements` is used to retrieve the last measurements of a path.

```
start_measurement ?  source & destination & interval

stop_measurement  ?  source & destination

set_interval      ?  source & destination & interval

get_measurements  ?  source & destination
                  -> measurements
```

Figure 5.5: Measurement module API

Measurements are always defined between a source and a destination and are performed periodically (with a configurable interval between the measurements). In case of passive measurements, the sources and destinations as well as the passively obtained information are extracted periodically from the passively collected traces. The possibility to modify the interval of a measurement is not mandatory but is more convenient as it allows one to adapt the measurement rate dynamically without disrupting a measurement campaign. If such command is not available, it means that the measured values must be stored outside the measurement module. Indeed, without the `set_interval` command, the measurement has to be stopped, then restarted from scratch meaning that all the state in the measurement module instance is lost for this measurement. Finally, the `get_measurements` command returns all the measurements performed so far for the <source, destination>.

It is important to notice that the decision of measuring a path is done either by configuration or triggered by the Prediction module, not directly by the requests. However, the content of the requests can be seen as passive measurements and can be used to dynamically determine the paths to measure.

Prediction Module

The prediction module contains all the intelligence of IDIPS. Indeed, IDIPS is a service that aims at determining the best paths to use. However, determining the best path to use is a prediction exercise as the future behavior of a path is seldom know, particularly when considering inter-domain paths. Determining how to predict a path behavior the best is out of the scope of this chapter but this section presents how a Prediction module has to be implemented in IDIPS.

As already expressed earlier, IDIPS modules are running independently. However, the Querying module needs to know the path attributes computed

```
start_prediction ?  path

stop_prediction  ?  path

get_prediction   ?  path
                 -> prediction
```

Figure 5.6: Prediction module API

by the Prediction module. In addition, the Prediction module has to know the path it has to predict the performance metric for. To this aim, the Prediction module provides the API presented in Fig. 5.6.

This API has two components. On the one hand, the `start_prediction` and `stop_prediction` commands are used to specify the path to predict performance metric for. On the other hand, the `get_prediction` command is used to retrieve the predictions.

`get_prediction` always returns a value. If the attribute value is not defined, an error or a meaningful default value is returned. For example, if the bandwidth of a path is not known a default value of zero can be returned making the path less interesting than any other path.

The decision of measuring or predicting a path is highly related to the deployment policies, the topology and the traffic. The decision of predicting a path is thus not provided by IDIPS but is considered case by case by the Prediction module or by configuration. There exists three ways of determining if a prediction has to be started or stopped. First, an operator can manually determine the path to do prediction for and uses `start_prediction` and `stop_prediction` commands to do so. Second, a prediction module instance can determine by itself if a path is worth being measured or not. For example, if a prediction module received enough `get_prediction` for a path it is not predicting yet, it can decide to start predicting it. In this second case, the `start_prediction` and `stop_prediction` commands are not used. Finally, a prediction module instance can predict that a path has to be predicted and command another prediction module to start predicting the path. For example, a prediction module instance can be in charge of predicting if a path is important or not based on the traffic it carries. If the path is considered as important, it can ask to start the delay prediction for that particular important path.

To predict the future path behavior, a prediction module often needs information from the measurement module. Like the Querying module can retrieve a prediction with a simple `get`, the Prediction module can retrieve the measurements from the Measurement module with the `get_measurements` (see Sec. 5.3.1). The Prediction module can use the last measurements to predict the future behavior of a path. Based on the prediction and on its quality, the prediction module can decide to modify the frequency at which a measurement has to be performed (with the `sec_interval` command) or ultimately to start or stop a measurement. In addition, because a prediction module aims at providing the path performance for the near future, the `get_prediction` only returns one result as opposed to `get_measurements` that returns a list of measurements. Obviously, this API does not preclude an extended API that would return more information about the quality of the prediction (for example a TTL) or several predictions at once.

Path asymmetry is common in the Internet [PPZ+08] and some metrics like the bandwidth strongly depends on the followed direction. It is then important that the IDIPS measurement and prediction modules take this factor into account to accurately rank the paths.

5.3.2 XORP Implementation

Sec. 5.3.1 presents a potential generic architecture for IDIPS. In this section, we present how we have implemented IDIPS within the XORP framework [HHK03]. XORP is an open source routing platform that facilitates the implementation of control plane protocols. In XORP, each control plane protocol (this including their communication channels) is implemented as an independent process. The XORP inter-process communication is ensured by the use of the *Xorp Resource Locators (XRL)*. An XRL is very similar to RPC but much simpler to implement and use. XRLs are always asynchronous. It means that once a process has sent an XRL, it does not block waiting the answer. When the target of the XRL has computed the result, it will instead trigger the call of a callback function at the process that sent the XRL. This is possible because a callback function (a C++ method reference) is always associated to an XRL call. The callback is just a call to the method on the instance with the parameters set to the values computed by the target of the XRL. We have chosen to implement IDIPS within XORP for two reasons. First, implementing IDIPS in XORP gives direct access to the routing table, meaning that routing information can be easily injected into IDIPS. XORP provides implementations for the most common control plane protocols, e.g., BGP, with well defined XRL interfaces to communicate with. Second, XORP is designed to be distributed. It means that it is possible to have processes interacting together but running on different hosts. We do not use process distribution in our prototype however, one could imagine running the prediction modules on dedicated servers or distribute the measurement module.

As in the generic architecture presented in Sec. 5.3.1, our implementation is decomposed in the Querying, Prediction and Measurement modules. The querying module is a single XORP process, while there are as many XORP processes as required to implement the Measurement and Prediction modules. For example, if IDIPS requires to measure the delay and the bandwidth,

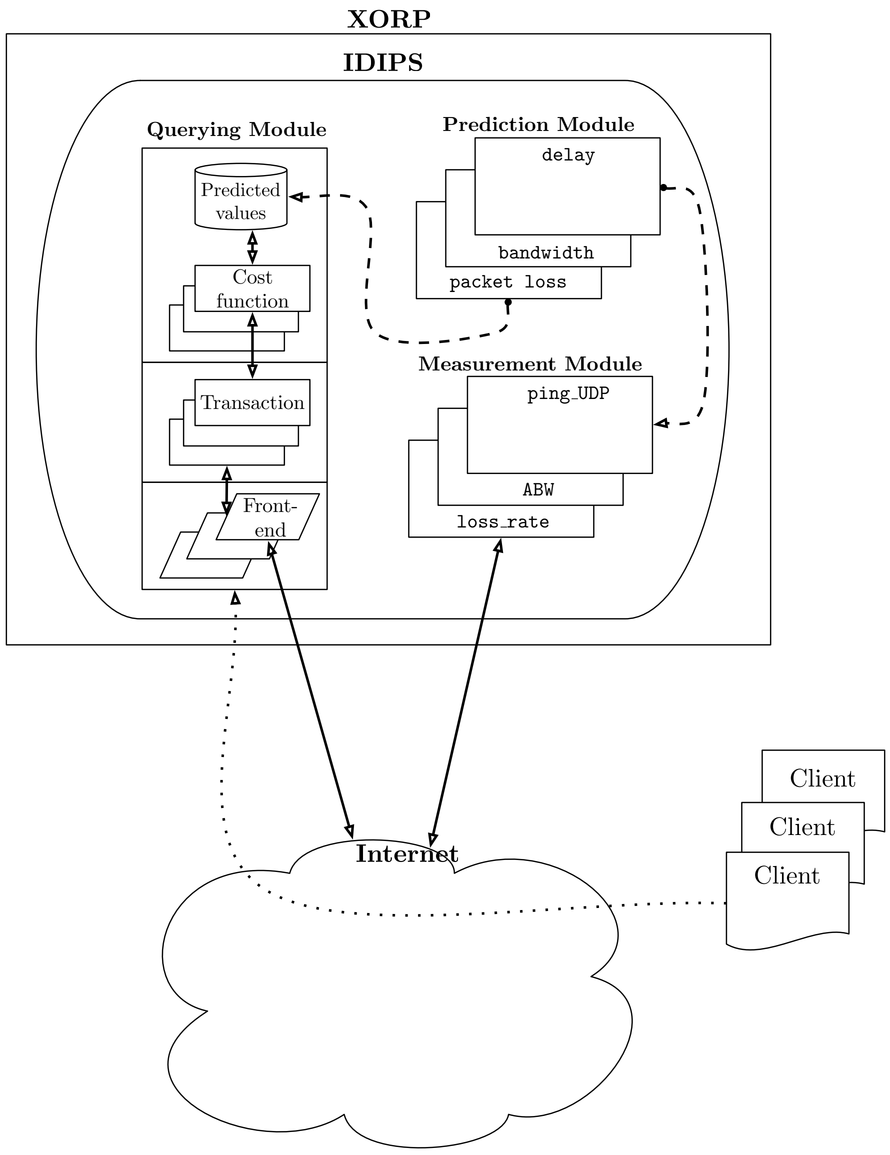


Figure 5.7: IDIPS within XORP

the Measurement module will contain two XORP module. One implementing a delay measurement and the other implementing the bandwidth measurement. Fig. 5.7 shows the IDIPS architecture in XORP, while Fig. 5.8 shows how the different modules interact with each others.

The Querying module is decomposed in three main parts: (*i*) the *Front-end* part, (*ii*) the *Transaction* part and (*iii*) the *Cost function* part. The Front-end part receives the requests from clients and returns the ranking re-

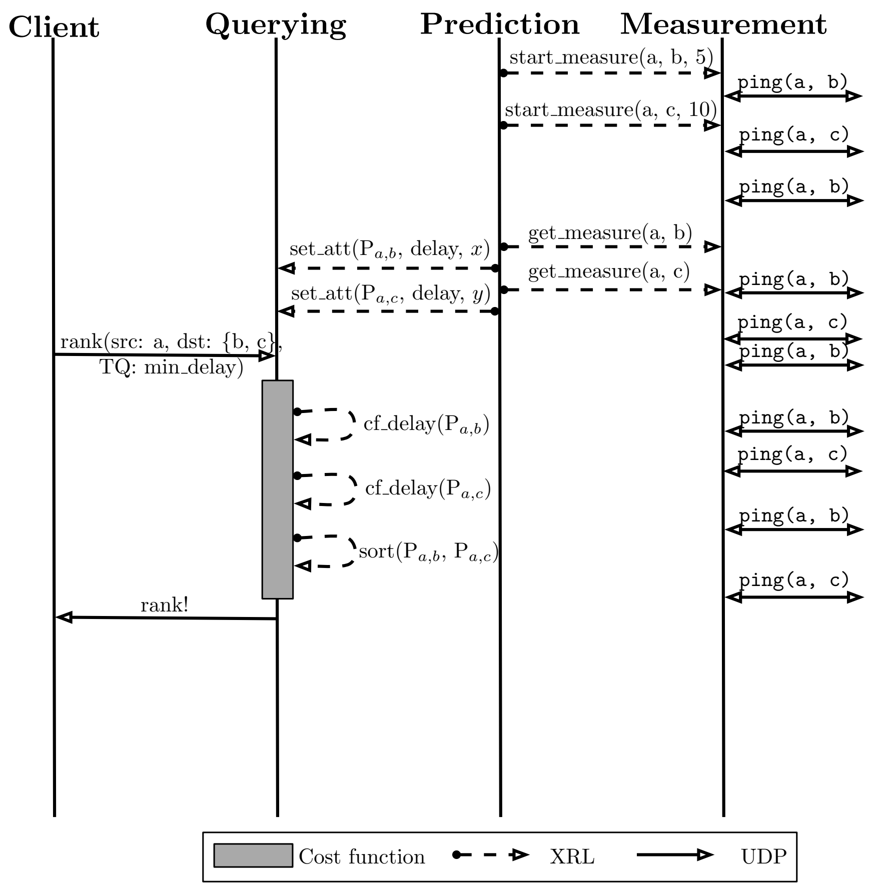


Figure 5.8: Example of modules interactions in IDIPS

sults. The Transaction part processes the requests received by the Front-end and computes the path rank for these requests. Finally, the Cost function part implements the cost functions. Our implementation has two different Front-ends. On the one hand, a client can request IDIPS to rank paths by sending UDP messages. On the other hand, any XORP process can request paths ranking just by sending an XRL to the Querying module. The typical use of UDP messages is for clients independent of IDIPS, e.g., a P2P client while the XRL interface allows any XORP process to use IDIPS to improve its decision. For example, the FIB as computed by XORP could be optimized by taking IDIPS ranks into account.

IDIPS must potentially handle many ranking requests at the same time. To support a potentially high load, requests are abstracted into *transactions*. Therefore, for each request, a transaction instance is created by a unique identifier. Each transaction runs independently of the others and maintains the list of sources, the list of destinations and the path ranking criterion. If the request uses the synchronous mode, the transaction also maintains information to send the reply to the requester. When a request is received, the Front-end instantiates an empty transaction and adds all the paths from the request. At that stage, the paths are computed blindly: for each source s, for each destination d in the source and destination lists, the $< s, d >$ path is added to the transaction. Once all the paths are added to the transaction, the `run` method is called on the instance.

The job of the `run` method is to determine the cost and the rank of each path, according to the path ranking criterion and to build the sorted list of ranked paths. A transaction is *ready* once the ordered list of ranked paths has been built completely. The cost of each path is determined by calling the appropriated cost function on the path. Once the cost is determined for a path, the path, tagged with its cost is added to the priority queue `_costs`. The `_costs` structure is maintained ordered by the path cost. It means that at any time, the ith entry in `_costs` has a lower or equal cost than the $i+1$th entry. The transaction is set ready once the cost and rank of each path is known. If the request was in synchronous mode, the transaction triggers the transmission of the reply to the request once the transaction becomes ready. If the request was in asynchronous mode, the method stops. As long as the transaction is not ready, a call to retrieve a path for an asynchronous mode request returns an error.

We have implemented the call of the cost function in two different ways. By using XRL or by directly calling the method on the querying module class instance. We use the XRLs to parallelize the processing. However, as the processing of XRL is centralized (via the `finder`) and because the management of XRLs is sequential and implemented with a list, this implementation does not improve the performance, even worse, it reduces the number of requests IDIPS is able to sustain and may cause ranking failures because XRLs can be lost. Indeed, the XRLs are enqueued in a list limited in size and number of entries. Therefore, once the list is full, XRLs can be lost. The performance can also drop because an XRL at position i in the queue will not be dispatched to the Querying module before the XRLs prior to position i have been dispatched to their target process. Even if the processes are different. With an experiment where the requests ask to rank 50 different paths, we observed a drop of 54% of requests per second supported by IDIPS compared to an implementation calling the cost function directly without XRLs. We also noticed about 12% of failing transactions and a time to compute the rank 56 times higher with the XRL implementation. However, for the requests that succeeded, the time perceived by the client was 13% faster with the XRL implementation. The time perceived by the client is the time elapsed between the sending of the request and the reception of the ordered list of ranked paths. Despite the better client perceived time with the XRL implementation, we recommend not to use the XRL implementation. Indeed, without the use of XRLs, IDIPS can handle more simultaneous requests and does not face loss of requests due to the limited size of the XRL queue.

Sec. 5.3.1 proposes to keep the modules independent thanks to the use of getter functions: when a module needs information from another module, it sends a get to the module to retrieve the values. In our implementation, every module implements such getters. However, we have also implemented a paths attributes cache within the Querying module. This cache stores, for each path, all the known attributes for the paths. The attribute values are computed by the Prediction module. This cache is based on a *push* model. It means that it is not the Querying module that populates it but the Prediction module that pushes the values to that cache. The querying module thus implements the `set_attribute` and `get_attribute` XRLs. Therefore, when a prediction is computed, the Prediction module immediately calls the `set_attribute` XRL on the Querying module to set the attribute value for the path that as just been computed. This mechanism is implemented to speed-up the cost computation for the paths. Indeed, as presented in Sec. 5.3.3, the cost of a path is computed with a cost function that potentially needs the attributes of the path. Thus, without an attribute cache at the Querying module, an XRL must be called to the appropriate Prediction module instance for each attribute to retrieve. However, calling XRL implies some delay as the call must be sent first to the main XORP process, i.e., the finder, then it is sent to the appropriate Prediction module instance. This delay can become non negligible if the Prediction module is not running on the same host as the Querying module. For this reason, the Querying module does only rely on this cache. If the cache has no entry for the path attribute, it is considered that the path is not under measurement/prediction and the Cost function must determine an appropriate cost. It is important to remark that our implementation does not allow the prediction module to determine by itself that a path merits to be predicted. Indeed, the Querying module never calls the `get_attribute` on the prediction module so the prediction module cannot count the number of failing calls. However, one could imagine a measurement module instance monitoring the cache misses at the querying module. The prediction module could then determine the paths that are worth being predicted.

The notion of module is translated into XRL interfaces in XORP. Except for the Querying module, there might be several C++ classes implementing a module and possibly several instances of a class as illustrated in Fig. 5.7. Each class must implement the XRL interface corresponding to the module it is related to. Fig. 5.9 gives the XRLs that must be implemented by the class implementing the Querying module. It is important to notice that the interface for the querying module is only composed of the setter and the getter for the path attributes. It does not include an interface for clients to query IDIPS. Indeed, XRL interfaces are only related to the implementation.

```
interface idips_querying/0.1 {
  /**
   * Get a path attribute
   * @param path to get the attribute from
   * @param name of the attribute
   * @param value of the path attribute
   * @param rpath echo of path
   */
  get_attribute?path:txt&name:txt->value:u32&rpath:txt;

  /**
   * Set a path attribute
   * @param path to set the attribute to
   * @param name of the attribute
   * @param value of the path attribute
   */
  set_attribute?path:txt&name:txt&value:u32;
}
```

Figure 5.9: XORP Querying module XRL interface

Nevertheless, we have implemented the client-related commands described in Fig. 5.2 and Fig. 5.3 with XRLs to make IDIPS usable directly by any XORP process. It means that our IDIPS implementation has two front-ends, one listening on UDP and the other listening on XRLs. Fig. 5.10 lists the XRLs that must be implemented by the classes implementing the Measurement module. Finally, Fig. 5.11 shows the XRLs that the classes implementing the Prediction module must implement. Each class implementing one and only one technique. For example, one class can implement a UDP ping for the measurement module and another can measure the path bandwidth and one class can implement a delay bandwidth product predictor based on the delay and bandwidth measurements.

The whole process is presented in Fig. 5.8. The Prediction module asks an instance of the Measurement module (i.e., the delay measurement instance) to measure a path. A path to measure is defined by a source and a destination. For the sake of generality, the source and the endpoint of any path to measure is represented textually, meaning that it can be a name, an IP address a network interface, or any other suitable information. Each path installed in a measurement module is periodically measured with a configurable interval between measurements (e.g., 5 for path (a,b) and 10 for (a,c) in Fig. 5.8). The use of IP prefixes instead of IP addresses is particularly interesting to aggregate information. For example, if a site has one IP prefix p/P for its clients and that the performance are considered to be the same for any of them, then all the paths can be aggregated by using the p/P source instead of the client IP address.

```
interface idips_measurement/0.1 {
  /**
   * Start periodically measuring a destination
   * @param destination destination to measure
   * @param interval interval in seconds between two measurements
   */
  start_measurement?source:txt&destination:txt&interval:u32;

  /**
   * Stop measuring a destination
   * @param destination destination to stop measuring
   */
  stop_measurement?source:txt&destination:txt;

  /**
   * Change measurement interval for a destination
   * @param destination destination to change the measurement interval
   * @param interval new measurement interval for the destination
   */
  set_interval?source:txt&destination:txt&interval:u32;

  /**
   * @params destination  destination to get the past measurements
   * @params measurements list of measurements
   * @params clean remove elements after retrieving them
   */
  get_measurements?source:txt&destination:txt
              &clean:bool->measurements:list<u32>;
}
```

Figure 5.10: XORP Measurement module XRL interface

```
interface idips_prediction/0.1 {
  /**
   * Start a prediction model for a path
   * @param path to predict
   * @param src source IP for the measurements
   * @param dst destination IP for the measurements
   */
  start_prediction?path:txt&src:ipv4&dst:ipv4;

  /**
   * Stop a prediction model for a path
   * @param path to stop the prediction for
   */
  stop_prediction?path:txt;

  /**
   * Get the prediction for a path
   * @param path to get the prediction for
   * @param prediction for the path
   */
  get_prediction?path:txt->prediction:u32;
}
```

Figure 5.11: XORP Prediction module XRL interface

The `start_measurement` XRL function triggers the measurement of the path defined by the `source` and `destination` parameters. The path is then measured every `interval` seconds (e.g., 5 for the path (a,b) in Fig. 5.8).[2]

The various Measurements module instances keep locally the last measurements they obtained for the paths they are measuring. When a Prediction module needs a measurement, it sends a `get_measurement` XRL to the adequate instance of the Measurement module and retrieves the measurements for the path. The measurement is then sent to the Querying module, with the `set_attribute` function, for being stored in the Predicted values storage.

Examples of module implementation

This section presents two examples of module implementation. We first present a Measurement module that implements a UDP ping and then describe a Prediction module that implements an average delay predictor. The Prediction module uses the measurement module to predict the delay of the paths.

[2]To avoid synchronization, the time between two measurements should be set to be equal to the `interval` parameter on average.

Measurement module example For the sake of the example, we propose a UDP ping Measurement module. This module does not aim at being used in a real environment where more robust measurements techniques should be used. To estimate the round-trip delay between a <source, destination> IP pair, we send a UDP segment to the destination on a port number that is very unlikely to be open. If the port is not opened and if no filtering applies, an ICMP port unreachable is expected to be returned to the Measurement module. The sending of the UDP segments is done by using the XORP socket API. XORP sockets are similar to the POSIX sockets except that they are asynchronous and that they are implemented with XRLs. In the reminder of this section we will use the term *socket* to refer to the XORP socket abstraction. A XORP process that wants to use a socket has to implement the `socket4_user`[3] XRL interface. This interface defines several XRL like `error_event` or `recv_event` that respectively indicate if an error occurred with the socket or if bytes are ready to be read on a socket. The `socket4_user` is used to signal the XORP process about events on the sockets it is in charge of. To open, bind, connect, listen, send data on or close a socket, the IDIPS must use an XRL Socket Client. XRL Sockets Clients are classes that implement the `socket4` XRL interface and are directly provided in the XORP framework.

To implement the UDP ping, we create one connected UDP socket per <source, destination> IP pair and periodically send a UDP segment with it. The time at which the packet is sent is stored for later use. Because the destination does not listen on the port, it sends an ICMP port unreachable that eventually triggers the call of the `error_event` XRL in our process. The error indicates on which socket the error arrives and the nature of the error. The delay is thus simply computed by doing $now - measure$ where now is the time at which the XRL is called and $measure$ is the time at which the probe was sent.

The module needs to keep some state about the <source,destination> IP pairs it measures. To do so, different datastructure are required. First, the `_destinations` map maintains measurement information for each <source, destination> IP pair. This information contains the interval at which the pair must be measured and the list of the measured delays for the pair (the closer to the end of the list, the more recent the measurement). Once a delay has been measured for a pair, it is appended to its measured delay list. When the `get_measurements` command is called on the measurement module, this is the measured delay list for the requested pair that is returned. Two other datastructures are used to map a socket identifier to a pair and vice versa. The `_socket_info` maps gives information about the socket indexed by the socket identifier. The related information is the source and destination addresses and the time at which the last segment has been sent

[3]`socket6_user` for IPv6

on this socket (the *measure* variable). The `_sockets` map is the opposite of the `_socket_info`. `_sockets` gives the socket identifier for any pair. The `_socket_info` is unfortunately required as there exists no way in XORP to retrieve meta information on a socket like those we need.

The IP pairs are measured periodically. To implement these periodic probings, we use a XORP periodic timer. Every second, this timer calls the `loop` method of our process. When this method is called, a UDP segment is sent to each <source, destination> IP pair that should have been measured at the latest when the `loop` method is called. To efficiently determine the pairs to measure at the `loop` call, the `_to_measure` priority queue is maintained for each source covered by the measurement module. The key in the priority queue is the time at which the measurement has to be done and the value is the destination address. When a measurement is sent by the `loop`, the entry is removed from the priority queue and the next measurement time is computed for that entry. The new measurement time is then added to the priority queue. Fig. 5.12 shows the pseudo-code of the `loop` method.

Lines 17 – 21 ensure the measurement periodicity of <source, destination> pair that has not been stopped. The salt is used to avoid synchronization of measurements and is a small random value [AKZ99].[4] With Fig. 5.12, we can see that stopping a measurement by calling the `stop_measurement` does not apply immediately and an ultimate probe is sent after such a call. We can also see that there is never more that one entry par <source, destination> pair in the queue which is optimal from a memory point of view.

Fig. 5.13 shows how the ICMP port unreachable is processed by our module.

It is not possible, without changing XORP to associate a time to an event on a socket. This explains why line 0 is required in the algorithm of Fig. 5.13. The retrieval of the time has to be carried out as soon as possible to limit the inaccuracy of the delay estimation.

Fig. 5.14 compares the accuracy of the UDP ping we have implemented in a Measurement module with the standard traceroute UDP ping and the ICMP ping command. Fig. 5.14 plots the average of delay measured by the technique and the 95% confidence interval. For the comparison, we made 1,500 pings for each technique. The interval between two probes is one second. The setup uses two machines directly on the same VLAN. One running IDIPS on Linux, and the other receiving the UDP probes and running on Linux. The UDP ping is performed with the `traceroute` command and the ICMP ping with the `ping` command.

From Fig. 5.14 `UDP ping` and `XORP` labels, we can see that using XORP introduces a bias in the measurement. This bias is mostly due to the process context switching introduced by the XRL based implementation of the XORP sockets. Indeed, when an segment arrives at the host, it is not

[4]In our implementation, the salt is zero.

```
FOREACH src IN _to_measure
DO
     WHILE _to_measure[src] IS NOT EMTPY
     DO
         entry := _to_measure[src].pop

         IF entry.key > NOW
         THEN
            MOVE TO NEXT SOURCE
         END

         dst := entry.address
         socketid := _sockets[src][dst]
         _socket_info[socketid].last_call := NOW

         send_UDP_probe(socketid, src, dst)

         IF (src, dst) NOT STOPPED
         THEN
             entry.key := NOW
                   + _dsts[src][dst].interval
                   + salt
             _to_measure[src].push(entry)
         END
     DONE
DONE
```

Figure 5.12: Measurement module `loop` method pseudo-code

```
SOCKET4_USER_0_1_ERROR_EVENT(socketid, error)
now := NOW
IF error = ICMP_PORT_UNREACHABLE
THEN
     si := _socket_info[socketid]
     measure := si.last_call
     delay := now - measure
     _destinations[si.source][si.destination].measurements.append(delay)
END
```

Figure 5.13: UDP ICMP port unreachable management

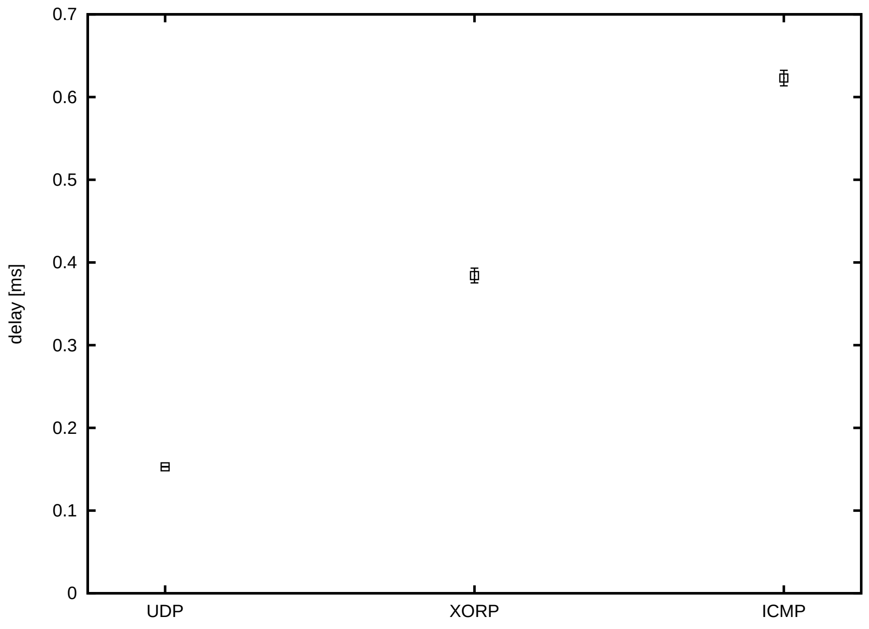


Figure 5.14: Delay precision comparison between standard UDP ping, ICMP ping and XORP UDP ping

delivered immediately to the measurement process. It is first received by the main XORP process that then notifies the Measurement module process that an event occurred on the socket (i.e., the reception of the ICMP port unreachable). This design thus imply process switching in addition to the standard kernel/user space switching with POSIX sockets. In addition, this scheme is also applied when sending the segment meaning that the time between the probe is sent from the Measurement module process and the time the probe is effectively received by the kernel depends on how rapidly the XRL is sent and processed by the main XORP process. However, the bias is of less than 0.3ms in our setup which is acceptable for most of the measurements. More interestingly, the measurement is better with the UDP ping implemented in IDIPS than the standard ICMP ping. Unfortunately, we did not managed to determine the reason why it performs better but the difference could come from the fact that content of the ICMP must be copied while it is not required with the UDP ping.

Prediction module example The Measurement module presented above does delay measurement by the mean of UDP pings. The Prediction module example in this section uses the round-trip-delays measured by the UDP ping measurement module to predict the delay expected for the paths in the near future. The Prediction module simply averages the last round-trip-delays measured for a <source, destination> IP pairs. The average delay is the prediction of the delay for the path defined by the pair.

In this module, a path is defined by a source and a destination IP address. When a `start_prediction` command is received by the prediction module, it requests the UDP ping measurement module to start a measurement for the <source, destination> IP pair that defines the path the delay prediction has to be performed for. The prediction module then periodically retrieves the list of the last measurements for the path. Because the prediction module is the single one to use the UDP ping prediction module, it requests the measurement module to flush its memory. The prediction module then computes the average of the measured delays in the list. This average is considered as the future value of the delay until the next retrieval of the measurements list for the path.

The prediction module maintains two datastructures. On the one hand, the `_paths` map maps a path to a <source, destination> IP pair. On the other hand, the `_delays` map stores the predicted delay for each path.

To speed-up the Querying module processing, the prediction module also pushes the prediction delays to the Querying module path attributes collection. That is, when the Querying module needs the delay prediction, it does not need to request the prediction module. Doing so limits the use of XRLs and thus the number of context switches.

Our example implementation has no other intelligence. Indeed, the list of measurements is retrieved at the same rate for each path (once every 10 seconds thanks to a XORP periodic timer) and the prediction module requests the UDP ping measurement module to send a probe every second. However, it would not be a hard task to modify the module to enable an adaptive measurement rate and an adaptive measurements list retrieval.

5.3.3 High Level Cost Functions Implementation

In this section, we show how to construct simple fundamental cost functions and how to combine them to implement an ISP policy. Our example is based on a situation in which an ISP has three customer families: (*i*) *premium users* always requiring the best available performances, (*ii*) *standard users* requiring a good performance/cost trade off, and (*iii*) *light users* always requiring the lowest cost. The traffic engineering changes between the night and the day for standard users: during the day, a lower cost is preferred while during the night, the performance is preferred. The monetary cost of a path depends on the 95th percentile load of the link used to reach the Internet.

In our example, we assume that the prediction module feeds the querying module with the following information:

- routing reachability of the paths. A path is reachable if there exists a

Algorithm 1 Example of cost function for the reachability

```
Ensure: Integer value representing the result of this Cost Function.
1: procedure IS_REACHABLE_CF(src, dst)
2:     reachable ← get_attribute(<src,dst>, REACHABILITY)
3:     return reachable
4: end procedure
```

Algorithm 2 Example of cost function for the path locality

```
Ensure: Integer value representing the result of this Cost Function.
1: procedure LOCALITY_CF(src, dst)
2:     origin ← get_attribute(<src,dst>, ORIGIN)
3:     if origin = LOCAL_ASN then
4:         return 0
5:     end if
6:     return 1
7: end procedure
```

route in the FIB to forward traffic from its source to its destination, this information is stored in the `REACHABILITY` attribute

- originating ASN. The originating Autonomous System Number of a path is the originating AS number of the prefix of the destination as discovered by BGP. This information is stored in the `ORIGIN` attribute
- monetary cost of the paths. The monetary cost of a path is the expected cost it would represent to carrying one additional Mega bit per second of traffic on it. This cost is computed by applying the 95th percentile technique [DHKS09] and is stored in the `COST` attribute
- available bandwidth of the paths. The available bandwidth of each path is estimated and is expressed in Mbps stored in the `ABW` attribute
- customer family. A customer can be premium, standard or light user. The customer family, stored in the `FAMILY` attribute, of a path is determined simply by considering the source of the path and ignoring its destination

We first have to define if a destination is reachable or not from a given source address. A path, defined by a <source, destination> pair, has its `REACHABILITY` attribute equal to 1 if it is reachable. Otherwise, the attribute is set to the maximum integer value. The cost function `is_reachable_cf`, implemented in Algorithm 1, thus makes reachable destinations more preferable than unreachable ones.

The locality of a path is determined by the originating AS number of the path destination. If the destination prefix is originated by the operator,

Algorithm 3 Example of cost function for the cost minimization

```
Ensure: Integer value representing the cost of using the path defined by src, dst.
1: procedure MINIMIZE_COST_CF(src, dst)
2:     cost ← get_attribute(<src,dst>, COST)
3:     return cost
4: end procedure
```

Algorithm 4 Example of available bandwidth cost function

```
Ensure: Integer value representing the result of this Cost Function.
1: MAX_BW the highest theoretical available bandwidth in the network
2: procedure AVAILABLE_BW_CF(src, dst)
3:     abw ← get_attribute(<src,dst>, ABW)
4:     return (MAX_BW – abw)
5: end procedure
```

Algorithm 5 Example of customer family cost function

```
Ensure: Integer value representing the customer family for traffic from src to dst.
1: procedure CUSTOMER_FAMILY_CF(src, dst)
2:     family ← get_attribute(<src,dst>, FAMILY)
3:     return family
4: end procedure
```

the path is considered local. Algorithm 2 shows how to implement the `locality_cf` cost function that prefers local paths over non-local ones. In this function, `LOCAL_ASN` is operator AS number.

Algorithm 3 shows the `minimize_cost_cf` cost function that returns the monetary cost of using a path. This function makes path with the lowest monetary cost more attractive. To avoid oscillations, it is a good idea to use classes of monetary costs instead of the exact monetary cost. For example, the `COST` attribute could be the reminder of the division of the monetary cost by x instead of being the raw value of the monetary cost.

When considering bandwidth, the best paths are those having the highest available bandwidth. The implementation of a cost function preferring paths with the highest bandwidth is not straightforward. Indeed, IDIPS, by definition, always prefers the lowest cost while in terms of bandwidth, the highest is the best. Thus, to prefer the paths with the highest bandwidth, the value of the available bandwidth is subtracted to the highest theoretical available bandwidth for the operator (i.e., the total network capacity). Algorithm 4 provides the implementation of such a cost function, `MAX_BW` being the highest theoretical available bandwidth in the network.

As for cost minimization, the customer family cost function only has to

Algorithm 6 Example of a complex cost function

```
Ensure: Encounters customers requirements
 1: PREMIUM_USER = 1
 2: STANDARD_USER = 10
 3: LIGHT_USER = 20
 4: procedure CUSTOMER_MANAGEMENT_CF(src, dst)
 5:     if (is_reachable_cf (src, dst) = MAX_INTEGER) then
 6:         return (UNREACHABLE)
 7:     end if
 8:     customer ← CUSTOMER_FAMILY_CF(src, dst)
 9:     if (customer = PREMIUM_USER) then
10:         cost ← AVAILABLE_BW_CF(src, dst)
11:     end if
12:     if ((customer = STANDARD_USER ∧ DAY) ∨ customer =
            LIGHT_USER) then
13:         cost ← MINIMIZE_COST_CF(src, dst)
14:     end if
15:     if (customer = STANDARD_USER ∧ NIGHT) then
16:         cost ← AVAILABLE_BW_CF(src, dst)
17:     end if
18:     return ( LOCALITY_CF(src, dst) · cost ) + cost
19: end procedure
```

return the customer family. Algorithm 5 shows the implementation of this cost function. In the system, the family 1 corresponds to premium users, 10 is for standard users and 20 for light users.

The previous algorithms can be combined by the network operator to build more complex policies. Algorithm 6 combines all the blocks in order to reflect the operator policies proposed earlier in this section. In particular, Algorithm 6 first checks whether the destination `dst` is reachable from the source `src`. If the path is reachable, it applies the policies previously defined, based on the `FAMILY` attribute. For *premium* clients available bandwidth is always preferred. For *standard* clients the applied policy depends on the time period; the available bandwidth is used as cost function during the night, while cost minimization is preferred during the day.

The last line gives preference to a local paths. This line is an example of weighted sum of cost functions. More particularly, the cost result by the `CUSTOMER_MANAGEMENT_CF` is a weighted sum of the costs from other cost functions, weight by the cost returned by a cost function. The principle in the example is to double the cost if the path is not local.

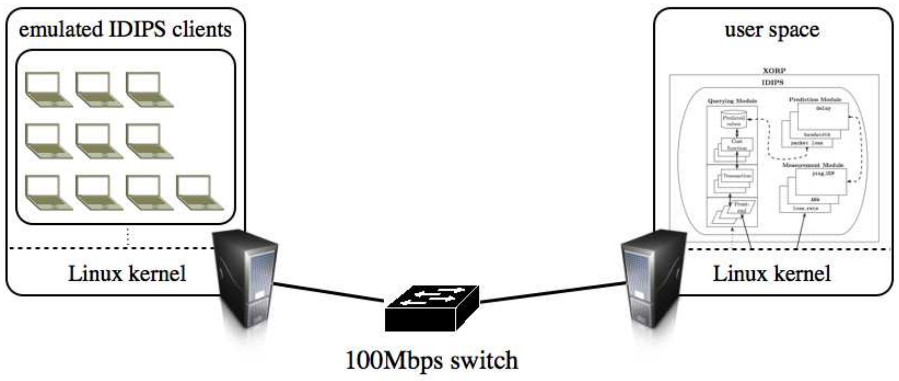


Figure 5.15: Evaluation testbed

5.4 Evaluation

In this section, we evaluate the performance of our Idips implementation. We first present the methodology we follow and the testbed we build (Sec. 6.4.1). The testbed is based on the Xorp implementation of Idips we discussed in Sec. 5.3.2. We next discuss the results (Sec. 5.4.2). In particular, we focus on the heart of Idips, i.e., the ranking process.

5.4.1 Methodology

Fig. 5.15 provides a description of the testbed we build for evaluating Idips. The server on which Idips is running is a quad-core Xeon E5430 at 2.66GHz. The server has a 6MB processor cache and has 4GB of RAM. The server runs a Linux 2.6.31-22-server 64 bits distribution. The predicted values storage is populated with 1,500,000 paths uniformly randomly generated (both source and destination addresses are randomly generated). Each path is assigned a uniformly randomly generated delay in [0, 1000ms]. The cost function implemented is the `minimize_delay_cf(src, dst)`.

The client is running a dual-core Xeon 3060 at 2.40GHz, with 5GB of RAM and a Linux 2.6.32-5 32 bits distribution. If there is a single hardware representing the client, the machine ensures that 100 instances of the client are running in parallel. Each client instance sends a request to Idips, each request containing a source address and a list of destination addresses. Those destinations are randomly generated but we ensure that roughly 10% of the randomly generated destinations are not present in the paths stored by Idips. Each instance of the client is in charged of sending 10,000 requests to Idips, meaning that a total of 1,000,000 requests are sent to Idips. We consider the following values for the number of destinations in a request: 1, 2, 5, 10, 50 and 100.

The clients and the IDIPS server are are attached with a 100Mbps switch.

Each experiment in Sec. 5.4.2 is repeated ten times. Each data point represents the mean value over ten runs of the experiment, the clients and the server process being rebooted before each run. We determine 95th confidence intervals for the mean based, since the sample size is relatively small, on the Student t distribution. These intervals are typically, though not in all cases, too tight to appear on the plots.

5.4.2 Results

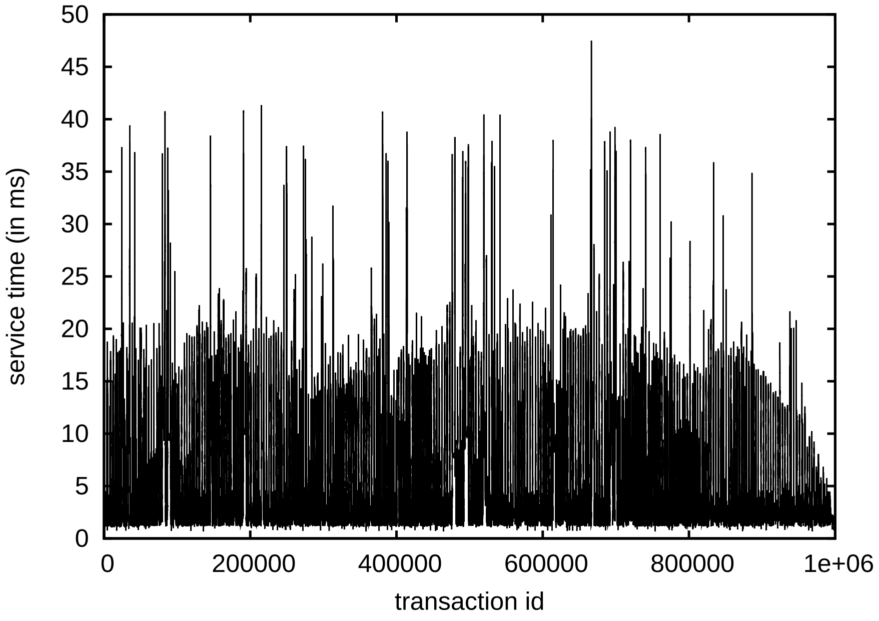
Fig. 5.16 shows the IDIPS service time from the client perspective. In particular, Fig. 5.16(a) gives the time (in ms) between each request and the associated reply for a particular run when the client request contains ten destinations. Fig. 5.16(a) shows how IDIPS is stable over transactions, no drift is observed.

Fig. 5.16(b) shows the IDIPS time distribution as quantiles. The dotted line represents the median value, while the box plot gives the minimum and 95th percentile values as well as the 25th and 75th percentiles.

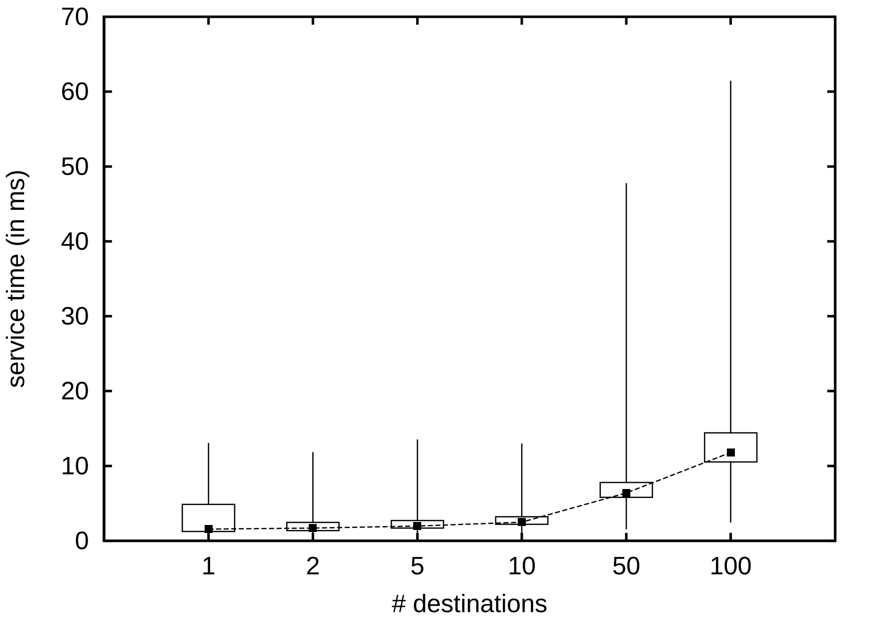
Obviously, the IDIPS service time increases with the number of paths (i.e., the number of destinations in this example) induced by the client requests. The service time linearly increases with the number of paths. The linear dependency is the result of the conversion of the list received from the client in text into a binary format into the querying module implementation, the construction of the possible paths and the cost computation for each of such paths. The cost function having a temporal complexity of $\mathcal{O}(1)$, the total complexity is $\mathcal{O}(s * d) = \mathcal{O}(n)$ where s is the number of sources in the request, d, the number of destinations and n the number of paths. In our experiment, $s = 1$ making $n = d$. In addition, the bigger the number of destinations, the less stable the service time as suggested by the service time distribution amplitude. The higher dispersion observed for the list of one destination can be explained by the overhead caused by the switching between the XORP processes (i.e., the finder and the querying module).

Fig. 5.17 breaks the IDIPS service time down into three categories: the network delay (labeled "network" - descending line pattern portion of the stacked bars), the path ranking (labeled "IDIPS" - ascending dashed line pattern portion of the stacked bars), and the internal XORP processing (labeled "XORP" - descending dashed line pattern portion of the stacked bars). For plotting those results, we consider the median value among the ten runs. Instead of plotting the median value of the service time, we rather consider the time proportion of each category.

The time consumed by the network is negligible. This is due to clients and server, in our testbed, are separated by a single switch. However, as the IDIPS server is supposed to be deployed within a campus or an ISP network (in the fashion of DNS service), one can imagine that the required network



(a) service time stability - destinations=10



(b) service time distribution

Figure 5.16: IDIPS service time as perceived by the client

time (i.e., time spend in the network between the client and the server and

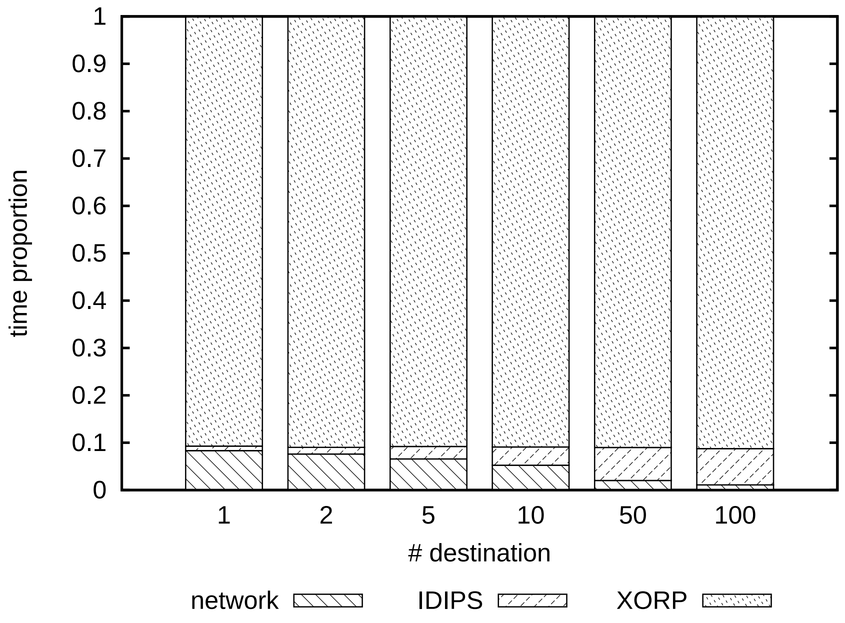


Figure 5.17: Proportion of the service time split - median over the ten runs

vice-versa) would be very close to what we experienced in our testbed.

In general, the time spent in the whole ranking process in IDIPS increases linearly with the number of paths from the requests. With about only 12% to 20% of the time spent building the list to return to the client once the costs are computed. The rest being spent by the cost computation and attribute retrieval. The internal XORP processing represents most of the service time (around 90%) and is also linearly dependent with the number of paths from the requests. The time spent directly in the XORP internals is mostly due because of the marshaling and unmarshaling of the XRLs and the context switching between the `finder` and the querying module (remember that the XRLs are always processed by the `finder`.

Fig. 5.18 shows the load on IDIPS in term of requests/second number. Obviously, the capacity of IDIPS to process requests decreases with the request size. It is a normal behavior as large requests require more processing time, in terms of IDIPS (typically more cost function to evaluate and, thus, more lookup into the predicted values storage) and internal XORP processing (as already suggested by Fig. 5.17).

We also notice that, in the worst case (i.e., 100 destinations per request), IDIPS can still process more request per second than what could be required for peer-to-peer applications [GCX$^+$05].

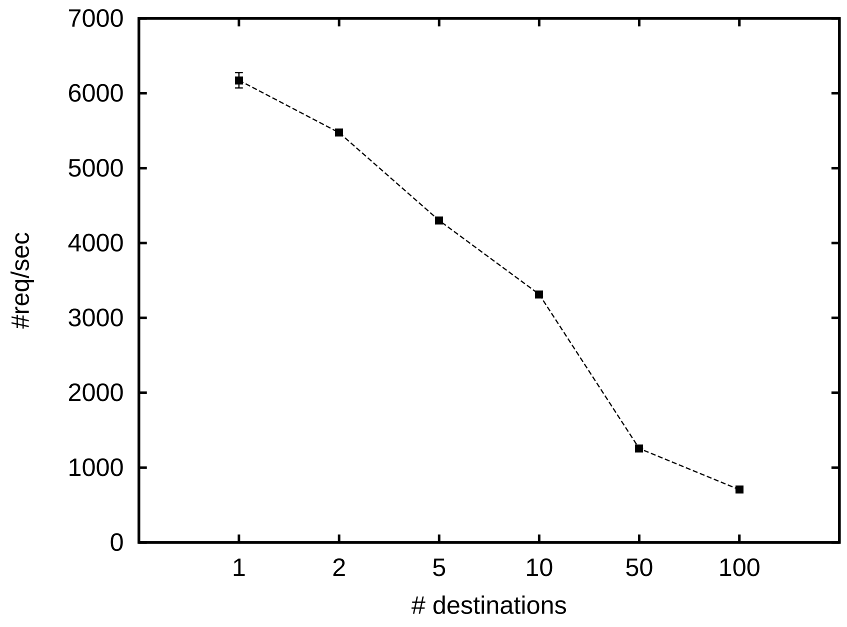


Figure 5.18: Load on IDIPS without XRL

5.5 Related Work

In 2007, when we started to design IDIPS P2P optimization was not an objective [SDIB08]. However, IDIPS can help to select the best peers in P2P systems. Despite the fact that we do not use IDIPS for P2P traffic optimization, most of the IDIPS related work are in that field.

Bindal et al. show by simulation that biasing the peer choice in P2P can improve the overall performance [BCC$^+$06]. On the one hand, download times can be reduced and on the other hand ISPs can reduce their operational costs. This simulation based study shows that an informed peer selection gives better results than an unbiased peer selection when using Bittorrent [PGES05].

Aggarwal et al. [AFS07] propose an *oracle* service very similar to IDIPS that would be configured by the network operator and queried by P2P applications [AFS07]. The underlying assumption in oracle is that traffic would be optimized by letting ISPs and P2P clients collaborate. The oracle is a server operated by the network operator. When a P2P client needs to select peers in a swarm, it sends the list of peer's IP addresses to the oracle. The oracle returns these addresses labeled with a rank. The way the rank is computed by the oracle is hidden to the clients. The paper shows that preferring peers within the local AS increases the overall performance and should reduce the operational costs for the ISP as the traffic tends to remains local. [AFS07] characterizes performance metrics to evaluate the benefits of using a biased peer selection like oracle. While the oracle limits the ranking to destination addresses, IDIPS proposes to rank paths or even groups of paths, using prefixes. Another difference between the oracle and IDIPS is the ranking scope: the oracle ranking is limited to local or peering domains while such a limitation does not exists in IDIPS.

Choffnes and Bustamante [CB08], as opposed to the oracle approach [AFS07], propose a biased peer selection in P2P without the need of a specific infrastructure. Instead, Choffnes and Bustamante propose *Ono* that uses the information that are already available within commercial CDNs. In general, CDNs like Akamai [5] and Limelight [6] control the traffic from the clients by using the DNS. To be simple, depending on the client location, the DNS replies for a name are different. By periodically resolving names for CDNs servers, the peers can build an abstracted representation of the relations they have with these CDNs, this representation is called the *behavior map*. Each Ono-enabled peer resolves the name of the same 6 well chosen CDN names. The corresponding C-class prefix is considered instead of the addresses. Indeed CDNs often have several servers in the same data-center and the servers are grouped in C-class prefixes. The addresses belonging to the same C-class prefix can be safely considered as belonging to the same datacenter. The peers do the resolution several times to determine the dynamic of the change in the CDN name map. Two peers "close" to one name will have the same dynamics in the CDN map. In addition, to the change ratio of the CDN names in the map, the peers ping the returned addresses. The measured delay is used to weight the map according to the delay with the servers. When a peer must select another peer, it will chose a peer that presents a similar weight ratio map. The RTT weighting relies on the principle that the closer two peers are, the more likely the name will be resolved in the same /24 subnet. and the delay to the server for a given name will be close as well. The deployment of Ono on hundred of thousands nodes shows that doing so can reduce the average delay and increase the download rate by about 30%. It also shows that in 33% of the cases, the chosen peers were within the local ISP. The major difference between IDIPS and Ono is that no specific infrastructure must be deployed for the service to work. However, the reduction of inter-AS cost is only a side effect of the optimization of the perceived client performance. Ono does not leverage an ISP P2P collaboration.

Xie et al. propose *P4P* [XYK+08]. P4P provides topology hints to P2P clients. In P4P, the topology and policies are abstracted into the p-distance. The p-distance summarizes the metrics that are relevant for the operator and is used by ISPs to communicate their preferences and status for the

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

[6] `http://www.limelightnetworks.com`

traffic. The application uses the p-distance to create an abstracted view of the network connectivity. The applications can then select their destinations based on the minimal p-distance in the abstracted network view. ISPs, or third parties, maintain iTrackers to communicate the p-values. These iTrackers are contacted by P2P clients to compute the set of the best swarm peers. The iTrackers are used to retrieve the policy information, the p-values but also the network capabilities.

During the mid 2000ies, network operator started to complain about the operational cost of supporting P2P traffic [Lig, She, KTCI04]. Indeed, the P2P overlays do not take the underlay network topology into account and neighbors are in general selected at random. Therefore, it is possible that a packet sent by a peer in an AS A to a peer in the same AS follows a path that crosses several times the network boundaries, possibly via provider links. Providers have then started to block P2P traffic or to limit the bandwidth for P2P flows. However, P2P application now rely on encryption or simply behave like HTTP to masquerade their traffic. The problems faced by the ISPs is because ISPs are black boxes and it is hard for a client application to know the peers that are efficient but not harmful for its ISP. Industry and academy started to collaborate on a solution for this problem with an IETF P2P Workshop meeting specially hold in May 2008 in Boston.[7] During this workshop researchers and operators were invited to present position papers about the problem and the solutions they though to be good for this. The conclusion of the workshop is that P2P and ISPs must collaborate somehow. Some proposed to add caches and install the trackers inside the ASes, but this would have caused legal issues as it is known that P2P traffic is mostly used for illegal content. As a result, the *Application-Layer Traffic Optimization* (ALTO) [SB09] Working Group as been set up at the IETF. ALTO aims at designing and specifying services helping applications such as peer-to-peer, content delivery networks, and mirror selection, to select the best peers. Factors of interest in this selection are, among others, maximum bandwidth, minimum cross-domain traffic, lowest cost to the user, etc. ALTO is a mix of the best features of P4P, oracle and IDIPS with additional contributions. Clearly, IDIPS is in line with the ALTO Working Group thematic. However, while we have been actively working in the ALTO working group at its beginning [AFP+09] to make sure that the protocol would be able to support IP prefixes, names or AS numbers, we have chosen not to implement the ALTO protocol in IDIPS because many features are P2P related and because ALTO is closer to P4P than to oracle or IDIPS. Nevertheless, IDIPS could be adapted to implement the ALTO service as it meets most of the ALTO requirements [KPS+11].

As opposed to oracle, Ono, P4P and ALTO in general, IDIPS is not designed specifically for P2P. IDIPS is an extension of the *NAROS Name,*

[7]`http://www.funchords.com/p2pi/`

Address and ROute System (NAROS) server that has been designed for multihoming IPv6 host-centric traffic engineering in the early 2000 [dLBL03]. NAROS was proposed to select the best source address in multihomed IPv6 sites. A NAROS client sends a request to a NAROS server with the destination address it aims to use and the list of its source addresses. The server returns the best source address to use for the destination. The prefix of the destination is also returned to avoid a client requesting several times the server for destinations within the same prefix and that would thus have the same result. The major difference between IDIPS and NAROS is that IDIPS return all the possible aggregated <source,destination> pairs and ranks them. This change allows IDIPS to be used equally for incoming and outgoing traffic engineering and enables the use of multi-path routing or transport.

A proposal that shares objectives similar to IDIPS is *Morpheus* [WAR07], which determines the best path to use according to the operator policies and, then, sends BGP updates to its BGP router target (via multihop eBGP). Like IDIPS, Morpheus is very modular but is restricted to BGP as the signaling is performed using BGP messages while IDIPS has its own messaging format, allowing a finer-grained interaction between the client and the path selection service.

Finally, a number of vendors have proposed proprietary path selection solution ([Int05, Ava05, Rad, Cis]). These solutions all follow the same principle. Specialized boxes are deployed in the network and monitor it actively, passively or both. The measurements are combined with policies to determine the quality of the different routes. Based on the observations and the configuration, the boxes can inject prefixes into BGP to influence the incoming traffic but they can also modify link costs in the IGP or inject partial BGP tables into it to control the outgoing traffic as well. Finally, NAT can be used to ensure that some flows enter the network via a given link.

5.6 Conclusion

The Internet is evolving. During this last decade, we have seen the emergence of applications having more and more requirements in terms of delay, jitter,or bandwidth. In addition, the single path assumption between a source and a destination does not hold anymore. As a consequence, the applications can use several paths to retrieve their content. It might thus be interesting to provide those applications a service for selecting their paths better than randomly, i.e., a service for selecting paths that meet applications requirements.

In this chapter, we proposed an Informed Path Selection Service (IDIPS), that is able to rank paths. IDIPS is a generic, scalable, lightweight and easily deployable solution allowing ISPs, enterprises, or campus networks to qualify paths between a source and a set of destinations. IDIPS makes use of passive and active measurements to keep track of the network conditions.

In this chapter, we have shown that IDIPS can be used to enable performance based incoming traffic engineering with LISP.

We discussed our IDIPS implementation inside XORP and focused on simple cost function (i.e., the way IDIPS assigns a cost to a path) construction and how to combine them to reflect more complex ranking strategies. We built a testbed and evaluated the performance of IDIPS. In particular, we focused on the heart of IDIPS, the ranking process. We demonstrated that IDIPS is robust as it is able to process a large number of requests/second while providing a stable response time to the client.

Chapter 6

Measurement Reduction with Clustering

6.1 Introduction

Using measurements collected at network vantage points to infer the Internet conditions is an important component of performance aware services like IDIPS. However, constantly probing the network leads to scalability issues. Indeed, probes injected in the network might burden the traffic. Further, if those probes come from multiple vantage points, they might be considered by some as a kind of distributed denial-of-service attack. In addition, because of memory and processing constraint, it is not possible to constantly measure all the network. As previously mentioned by Cheswick et al., any networking measurement system must be engineered very carefully [CBB00].

There exist several ways for reducing the amount of required measurements and resource consumption. A first possibility is to allow collaboration between probing monitors and to cluster probe targets. *Clustering* means aggregating a subset of targets into the same hat and considering a measurements towards one of the target as being representative of the whole cluster [Bro04, SPPVS08, KW00]. Another way is to modify the probing technique so that it consumes fewer resources.

In this chapter, we investigate the first solution (i.e., collaboration and clustering). We explain how measurement reduction can be achieved through collaboration between vantage points and destination clustering. We also discuss several metrics that can be used to evaluate the performance of a cluster based measurement campaign. We further explain that a greater measurement reduction can be achieved if collaboration between measurement sources is added to clustering.

In addition, we discuss five clustering techniques that can be used to reduce the measurements impact. These clustering techniques are based on

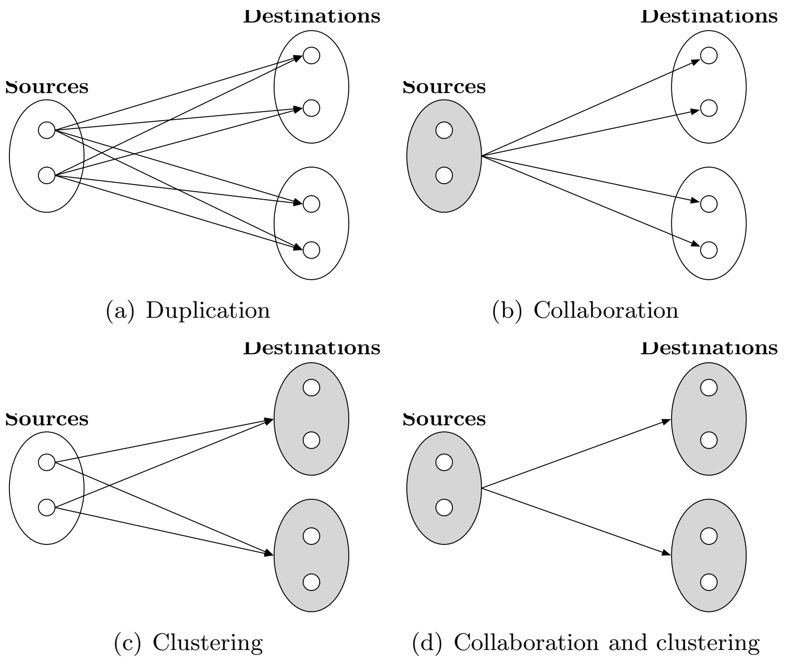


Figure 6.1: Illustration of measurements duplication and reduction

available network information.

Based on real data collected, we evaluate these clustering techniques using the metrics we propose. We show that they are reasonably accurate while allowing a strong reduction in the amount of required probes.

The remainder of this chapter is organized as follows: Sec. 6.2 provides a theoretical background for measurement reduction through collaboration and clustering; Sec. 6.3 discusses five clustering techniques and positions them regarding the state of the art; Sec. 6.4 evaluates these clustering techniques and shows how clustering can reduce the measurement overhead; Finally, Sec. 6.5 concludes this chapter and discusses future directions.

The contribution of this chapter is to provide a technique that is a trade-off between the number of measurements and the quality of the measured value. To do so, we propose a new lightweight clustering technique that respects the topology to group probing sources and measurement destinations.

6.2 Theoretical Background

6.2.1 Measurement reduction

Active Internet measurements reduction is a strategic issue when large-scale measurements are required. Up to now, several solutions have been proposed for delay measurements [NZ02, NZ04, DCKM04], Internet topology discovery [DRFC05, DFC05], and bandwidth estimation [HS05, RMK$^+$08]. However, these techniques, despite their strong advantages in term of measurement reduction, are complex to deploy and can require to modify the measurement technique itself.

If keeping the measurement mechanisms intact is a requirement when trying to reduce the probing impact, two solutions are imaginable: reducing the number of measurement vantage points (i.e., the *sources*) and reducing the number of measurement targets (i.e., the *destinations*). If both lead to a measurement reduction, they can also lead to a lower accuracy. A balance must thus be found between measurement reduction and accuracy.

These two solutions might be implemented through *collaboration* and *clustering*. Collaboration can help to reduce the number of sources involved in measurements while clustering is useful to reduce the number of destinations to probe.

Any two nodes a and b could collaborate if they are topologically close. For instance, if they both belong to the same campus network. Indeed, if they share the same first hops, when measuring a destination d, it is very likely that probe packets will follow, for both a and b, the same path. Or at least, they will share large path segments. Consequently, the resulting measurement will be very close and if both nodes act in isolation from each other, they unnecessarily duplicate their efforts, as illustrated in Fig. 6.1(a). On the contrary, if a is aware of the measurements already performed by b to d, then, a no longer needs to measure d. In other words, collaboration avoids duplication of measurements and thus reduces the measurement overhead. Collaboration is illustrated in Fig. 6.1(b). How measurement sources can collaborate is still an open issue. However, efforts have been made in the context of Internet topology discovery [DRFC05].

On the other hand, the key idea behind clustering is to aggregate a set of nodes under the same hat and consider that all nodes within a given hat share the same properties. A *Cluster* $\mathcal{C}$ is defined as a set of nodes sharing the same properties. Clustering is illustrated in Fig. 6.1(c).

The basic assumption behind clustering is that all the nodes that belong to a cluster share the same path performances (i.e., delay, bandwidth, etc). Consequently, measuring a single point within a cluster would be sufficient. The node that is measured to estimate the cluster path performances is called the *Reference Point*. It is thus worth to notice that clustering allows to reduce the number of measurements as only the reference point has to be measured.

In the remainder of this chapter, we only consider one reference point per cluster. However, it is conceivable to have more than one reference point in some clusters. Studying the impact of varying the number of reference point per cluster is let for further investigations.

A cluster is said *popular* if any destination within this cluster is often measured. We evaluate the popularity $\pi_{\mathcal{C}}$ of a cluster $\mathcal{C}$ by counting the number of sources sending probes towards $\mathcal{C}$.

We can bring together collaboration and clustering by defining $\mathcal{S}_{\mathcal{C}}$, the *Source Cluster* of $\mathcal{C}$. $\mathcal{S}_{\mathcal{C}}$ represents the set of sources measuring a cluster $\mathcal{C}$. Remember that this makes sense only if all nodes in $\mathcal{S}_{\mathcal{C}}$ are topologically close. Collaboration and clustering together allow a greater reduction in probing effort, as depicted in Fig. 6.1(d). It is worth to notice that the set of collaborating nodes is a cluster itself.

We propose a metric for evaluating the measurement reduction: the *Measurement Reduction Factor*. The Measurement Reduction Factor ρ for clustering technique t on $\mathcal{P}$ is:

$$\rho = \frac{|\mathcal{P}| - \sum\limits_{\mathcal{C}_i \in \mathcal{C}_{\mathcal{P}}} |\mathcal{R}_{\mathcal{C}_i}|}{|\mathcal{P}|} \tag{6.1}$$

where $\mathcal{P}$ is the set of $< s, d >$ nodes pairs such that s, the *source*, has to measure d, the *destination*. $\mathcal{C}_{\mathcal{P}}$ is the set of all the clusters on $\mathcal{P}$ assuming clustering technique t. $\mathcal{R}_{\mathcal{C}}$ is the set of all the reference points of $\mathcal{C}$.

Positive values for ρ means that the measurement reduction technique used effectively reduces the number of measurements. On the contrary, a negative value means that more measurements have to be performed than without reduction. For instance, if $\rho = 0.5$, the number of measurements is reduced by 50%.

Finally, note that measurement reduction is achieved on $\mathcal{P}$ if it exists a cluster $\mathcal{C}$ such that $|\mathcal{S}_{\mathcal{C}}| > 1$ or $|\mathcal{C}| > 1$ or both.

6.2.2 Clustering Accuracy

In a cluster $\mathcal{C}$, only the reference point is used to estimate the performance of all the destinations within $\mathcal{C}$. It would be a matter of concern if measuring a single point (or a few points) within a cluster would lead to a strong measurement accuracy loss. This section defines how to estimate the accuracy of clustering techniques.

Estimation Error For a path performance metric m (delay, bandwidth, etc), the estimation error between node i and node j is defined as:

$$e_{ij} = \frac{|\mathtt{m}_{ij} - \widehat{\mathtt{m}}_{ij}|}{\mathtt{m}_{ij}} \tag{6.2}$$

where $\mathtt{m}_{ij}$ is the measured value for m between i and j and $\widehat{\mathtt{m}}_{ij}$ its estimated value.

The estimation error gives the error proportion if the performance of a node is based on the estimated value instead of the directly measured value. The estimated error is expressed in percentage. The closer to zero the estimated error, the more accurate the measurement estimation. Thus, a value of zero means that the estimation is perfect, while, for instance, a value of 0.5 means that there is a difference of 50% between the estimated and the actual values.

For any clustering technique t (see Sec. 6.3), the estimated metric $\widehat{\mathtt{m}}_{ij}$ from a node i to a node j in a cluster $\mathcal{C}$ is the metric associated to all nodes within the cluster.

The measurement error shows how the estimation fits with the reality for a given node in a cluster. However, it can be interesting to characterize the error for the whole cluster and not only a particular node within this cluster. For such an information, statistical tools like mean, percentiles or standard deviation can be applied on the set of all cluster estimation errors.

6.3 Clustering Techniques

In this section, we discuss five clustering techniques. These techniques offer the strong advantage of being very easy to setup as they only require simple information already, or easily, available in the network. With these techniques, clusters are a set of IP prefixes such that a simple longest prefix matching is enough to determine to which cluster a given IP address belongs.

AS Clustering: clusters are defined based on the Autonomous System (AS) membership of Internet hosts (e.g., all the nodes of AS 2611 are put in the same cluster). This is somewhat equivalent to the notion of *super-cluster* introduced by Krishnamurthy and Wang [KW01] for modeling the Internet topology.

Geographic Clustering: clusters are defined based on the geographic localization of Internet hosts (e.g., all the nodes near Paris are within the same cluster).

n-agnostic Clustering: clusters are defined as fixed-length IP prefixes. All the nodes sharing the same n bits prefix are put in the same cluster. The `/24` division proposed by Szymaniak et al. [SPPVS08] is a particular case of such a technique (i.e., 24-agnostic clustering).

BGP Clustering: clusters are built according to BGP. Every advertised BGP prefix refers to a cluster. Each IP address belongs to the cluster with the longest matched BGP prefix. The clustering in BGP prefixes was first proposed by Krishnamurthy and Wang [KW00].

n-hybrid Clustering: clusters are built according to BGP prefixes but the minimum length of the prefixes is fixed to n. When the BGP prefix length is lower than n bits, it is divided into as many `/n` as needed instead of the single prefix length advertised in BGP. n-hybrid Clustering is thus a mix between BGP and n-agnostic Clustering. To the best of our knowledge, we are the first to propose this technique.

Except for n-agnostic Clustering, all these clustering techniques rely on a dataset defining to which cluster an IP address belongs. For Geographical Clustering, the dataset must contain a correspondence between IP addresses and a geographical position. We believe that the MaxMind database [Max02] is accurate enough for most applications, even if it has been shown that it is less accurate (in term of geolocation) than active probing [SGU08, PUK$^+$11]. In the case of AS Clustering, the dataset consists of a mapping between an Autonomous System Number (ASN) and an IP prefix. BGP feed, iPlane [MIP$^+$06] or Cymru [Tea04] provide this information. In both BGP and n-hybrid Clustering, IP addresses are mapped to their longest matched BGP prefix. To know the advertised BGP prefixes, a BGP feed is required. It can be obtained directly through a BGP session or a looking glass (e.g., RIPE). It is worth to notice that BGP Clustering is equivalent to 0-hybrid Clustering and that, in 32-agnostic, clusters have a cardinality of one for IPv4.

We propose n-hybrid Clustering to avoid very large prefixes announced by BGP (e.g., some `/8`). Indeed, some of them may concern hosts spread all around the world and, thus, lead to a very low measurement accuracy.

Clustering techniques for reducing the amount of required measurements have already been extensively studied by the research community. Krishnamurthy and Wang introduce the BGP Clustering in the context of web caching [KW00]. In addition, they propose an adaptive clustering for addresses not classified with BGP. Unfortunately, their technique is based on reverse DNS and traceroute which is not suitable as traceroute is intrusive. Instead, we map undefined IP addresses to an agnostic `/n` prefix. Szymaniak et al. show that latencies are globally equivalent within the same /24 prefix on the Internet [SPPVS08]. Based on that, Szymaniak et al. suggest to use a /24 cluster division (equivalent to what we call 24-agnostic Clustering in this chapter) to reduce the amount of required measurement. Finally, Brown proposes to use clustering for optimizing traffic from a local network to a set of IP addresses put into a given cluster [Bro04]. Based on measurements to a set of *scanpoints* (equivalent to our Reference Points), an entity might decide to choose a particular Internet path for traffic towards the cluster.

In addition, others proposed techniques for monitoring networks and, possibly, reducing the amount of required measurements while keeping accurate the measured metrics [YSB$^+$06, LL08, MZPP08, DCKM04, DFC05]. In particular, Donnet et al. [DFC05] impose a limit on the number of traceroute monitors that can probe a given destination. This is done by dividing the traceroute monitors into clusters, each cluster focusing on a particular portion of the traceroute destinations.

6.4 Evaluation

6.4.1 Methodology

Our evaluation relies on two datasets. The first dataset is taken from the CAIDA's *Archipelago* measurement infrastructure [HHA+]. Archipelago collects traceroute and RTT information towards all routed /24. For our study, we consider data collected in August 2008 by two Archipelago monitors: `bcn-es` (Barcelona, Spain) and `san-us` (San Diego, USA). From these dataset, we extract only complete traceroutes, i.e., traceroutes terminating at the destination. For the rest of this chapter, this dataset is named *Archipelago*. For the second dataset, we collected full NetFlow traffic traces on our campus network. The traces were collected at the single 1Gbps Internet connection of the campus. A total of 45.4 TB of outgoing traffic has been monitored. However, in this chapter, we do not consider 0 bytes or 0 packets flows and ignore the traffic the traffic exchanged with the intensive calculus data-center of our network. Thus, after filtering, the outgoing traffic represents 7.45 TB (i.e., 22.27 Mbps on average). The filtered dataset contains 10,084 different source IP addresses and 36,263,710 destination IP addresses for a total of 60,638,413 different layer-3 flows (i.e., the number of different <src, dst> IP address pairs). In the following of the chapter this dataset is named *NetFlow*.

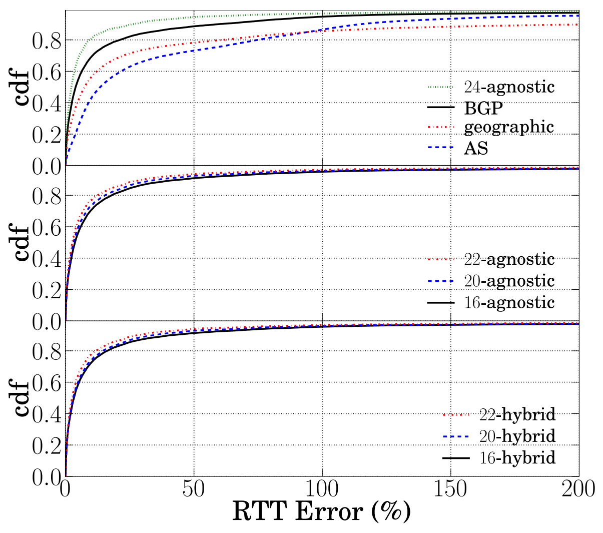
The NetFlow dataset is used to estimate the measurements reduction and the popularity while the Archipelago dataset permits to estimate the impact of clustering on performance estimation accuracy. Unfortunately, we have not found significant datasets that allowed us to estimate measurement accuracy and reduction at the same time. However, measurements from the `bcn-es` monitor is representative of the RTT we see on our campus network. We can thus consider the two dataset as not completely independent (`bcn-es` monitor is connected through the European research network like our campus).

The mapping between IP addresses and announced prefixes was done using the first BGP table dump of August 2008 retrieved from the University of Oregon RouteViews project [Uni]. ASN and IP addresses mapping was done using the iPlane dataset [MIP+06]. Finally, the geographic mapping was achieved using the MaxMind database [Max02].

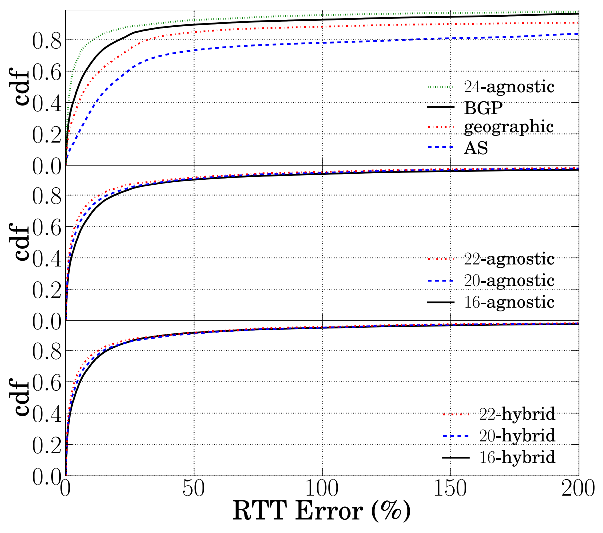
In the following, we limit the n value to $\{16, 20, 22\}$ for n-agnostic and n-hybrid.

6.4.2 Measurement Accuracy

In this section, we evaluate the measurement accuracy of clustering techniques. We focus on two networking metrics: the round-trip time (RTT) and the number of hops. We leave jitter and bandwidth for further work.



(a) `bcn-es`



(b) `san-us`

Figure 6.2: Estimation error applied to RTT

Fig. 6.2 shows the estimation error (see Def. 6.2.2) applied to the RTT for the five clustering techniques introduced in Sec. 6.3. Fig. 6.2(a) focuses on the `bcn-es` monitor and Fig. 6.2(b) on `san-us`. For each figure, there are three plots, the upper one comparing BGP, Geographic, AS, and 24-agnostic Clustering, the middle one n-agnostic, and, finally, the below one n-hybrid. The horizontal axis gives the RTT Error (in %) and the vertical axis the cumulative distribution form (CDF).

A first observation is that n-hybrid and n-agnostic Clustering perform better (n-hybrid being the best) than other clustering techniques, most of the measurements having an error smaller than 50%. More precisely, in 80% of the cases, the RTT Error is less or equal to roughly 25%. The AS Clustering offers the worst performance. This is somewhat expected as some ASes are too large to asses the assumption that any measurement towards the cluster reference point is representative of the entire cluster. Indeed, an AS can be very large and may contain smaller entities that are separately administrated. From Fig. 6.2, we notice that the RTT Error might be very large, i.e., up to 1,000% for AS Clustering (not shown on Fig. 6.2). This is due to inherent limitations of our dataset. Some traceroute might take a long time to reach the destination and the RTT measured at this time might not be really representative of the actual "distance" separating the traceroute monitor and the destination.

Fig. 6.3 shows the estimation error applied to hops. Again, AS and Geographic Clustering provides the largest error. This is expected as they span larger areas. Further, for n-hybrid and n-agnostic, larger the cluster (i.e., smaller the n), larger the hop error. As for RTT error, n-hybrid provides the best performance.

Fig. 6.4 shows the *estimation variation* (the percentiles of all the estimation errors for a given cluster) applied to RTT (Fig. 6.4(a)) and to the number of hops (Fig. 6.4(b)). The horizontal axis gives the various clustering techniques while the horizontal axis, in log-scale, gives the estimation variation. Fig. 6.4 plots stacked bars. The lowest bar (i.e., the darkest) refers to the 25th percentile of the estimation variation. The middle bar shows the 50th percentile (i.e., the median), while the highest bar (i.e., the lighter) gives the 75th percentile.

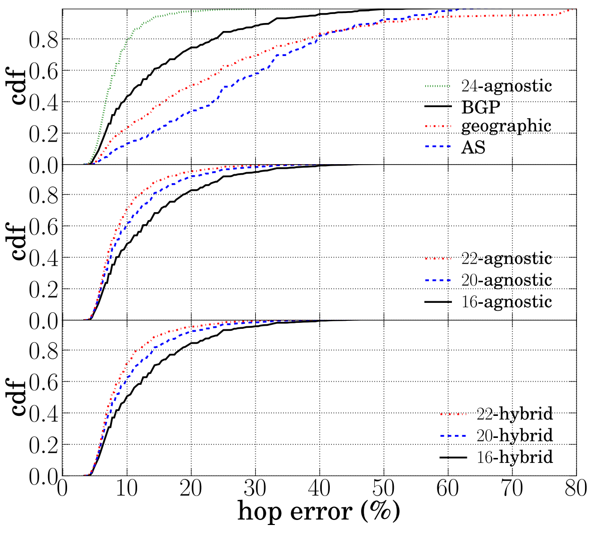
The main lesson learned from Fig. 6.4(a) is that, whatever the clustering technique, the RTT variation is low: the median has a maximum of 1.5% for 16-agnostic (for `san-us`). However, the situation is a little bit different for hop variation (Fig. 6.4(b)). The 75th percentile provides large hop errors, in particular for the largest clusters, i.e., 16-agnostic, 16-hybrid, BGP, AS, and Geographic Clustering.

To summarize, except in a few extreme cases, clustering provides thus pretty good measurement accuracy. In particular, this section suggests that n-hybrid Clustering is the most reasonable choice for a clustering technique deployment.

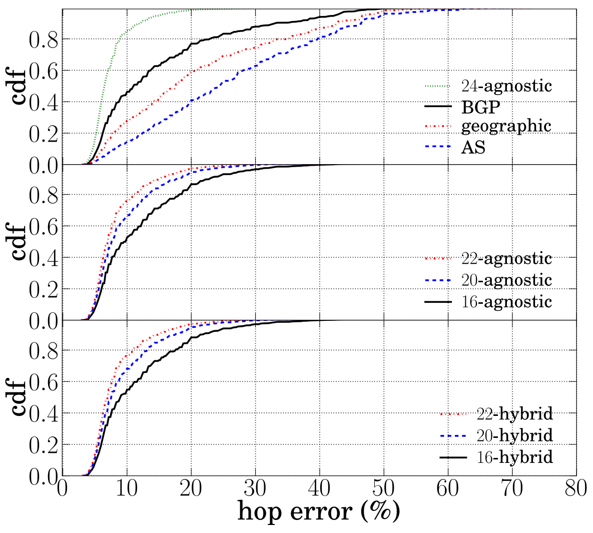
6.4.3 Measurement Reduction

In this section, we evaluate the measurement reduction observed on the NetFlow dataset for the clustering techniques introduced in Sec. 6.3.

We first determine if measurement reduction can be achieved, on the NetFlow dataset thanks to the collaboration. On Fig. 6.5, the horizontal axis, in log-scale, shows the popularity while the vertical axis gives the



(a) `bcn-es`



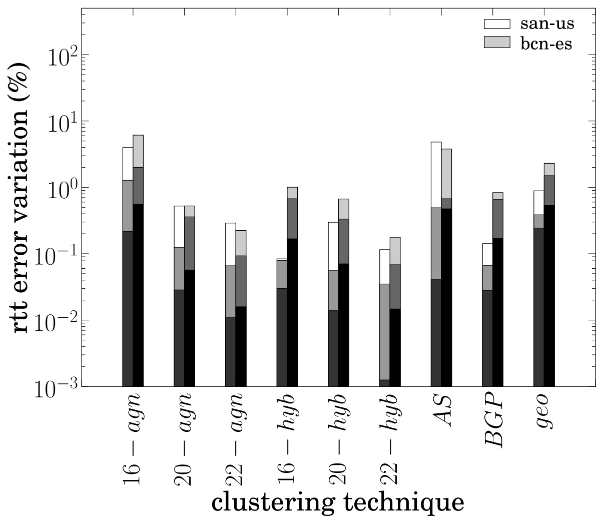
(b) `san-us`

Figure 6.3: Estimation error applied to hop

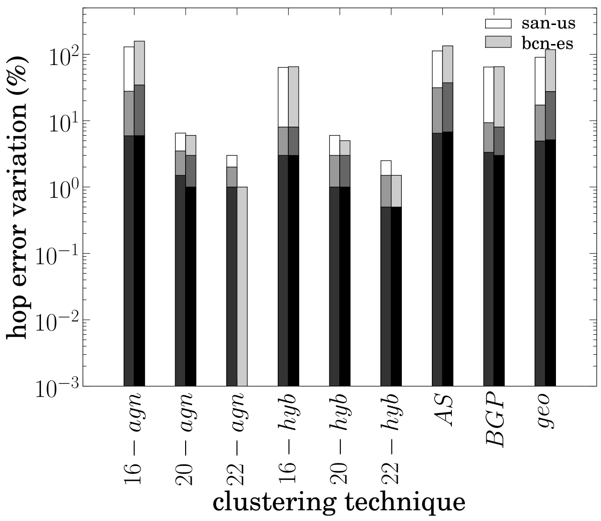
CDF.

Fig. 6.5 shows that 26.1% of the destinations are reached by, at least, 2 different sources. Collaboration will thus reduce measurements for this dataset. In addition, the top 250 destinations are reached by more than 2,000 different sources. Theses destinations are the major CDNs and web search engines.

Fig. 6.6 shows the CDF of the cluster sizes. The plot first shows that, on our dataset, clusters cover several nodes. Measurement reduction will thus



(a) RTT



(b) hop

Figure 6.4: 25^{th}, 50^{th} and 75^{th} percentile of estimation error

be observed on our dataset.

Fig. 6.6 also suggests that the clustering technique influences the cluster size. For instance, 50% of the clusters cover more than 200 addresses in AS Clustering while it is the case for less than 10% in BGP or Geographic Clustering. Moreover, the cluster size is also influenced by the n parameter when applicable. With small n, prefixes are large and have the possibility to absorb many destinations, the cluster size being therefore important. As expected, n-hybrid behaves partially like BGP for small n and like n-agnostic

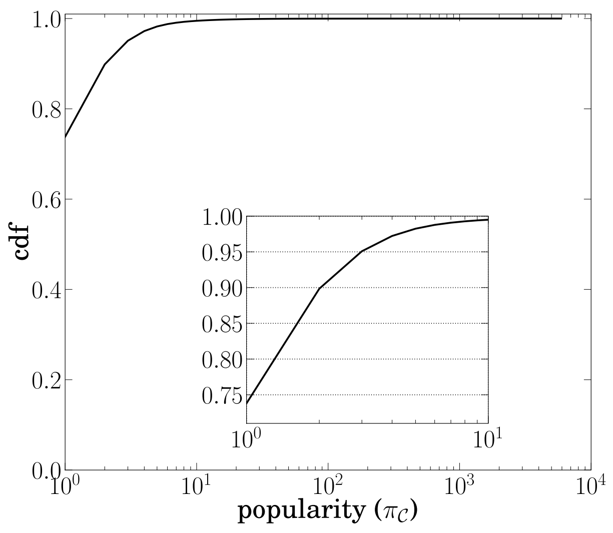


Figure 6.5: Popularity in the NetFlow dataset

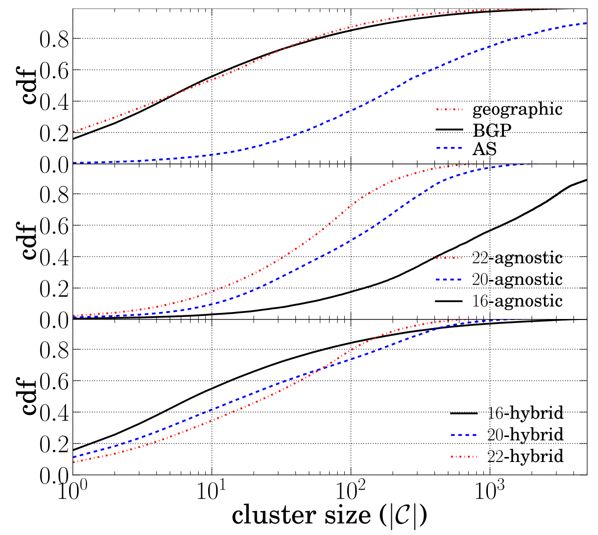


Figure 6.6: Cluster size distribution

for large n.

Fig. 6.7 shows the effective reduction factor with respect to the different clustering techniques, with or without collaboration. The vertical axis is the reduction factor while the horizontal axis indicates the clustering techniques. On Fig. 6.7, light gray bars show the reduction factor when the nodes are not collaborating. On the other hand, dark gray bars show the reduction factor when nodes are collaborating.

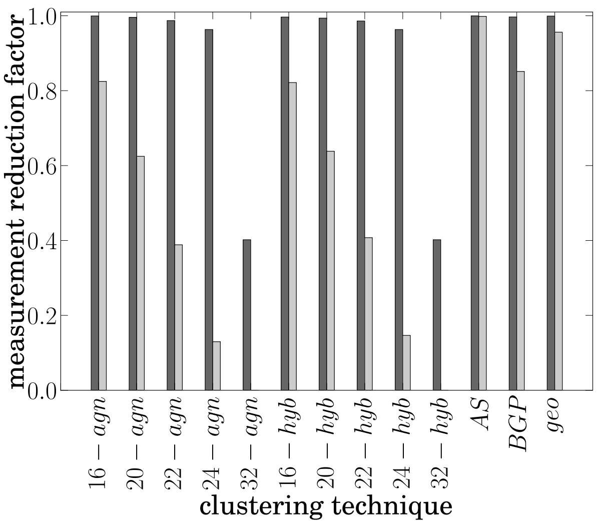


Figure 6.7: Measurement reduction in the NetFlow dataset

32-agnostic and 32-hybrid Clustering (one cluster per destination) from Fig. 6.7 confirm that collaboration can reduce measurements (by 40% in our dataset). Light gray bars confirm that clustering can reduce measurements.

Moreover, Fig. 6.7 shows that, for the same clustering technique, cooperation always offers better reduction factor than no cooperation or collaboration without clustering. Confirming so that combining collaboration and clustering gives even better measurements reduction than collaboration or clustering alone.

We also see on Fig. 6.7 that BGP, AS, and Geographic Clustering offer an important reduction factor because the number of BGP prefixes, ASes, or locations is very small compared to the number of different addresses in the NetFlow dataset.

When considering n-hybrid and n-agnostic Clustering, the reduction factor is more sensitive to n than the technique itself which is particularly true without collaboration.

Regarding to what we discussed above, we would suggest to use 20-hybrid clustering as this technique remains accurate and presents an interesting measurement reduction, even without collaboration. We would suggest not to use n-agnostic for small values of n as it will not reflect the topology. AS and geo clustering should be avoided.

In this section, we demonstrated that they effectively reduce the number of measurements while remaining quite accurate.

6.5 Conclusion

Measuring the quality of a set of paths, in terms of delay or bandwidth, is becoming more and more important for applications and services. Indeed, the resulting measurements could allow the application to select the best location for the required service or for getting a particular content. However, constantly monitoring the network through active measurements is not desirable due to scalability issues.

A solution for making measurements more scalable is to consider clustering and collaboration between measurement sources. Clustering aims at aggregating a subset of destinations into the same hat and consider that a single (or a few) measurement towards the cluster is representative of the whole cluster.

In this chapter, we first discussed how collaboration and clustering might lead to a reduction in probing effort. We defined metrics for evaluating the performance of any clustering technique. Those metrics evaluates the accuracy of an estimation based on clustering and the measurement reduction.

Secondly, we explained five different clustering techniques. Those techniques have the advantage of being very easy to setup, i.e., they do not required strong calculations. Based on real data collected, we evaluated those techniques with metrics we introduced. We demonstrated that clustering techniques offer quite accurate measurement estimations as well as measurements scalability.

In this chapter, we only took into account two networking metrics: delay and the number of hops. Future work should reveal how other network metrics, such as bandwidth and jitter, are impacted by cluster based measurements.

Chapter 7

Incoming traffic engineering with LISP and IDIPS

7.1 Introduction

On the one hand, LISP and LISP-Tree provide a way to do scalable incoming traffic engineering (see Chapter 3). Indeed, the mappings in LISP allow a site to use its multiple entry points and to give them any arbitrary preference. This implies that multiple paths can be followed to enter a LISP site and these paths potentially have different performance. On the other hand, in Chapter 5 we propose IDIPS that provides a way to rank paths in order to determine which paths offer the best performance. In this chapter, we propose to combine LISP and IDIPS to achieve performance based inter-domain incoming traffic engineering in a scalable way.

Sec. 3.2 shows how the priority, weight and TTL can be used to ensure traffic engineering with LISP. In LISP, the best locator is the locator presenting the lowest priority value. To some extent, we can thus consider the RLOC priority as its relative rank in the mapping. As a consequence, IDIPS ranks can be used to infer the RLOC priorities dynamically. Using IDIPS to attribute the LISP priorities enables automated performance based incoming traffic engineering. Sec. 2.3.3 shows that EID prefixes are likely to be attached to several locators. If an EID has several locators, which are not aliases, it means that it exists several paths to exchange packets with it. However, it is well known that the paths on the Internet are not equal [AAS03, AMS$^+$08, GDZ06, Int05, Cis, SAA$^+$99, Ava05, XYK$^+$08, DD06]. Some paths have lower delay than others, more bandwidth or are more stable. Fig. 7.1 shows that this observation holds with LISP.

Fig. 7.1, is obtained from the ping dataset we built in Sec. 2.5. Fig. 7.1 shows the relative difference of delay between all the locators of a mapping and the locator with the shortest delay in the mapping. The relative distance of the locator i with the fastest locator f of the mapping is computed by

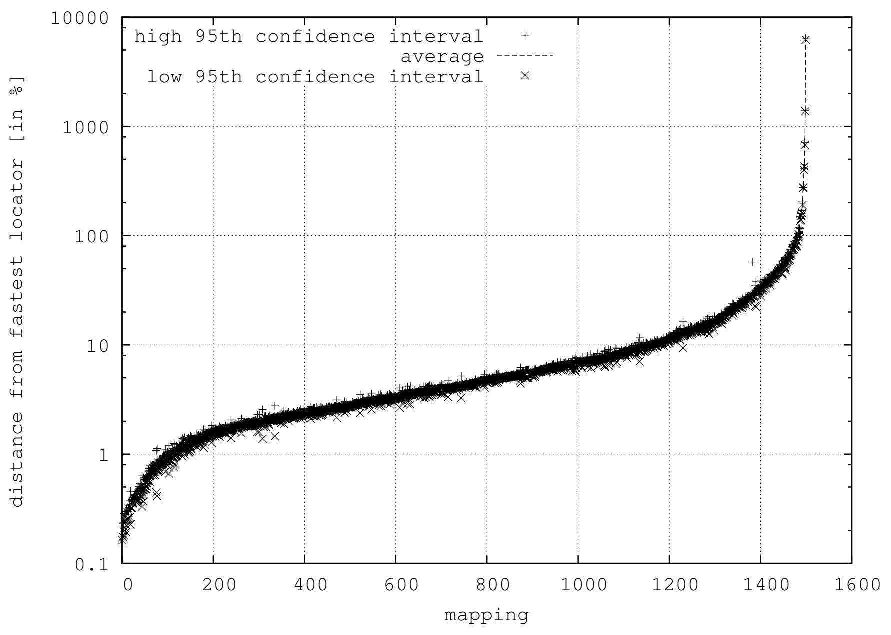


Figure 7.1: Relative distance between the locators and the locator with the shortest delay in the mappings

$\frac{rtt_i - rtt_f}{rtt_f}$. A 10% distance thus means that the locator has a delay higher by 10% than the locator with the shortest delay in the mapping. In Fig. 7.1, there is one mapping per measured stub BGP prefix. A mapping associates a BGP prefix and the IP address of its border router as determined in Sec. 2.5. In these mappings, the BGP prefix is the identifier and the border router IP addresses are the locators. The delay to the locators is the measured RTT between our vantage point and the locators (i.e., the border router). Fig. 7.1 shows the average distance to the shortest delay locator in the mapping and the 95^{th} % confidence interval. Fig. 7.1 shows that for 4% of the mappings, the locators had a delay more than 50% bigger than the shortest delay observed in the mappings with 1% of the mappings with locators presenting a delay more than twice longer than the shortest one. For few mappings, the average distance was even worst by more than 500% on average and up to 6309% higher on average for one case. Fig. 7.1 shows that a bad choice of locator can have a significant impact on the performance showing the necessity of assigning the priorities with care. However, Fig. 7.1 also shows that for 77% of the mappings, the average distance to the locator with the shortest delay in the mapping is lower than 10%. It thus means that for most of the mappings, traffic can be load balanced between at least two locators by playing with the weight without degrading the performances [KKSB07]. It thus imply that in general, it is not necessary to make measurements and simply use all the locators (or a random subset) is good enough. However, an operator that want to perform performance based traffic engineering must keep in mind that in some cases, some locators might be very bad, even if it is unlikely. Obviously, if an operator decides to measure the locators, these locators can be grouped into clusters with the techniques presented in Chapter 6. Hence, it is possible to share measurements to determine the performance of the locators of several mappings.

Fig. 7.1 shows that the traffic could benefit from a traffic engineering approach that considers performances as much as costs. IDIPS can help in the choice of the priorities an the weights in LISP. IDIPS is however designed to rank paths while LISP mappings are related to IP addresses. The locator R of a LISP mapping that is distributed on the mapping system can however bee seen as the destination of a path. Therefore, if differentiated mappings are not used, the source of the path can be any ITR RLOC. We can then define such a path as `<*,R>` where `*` corresponds to any source in the Internet. If differentiated mappings are used and the differentiation is done for the site S, then, the path is `<S,R>`. As we have seen in Chapter. 5, IDIPS supports paths defined by prefixes. Therefore, if P_S is the prefix for the site S, then the path can be defined as <P_S,`R/32`>. P_S being 0.0.0.0/0 for `*`.

As a result, IDIPS can be used to set the RLOC priorities but a translation work is required. Two options are possible to determine the priorities in a mapping. Either IDIPS is queried manually and an operator configures LISP with the appropriate priorities, or a tool automatically adapts the priorities. It is worth to notice that all along this chapter, we consider that LISP-Tree is used for the LISP mapping system in order to keep the system scalable. However, our proposition does not preclude the use of another mapping system.

The very contribution of this chapter is the presentation of a technique to perform performance based incoming traffic engineering. To do so, we combine the work presented all along this thesis. On the one hand, LISP is used as a way to perform incoming traffic engineering efficiently. On the other hand, IDIPS is a service that can determine the paths that provide the best performance. Thus, combining both provide a performance based incoming traffic engineering technique.

7.2 LISP and IDIPS combination

Operations related to interdomain traffic engineering are often performed manually by the network operators [CGG+04]. The operators configure their network such that it corresponds to the business objective and react to alarm triggered by monitoring tools. Manual operation can be continued with IDIPS used as a hint by the operator. Before setting the priorities, the operator requests IDIPS to rank the paths corresponding to the locators in the mapping. The operator then sets the priorities to reflect his interpretation of the IDIPS ranks. For instance, he can request IDIPS twice. Once with delay optimization and once with cost minimization. The best result(s) of each query being set with a priority of one, all the others with a priority of 10. Manual IDIPS-aided operation is interesting to support subjective criteria but is subject to configuration error and is time consuming in complex networks [WSR09, CGG+04].

If a translation tool is used between LISP and IDIPS, it is possible to fully automate the IDIPS-based priority computation. The tool listens the insert and update events in the EID-to-RLOC Database. The locators in the mapping that triggered the event are then extracted to build the destination list. The source list depends on whether the differentiated mapping is used or not. Once the source and destination lists are built, the tool requests IDIPS to rank these paths according to a pre-configured ranking criterion. Upon IDIPS reply, the tool translates the IDIPS rank to LISP priority by applying a t translation function on each rank such that:

$$t : \mathcal{R} \times \mathcal{I} \to \mathcal{P}$$

where $\mathcal{R} = [0; 2^{32}[\subset \mathbb{R}_0$ is the IDIPS ranking space, $\mathcal{I}$ is the space of all the possible IDIPS replies and $\mathcal{P} = [0; 255] \subset \mathbb{R}_0$ is the LISP priority space.

t can be any function and is defined according to the network policies. For example, if only the best locator can be used, t can be defined as follow:

$$t(x, m) = \begin{cases} 1 & \text{if } x = min(m) \\ 255 & \text{otherwise} \end{cases}$$

where $min(m) : \mathcal{I} \to \mathcal{R}$ gives the minimum rank observed in the IDIPS reply m.

Once the ranks have been converted into priorities, the mapping can be built. To do so, the rank of each locator r in the mapping is extracted from the IDIPS reply and converted into the associated priority. If the locator is not present in the IDIPS reply, it must be considered as non-reachable and should not be present in the mapping or with a priority set to 255.

In addition to the priority, a weight is associated to each locator in a mapping. We suggest to assign the same weight to all the locators sharing the same priority are to use statically configured weights. Indeed, if the cost functions used to determine the priority are built correctly, the priorities will react if a RLOC does not comply with the policies and the priority will be adapted accordingly.

The mapping TTL should be lower or equal to the IDIPS replied TTL. Therefore, when the mapping expires in the EID-to-RLOC database, IDIPS must be queried to re-build the mapping with up-to-date information obtained from IDIPS.

7.2.1 Proof of Concept

In this section, we show by a simple but complete proof of concept the benefits of the interaction between LISP and IDIPS. To do so, we build the testbed depicted in Fig. 7.2. The left hand network, labeled *Content Producer*, is a content producer and the right hand network is the consumer. Interdomain connectivity is ensured by LISP. For the test, we used the two types of customers *light* and *premium* and apply the `customer_management_cf` cost function presented in Sec. 5.3.3. As discussed in Sec. 5.3.3, the objective for light users is cost reduction. On the contrary, QoS has to be ensured for premium users. In the sake of clarity, in our experiments two clients with one flow per client are involved. The light client downloads a large file using FTP (TCP) while the premium client watches a video over UDP. The video must have at least a 1.4Mbps bandwidth and the jitter must be limited. The two networks are connected with two links: L1 and L2. L1 represents a peering link and L2 a customer/provider link (from the producer point of view). L2 is protected by a 128Kbps backup link. Penalties are due when QoS is not ensured for premium users. This proof of concept aims at simulating a VPN where bandwidth can be measured easily.

In the testbed, we use OPENLISP [ISB11]. OPENLISP implements the LISP protocol in the FreeBSD kernel. A particularity of OPENLISP is the *mapping socket* that allows user space applications to interact with the EID-to-RLOC mappings maintained in the kernel.

The content producer (resp. consumer) part of the testbed is operated with a dedicated Pentium 4 computer running FreeBSD and OpenLISP. The machine is used at the same time as xTR and as content producer (resp. consumer). The two machines are connected via a third machine that runs FreeBSD and dummynet to emulate the links [Riz97].

An IDIPS server instance runs in each network. At that point, neither OPENLISP nor IDIPS are aware of each other. A wrapper runs on each OPENLISP router to allow LISP to query IDIPS. The wrapper monitors the mapping socket and the IDIPS control plane. When there is an event concerning an EID of the local OPENLISP's map table, the wrapper retrieves all the RLOCs for that EID and asks the IDIPS server to rank them. The resulted ranks are translated into priorities and the EID's mapping is updated according to the information given by IDIPS.

The experiment is divided in four periods (P_1 to P_4). The RLOCs used for each period depends on the IDIPS rankings. During P_1, both L1 and L2 are working properly and IDIPS optimizes the performance for premium traffic and minimizes the cost for the light traffic. The beginning of P_2 corresponds to the L2 link failure: L2 traffic is diverted to the backup link. During L2, IDIPS is not involved and the RLOCs are not modified, premium traffic is degraded. In P_3, IDIPS is informed of the failure and modifies the mapping to minimize the cost and avoid backup links. It is worth to notice

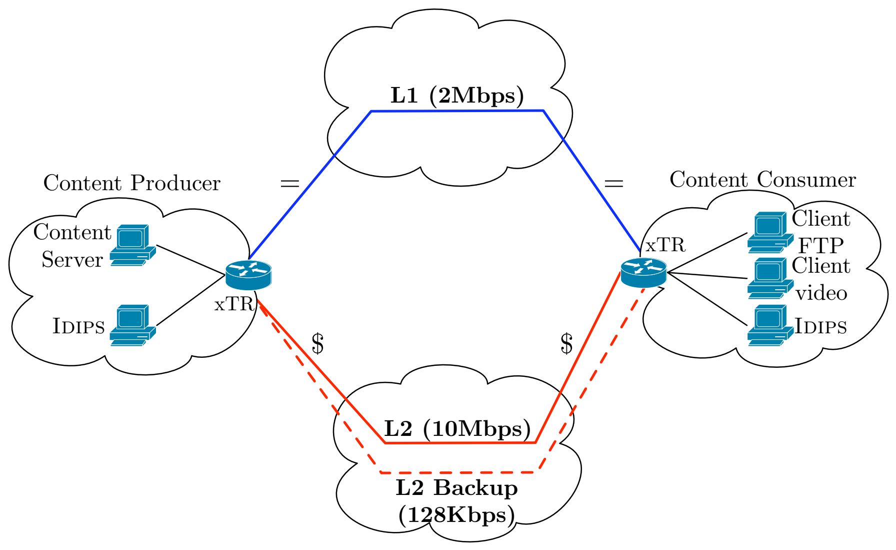


Figure 7.2: Case study testbed

that the gap between P_1 and P_3 is for illustration only, in practice, IDIPS can be informed of the topology change at the same time as the backup link activation (e.g., via SNMP). During P_3, the backup link is not used anymore. Finally, P_4 shows what happens if IDIPS policies are set to the original premium and light traffic requirements (as during P_1). In P_4, IDIPS decides to divert the light traffic (i.e., FTP) to the backup link and keep premium traffic on L1 to ensure its QoS requirements while minimizing costs. For the experiment, IPFW is instantiated on each machine to monitor the link usage thank to the IPFW statistics [LLB02, Riz97]. The volume monitored by IPFW correspond to the amount of traffic that crossed the links.

Fig. 7.3 shows the flows' dynamic during the different periods. The horizontal axis is the normalized time and the vertical axis the bandwidth (in Kbps). The best effort traffic consist of a big file transfer using FTP (TCP). The video is simulated with Iperf. Iperf continuously sends 700 bytes long UDP segments with a constant rate of 1.7 Mbps.

During P_1, both flows are working as expected: the video (premium customer) encounters a limited jitter and has enough bandwidth (1.7Mbps) and the cost for FTP (light customer) is minimized. After the failure, during P_2, the video stream is redirected to the backup link. The video flow bandwidth falls down to around 100Kbps, which is not sufficient to ensure QoS (1.4 Mbps is required to ensure QoS requirements). FTP traffic is not affected by the failure as it is carried by L1. P_3 presents the flow bandwidth when all the traffic is diverted on L1. For that period, the policies in IDIPS are to avoid backup links. However, this choice does not ensure QoS for the

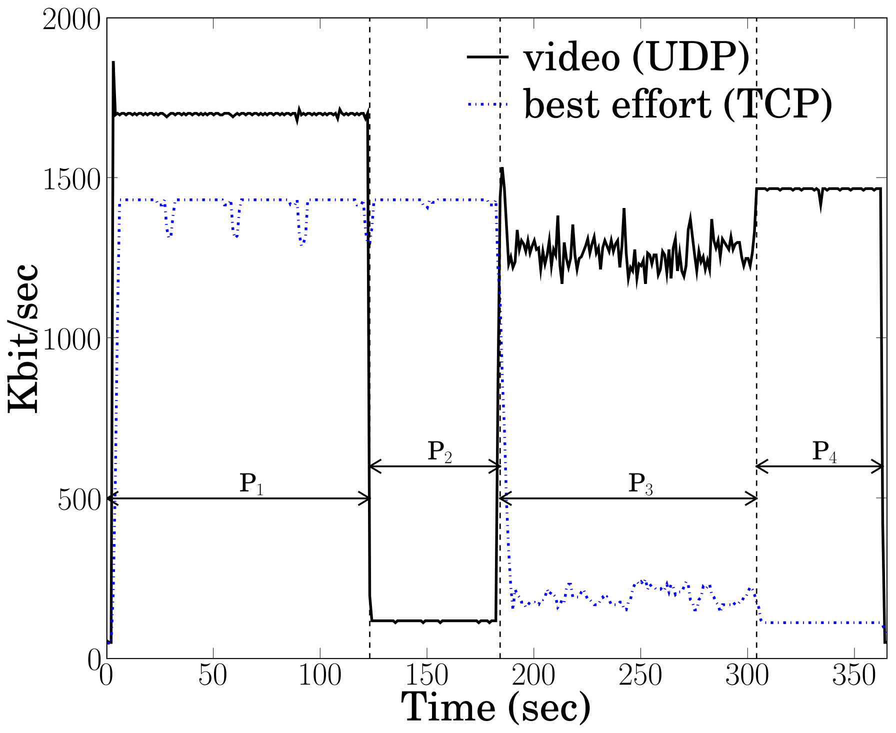


Figure 7.3: Evolution of the different flows bandwidth for the different network events.

video as the jitter is important and video bandwidth falls to 1.3Mbps. With this configuration, video traffic is influenced by the TCP behavior of the FTP flow. Period P_4 shows what happens if IDIPS is configured to ensure QoS and minimize costs, thus video is diverted on link L1 as this is the only one allowing QoS for video. The best effort flow is diverted to L2 backup link because the costs of using it is lower than the cost of losing QoS for video.

This experiment shows that IDIPS path selection algorithm can take administrative and technical questions into account (e.g., minimize costs but maximize bandwidth). Furthermore, it also shows that combining IDIPS and LISP enables performance based traffic control.

7.3 Conclusion

In this chapter, we have seen that combining LISP and IDIPS enables performance based incoming traffic engineering. On the one hand, LISP allows a site that is connected to the Internet with several providers to control the way its traffic is entering thanks to the mappings. In the mappings, a priority and a weight are attached to the entry points of the network (the locators). These priorities and the weights are used to enforce the way the traffic is entering the network. The priorities are such that they can be changed to prefer the most efficient locators and thus influence the paths followed by the packets to enter a network. To ensure that the mappings are built to optimize the performance, LISP can be combined with IDIPS. Indeed, IDIPS is a system which aim at ranking paths according to their performance and operator policies. Therefore, the priorities and weights that appear in the LISP mappings can be adapted automatically by requesting IDIPS to prefer the locators that are likely to offer the best performance. To do so, we first presented in this chapter how to build IDIPS requests from LISP mappings and how to translate the ranks provided by IDIPS to assign the appropriate priority and weight to the locators inside the mappings. We finally show with a proof of concept that doing performance based incoming traffic engineering is practically doable by combining LISP and IDIPS.

Chapter 8

Conclusion

Since its creation, the Internet has changed a lot and the first few dozens of nodes composing the Internet in the early days have been replaced by hundred of thousands nodes and billion of users. The Internet is now mostly driven by economical objectives and having a good Internet connectivity is of prime importance for many enterprises, and more generally to networks. It is then common for a network to be multihomed, i.e., connected to the Internet via several other networks. Increasing the interconnection degree in the Internet increases the number of paths between the sites. Unfortunately, all the paths from A to B are not equal. For instance, a path might be cheaper than another or a path can provide more bandwidth or lower delay. In such context, good traffic engineering is a requirement. An important topic in traffic engineering is the ability for a site to control the way its traffic is leaving or entering it. However, while it is relatively easy for a site to control its outgoing traffic, it is hard if not impossible to totally control its incoming traffic.

In Chapter 2, we describe the Locator/Identifier Separation Protocol (LISP) which is currently being standardized within the IETF. LISP separates the locator and the identifier roles of IP addresses. In LISP the IP address used to identify an end-system (i.e., the identifier) is not the IP address used for routing the packets to this end-system (i.e., the locator). This separation is done with the help of encapsulation: a packet destined to an identifier address is encapsulated in a packet with locator addresses. LISP allows to control the way the packets are routed by changing the locator. This control is transparent for the end-systems. The association between the identifiers and the locators is called a mapping. The mappings are controlled and distributed via a mapping system. LISP offers new traffic engineering perspectives. In Chapter 3, we show how to perform traffic engineering with LISP. With simple use cases, we show how the inter-domain incoming traffic can be controlled with LISP and its mappings. To this aim, every locator is assigned a priority and the locators with the lowest priority value are always preferred. In addition, we propose in Chapter 3 the mapping differentiation technique that allows a destination LISP site to inject source specific mappings. Mapping differentiation enables isolation of the TE decisions.

Playing with the mappings to perform traffic engineering in LISP has an impact on the control plane load. This is why Chapter 4 proposes LISP-Tree a new scalable mapping system for LISP. LISP-Tree scales and supports aggressive traffic engineering thanks to a DNS-like approach. In LISP-Tree, the nodes involved in the control plane are interconnected in a hierarchical way. The position of a node in the hierarchy being defined by the prefix assigned to the node. In LISP-Tree, the prefix of a node is always a sub-prefix of its parent node. The leaves of the hierarchy maintain pointers to the authoritative ETR of the prefix they own. On the contrary, the nodes inside the hierarchy are pointing to nodes deeper in the hierarchy until a leaf is pointed. The LISP-Tree can thus be seen as an ETR discovery mechanism with a loop-free property. LISP-Tree is designed to isolate configuration errors, to be secure and to be troubleshootable easily. LISP-Tree can be operated in two modes. In recursive mode, the Map-Request is forwarded on the control plane topology until it arrives a leaf. In iterative mode, the requester iteratively queries nodes in the hierarchy until it reaches a leaf, starting from the root. The iterative mode can be coupled with caching to enable fast retrieval of mappings. When caching is activated, the nodes do not need to start from the root if the EID to resolve shares a part of the path of a previously resolved EID. In Chapter 4 we provide a LISP-Tree deployment model and show by simulation that the tree is seldom traversed hence efficient.

LISP and its traffic engineering capabilities enable performance based traffic engineering. In Chapter 5, we describe our generic path ranking service called ISP-Driven Informed Path Selection (IDIPS). IDIPS relies on a request/response service. An IDIPS client can be any network application that has to choose among several paths but that must select the best ones, for any definition of best. In general, such an application selects the best paths without considering the constraints of its underlay network. This situation leads to a sub-optimal use of network resources. The most typical example is the peers selection in P2P networks. When a P2P node must select a few tens of peers among a list of hundreds of peers, it generally picks them randomly or selects them based on high level criterion. With P2P overlays, it is common to see a packet crossing the network boundaries several times. IDIPS allows the applications and the network to cooperate. When an IDIPS client must select a path among several, it queries the IDIPS server. The client sends its list of source addresses, its list of destination addresses and a ranking criterion. The server then builds the list of possible paths from the source and destination lists. The IDIPS server then ranks the paths according to the ranking criterion requested by the client. The particularity of IDIPS is that path rank is computed with cost functions and cost functions can be combined. Combining cost functions allows to rank the paths such that the best rank is provided to the paths that is the best for both the client and the network. The IDIPS server returns the sorted list of ranked paths to the clients. IDIPS supports any kind of ranking criterion but we emphasized our work on measurement based ranking. An IDIPS server runs measurement modules that monitor the paths on behalf of applications. Pooling the measurements avoid to duplicate measurements. Chapter 5 presents our IDIPS server implementation. Our IDIPS implementation evaluation shows that IDIPS can sustain a high request rate. Moreover, IDIPS is generic enough to be used by LISP to perform LISP performance traffic engineering. We describe in Chapter 7 how to combine LISP and IDIPS to perform performance based interdomain incoming traffic engineering. The solution consist in translating IDIPS ranks in LISP priorities in the mappings.

Nevertheless, measuring all the Internet is not scalable but Chapter 6 shows that the number of measurements can be reduced by clustering the nodes that present the same performance behavior. The behavior of one device in the cluster being representative for all the devices in this cluster. In Chapter 6, we rely on static clustering techniques. Static clusters are defined once for all and can be pre-computed. We can cluster the nodes to measure by autonomous system, BGP prefix, geographical position or by statically dividing the IP space into `/n` prefixes. Our evaluation shows that an hybrid between BGP clustering and static decomposition gives the better accuracy. On the hand one, the nodes tend to be grouped by topology (thanks to BGP) but the clusters remains of acceptable size (thanks to the agnostic decomposition).

8.1 Further Work

Performance based traffic engineering is composed of two distinct parts. On the one hand, the path performance must be evaluated and predicted. On the other hand, the traffic engineering decisions must be enforced.

In this thesis, we propose IDIPS, a service to simplify the operation of measuring and predicting path performance. However, IDIPS is only the beginning of the story and many work still have to be done for determining the path performance. First, we must find a solution to efficiently measure the path performances. The first step for measuring the paths is to determine the paths that are worth measuring. An algorithm that is able to build the list of the most important paths has to be designed. This algorithm must be designed such that it accepts almost any definition of “important path”, it must support technical constraints such as a limit in the memory allocated for running such algorithm and finally it must be accurate and dynamic enough to respond to traffic pattern changes. Once the important paths are detected, the measurements have to be performed carefully. The measurements are used to predict the path performance. The number and the frequency of measurements is driven by the expected accuracy of the prediction. Two solutions are possible to determine the number and frequency of measurements either reactive of proactive. For both approaches, machine learning techniques can be used. A particular care must be taken to avoid oscillations and instabilities in the network.

In this thesis, we use path performance prediction to control the traffic such that it always uses the best paths. Hereabove, we mention that the first challenge of performance based traffic engineering is the performance prediction. However, the prediction is not the only challenge. Another important challenge is the path enforcement. One it is decided that a path is worth using, the traffic must be constraint to use the path. For the path enforcement, we use the LISP protocol. LISP is a locator/identifier separation protocol that separates the two roles of the IP addresses. The separation simplifies the traffic engineering operations. Unfortunately, separating the locators from the identifiers makes the reachability detection harder. Indeed, in today's Internet routing failures are recovered locally and a packet can be delivered to any router able to reach the packet destination. With LISP, the packets are directly targeted to a particular router. If the router is not reachable, the packet cannot be delivered even if other routers are reachable for the destination. Reachability recovery for the locator/identifier separation is of prime importance and research must focus on solutions that minimize the packet losses. Another fundamental research topic related to path enforcement is the security of the enforcement. When a packet is forced to travel via a path, it must be ensured that the enforcement is legitimate. Any path enforcement system must be designed to embed security. More precisely, the system must be able to determine if an enforcement is valid or the result of an attack or a misconfiguration. LISP allows path enforcement thanks to a mapping system. However, the mapping system is virtually unsecured. In [SIB11, MEC$^+$11] we show the limitations of the LISP security and give the first milestones for securing LISP but the problem is vast an is still open to efficient and deployable solutions. Finally, the locator/identifier separation paradigm is a new way of thinking the routing and it would be interesting to model such paradigm to understand the connection between the fundamentals components of the map-and-encap mechanisms defined in LISP. Modeling LISP would help to have a better understanding of the Internet and would, for example, provide good insights about how to deploy LISP-Tree to be efficient for both economical and technical perspectives.

Appendix A

LISP-Click: A Click implementation of LISP

A.1 Introduction

From a standardization viewpoint, implementations are used to detect weaknesses in the protocol specifications and determine if performance requirements are achieved. From a research viewpoint, implementations authorize tests in the wild to validate simulations and theoretical analysis. To the best of our knowledge, OpenLISP, a dataplane implementation in the FreeBSD kernel, is the only open source implementation [ISB11] of LISP. OpenLISP is focused on performance while our approach is more focused on extensibility and simplicity for the researchers by implementing LISP in Click [KMC$^+$00].

A.2 LISP-Click architecture

LISP-Click is a Click 1.7.0 rc1 [KMC$^+$00] LISP data-plane implementation. Fig. A.1 shows the LISP-Click architecture. When an IP packet arrives at the router, it is inspected by an the `IPClassifier` that determines whether the packet is related to the LISP dataplane or the LISP controlplane. All the control plane packets are transmitted to the `LISPCache` element that is in charge of the EID-to-RLOC Cache and the EID-to-RLOC database. If the packet is for the LISP dataplane, the IPClassifier determines if the packet has to be LISP-encapsulated or LISP-decapsulated. Packets to decapsulate are transmitted to the `LISPDecap` element and packets to encapsulate are transmitted to the `LISPEncap` element.

The `LISPCache` element is the core of the router as it maintains the EID-to-RLOC Cache and the EID-to-RLOC Database. This element can be queried by the `LISPEncap` and `LISPDecap` elements at any time.

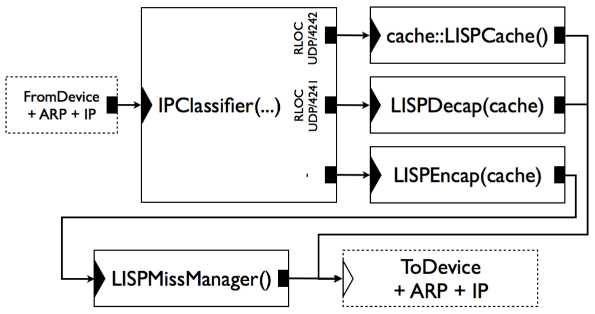


Figure A.1: LISP-Click Architecture

Annotation is used to mark packets that caused a miss while encapsulating them. Encapsulated packets are transmitted to the `LISPMissManager` element by the `LISPEncap` element. If the packet is annotated with the `miss` value, the `LISPMissManager` requests a mapping for the destination EID. Otherwise, the packet is simply transferred to the appropriate outgoing interface.

```
struct lisphdr {
#if CLICK_BYTE_ORDER == CLICK_LITTLE_ENDIAN
        unsigned rflags:5;
        unsigned e_bit:1;
        unsigned l_bit:1;
        unsigned n_bit:1;
#elif CLICK_BYTE_ORDER == CLICK_BIG_ENDIAN
        unsigned n_bit:1;
        unsigned l_bit:1;
        unsigned e_bit:1;
        unsigned rflags:5;
#else
#   error "unknown byte order"
#endif
        unsigned lisp_data_nonce:24;
        uint32_t lisp_loc_status_bits;
} __attribute__ ((__packed__));
```

A.2.1 LISPCache Element

The `LISPCache` element is a special element that stores the mapping to be used for the encapsulation and the decapsulation. To do so, the `LISPCache` element has the `match_db` and `match_cache` methods. Both methods return the list (in a vector) of valid locators associated to the

EID address requested. The `match_db` determines this list by querying the EID-to-RLOC Database, i.e., the source locators and the `match_cache` gives the list from the EID-to-RLOC Cache, i.e., the destination locators.

The datastructure used for storing the mappings in EID-to-RLOC cache and the EID-to-RLOC is a radix tree[1]. Contrary to OpenLISP, the cache and the database are stored in two separated tree instances. In OpenLISP, an EID prefix is either in the cache or in the database. In LISP-Click, an EID prefix can be at the same time in the cache and in the database. This design slightly increases the memory consumption but offers more flexibility. Indeed, in our implementation, it is possible to have an EID prefix that is at the same time local and distance. This feature has been thought for mobility where it could be interesting to encapsulate destination that belongs the source prefix. This feature is however to be considered with care to avoid infinite encapsulation and loops. Radix tree keys are the EID prefixes and the values are list of locators, the priority and weight is not implemented.

The `LISPCache` element also maintains meta information about the locators that appear in the mappings. The datastructure named `locator_status` maintains information about the locators in a hashtable. The stored information are the locator reachability, the status of the echo-nonce algorithm and the nonce associated to the locator:

```
struct rloc_info {
bool R;                 /* reachable */
bool E_in;              /* echo nonce, decided remotely */
bool E_out;             /* echo nonce, decided localy */
bool N;                 /* nonce present */
uint32_t nonce:24;
};
```

A C structure has been used for maintaining the locator information instead of a class to limit the cost of processing the information during the encapsulation or the decapsulation.

Handlers are access points through which users can interact with the internal state of a running click element. Handlers are used to implement user interfaces on Click routers.

The `LISPCache` implements three write handlers that allow a user to modify the element state:

insert: insert a new EID prefix in the EID-to-RLOC Cache or Database. `insert <local: bool> <prefix: prefix>` where local determines whether the database has to be modified or the cache. The inserted EID prefix has no RLOC associated.

[1]The radix tree has been extracted from the Quagga source code.

update: add or modify an RLOC in an EID-to-RLOC Cache or Database mapping. `update <local: bool> <prefix: prefix> <rloc: ip> <priority: short> <weight: short> <R: bool>`

rloc: modify the state corresponding to an RLOC maintained in the `locator_status` hashtable. `rloc <rloc: ip> <R: bool> <N: bool> <E: bool> <nonce: int>`. The R parameter sets the locator reachability. If N is set, the encapsulation must set the N bit and use the nonce given in parameter. If E is set, the encapsulation must request the ETR to run the echo-nonce algorithm with the nonce given in parameter.

A.2.2 LISPDecap Element

The `LISPDecap` element is the element that decapsulate LISP packets. It is a 1 input/1 output agnostic element. Click offers simple tools to perform decapsulation, as presented in the code below.

```
Packet * LISPDecap::simple_action(Packet *p_in){
01. click_ip * ip = (click_ip *)(p_in->data());
02. click_udp * udp = (click_udp *)(ip + 1);
03. struct lisphdr * lisp = (struct lisphdr *)(udp + 1);
04. click_ip * payload = (click_ip *)(lisp + 1);

05. ... // LISP stuff

06. int hsize = sizeof(click_ip) + sizeof(click_udp) \
                           + sizeof(struct lisphdr);
07. p_in->pull(hsize);

08. IPAddress deid = IPAddress(payload->ip_dst);
09. p_in->set_dst_ip_anno(deid);

10. return p_in;}
```

The block from line 1 to line 4 provides pointers to the different header in the packet to decapsulate. The block in line 5, not represented here, is where the the LISP header is processed to extract information like the nonce. [2] Finally, lines 6 – 7 are for the decapsulation itself. To decapsulate a packet in Cick, it is only required to call the pull method on the packet instance and specify the number of bytes (`hsize`) to strip from it. Lines 8 and 9 are required to define the new destination of the packet. Indeed, the initial destination address is the destination RLOC but once decapsulated, the destination is the destination EID contained in the inner header. Finally,

[2] not implemented

the decapsulated packet can be returned to be processed by the next element in the click module.

The `LISPDecap` element maintains a pointer to the LISPCache instance to perform sanity checks and LISP-related processing. For example, if the LISP header has the E bit set, the echo-nonce bit has to be started for the source locator.[3]. This is done simply by accessing the `locator_status` variable available at the cache instance. The LISPCache instance is passed to the `LISPDecap` element via the `configure` method. In the Click configuration file, it is thus as simple as writing the two following lines to link the `LISPCache` and the `LISPDecap` elements.

```
cache :: LISPCache;
encap :: LISPEncap(CACHE cache);
```

A.2.3 LISPEncap Element

The `LISPEncap` element is the element that encapsulates packets whose destination are EID addresses. It is a 1 input/1 output push element. The encapsulation of a packet can be decomposed in three parts. First, the source and destination locators have to be determined. Second, the packet has to be"stretch at the front" to accept the new headers (the outer header, the UDP header and the LISP header). Finally, the newly added headers have to be filled with the appropriate values. The code below shows the important lines of code corresponding to these different steps.

```
void LISPEncap::push(int, Packet *p_in){
01. cache->match_database(p_in->ip_header()->ip_src, sources);
02. IPAddress srloc = get_rloc(sources);
03. if(cache->match_cache(p_in->ip_header()->ip_dst, destinations) < 0)
04.   SET_MISS_ANNO(p_in);
05. IPAddress drloc = get_rloc(destinations);

06. int hsize = sizeof(click_ip) + sizeof(click_udp) \
                    + sizeof(struct lisphdr);
07. WritablePacket *packet = p_in->push(hsize);



08. click_ip *ip = reinterpret_cast<click_ip *>(packet->data());
09. click_udp *udp = reinterpret_cast<click_udp *>(ip + 1);
10. struct lisphdr *lisp = reinterpret_cast<struct lisphdr *>(udp + 1);

11. ... // header preparation
    ... ip->ip_ttl = p_in->ip_header()->ip_ttl;
```

[3]not implemented

```
12. ((Packet *)packet)->set_dst_ip_anno(drloc);

13. output(0).push(packet);}
```

Lines 1 – 5 determines the source and destination RLOCs from from the LISPCache element (cache). The line 4 annotates the packet with a MISS annotation if the packet caused an EID-to-RLOC cache miss. This annotation is used by the LISPMissManager element.

Lines 6 and 7 are used to increase the size of the packet, calling the push method on a packet add space before the packet. This empty space corresponds to the outer header, the UDP header and the LISP header (size equivalent tohsize). Lines 8 to 10 generates pointer to these headers.

The block in line 11 has the code that fed these headers. The content depends on the locator_status state from the LISPCache element instance. In this block, it is very important to fix the TTL of the outer header correctly, we set it to the same value as the inner header TTL. This way of assigning the TTL ensures that the packet will not last for ever in case of routing or mapping loop [4].

Similarly to what is done in LISPDecap element, line 12 annotates the new packet with its destination address, i.e., the destination RLOC. This operation is required as the original packet had and EID as destination.

Finally, the packet can be pushed to the next element in the module at line 13.

A.2.4 LISPMissManager Element

The LISPMissManager element is the last LISP specific element in the LISP-Click implementation. It is a 1 input/1 output port push element.

The LISPMissManager element can be seen as an annotation based filter. When a packet is passed to the element, its MISS annotation is read. If the miss annotation is not set, the packet is passed as-is to the next element in the module. On the contrary, if the miss annotation is set, a Map-Request is sent for EID that caused the miss[5] and the packet is dropped.

We study in Sec. 4.4 the effects of cache misses on packets. Three options are possible when a cache miss occurs: (*i*) drop the packet, (*ii*) send a data-probe and (*iii*) queue the packet until a mapping is known. These options have to be implemented in the LISPMissManager element. We have implemented the first solution only. From our point of view, data-probing is not a good solution as it can overload the mapping system by giving it data-plane functionality. The queuing strategy would be interesting to evaluate

[4]As long as the TTL of a packet is decreased as soon as it arrives a the router.

[5]code in comments as not compatible with the specifications

empirically to determine the impact it has on real traffic. To implementing the queuing strategy, the LISP-Cache must implement a solution that advertises the `LISPMissManager` element that the cache has been updated.

Bibliography

[AAS03] A. Akella, Shaikh A., and R. Sitaraman. A measurement-based analysis of multihoming. In *Proc. ACM SIGCOMM*, August 2003.

[ABH08] R. Atkinson, S. Bhatti, and S. Hailes. Mobility through naming: Impact on DNS. In *MobiArch*, August 2008.

[ABH10] Randall Atkinson, Saleem Bhatti, and Stephen Hailes. Evolving the Internet Architecture Through Naming. *IEEE Journal on Selected Areas in Communications*, 28(8):1319–1325, October 2010.

[ACE$^+$02] D. O. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. Overview and Principles of Internet Traffic Engineering. Internet Engineering Task Force, RFC3272, May 2002.

[ACK03] S. Agarwal, C-N. Chuah, and R. H. Katz. OPCA: Robust interdomain policy routing and traffic control. In *Proc. IEEE Conference on Open Architecture and Network Programming (OPENARCH)*, April 2003.

[AFP$^+$09] O. Akonjang, A. Feldmann, S. Previdi, B. Davie, and D. Saucez. The PROXIDOR Service. Internet draft, draft-akonjang-alto-proxidor-00, work in progress, March 2009.

[AFS07] V. Aggarwal, A. Feldmann, and C. Scheideler. Can ISPs and P2P users cooperate for improved performance. *ACM SIGCOMM CCR*, 37(3):29–40, July 2007.

[AFT07] Brice Augustin, Timur Friedman, and Renata Teixeira. Measuring load-balanced paths in the internet. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 149–160, New York, NY, USA, 2007. ACM.

[Ahm05] A. Reaz S. Atiquzzaman M. Fu S. Ahmed. Performance of DNS as location manager. In *IEEE International Conference on Electro Information Technology.* IEEE, 2005.

[AJ02] Daniel O. Awduche and Bijan Jabbari. Internet traffic engineering using multi-protocol label switching (mpls). *Comput. Netw.*, 40:111–129, September 2002.

[Aka] Akamai. Web application acceleration and performance management, streaming media services, and content delivery. See `http://www.akamai.com`.

[AKZ99] G. Almes, S. Kalidindi, and M. Zekauskas. A Round-trip Delay Metric for IPPM. RFC 2681 (Proposed Standard), September 1999.

[AL09] Sharad Agarwal and Jacob R. Lorch. Matchmaking for online games and other latency-sensitive P2P systems. In *SIGCOMM*, pages 315–326, August 2009.

[AMA$^+$99] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering Over MPLS. RFC 2702 (Informational), September 1999.

[AMS$^+$08] A. Akella, B. Maggs, S. Seshan, A. Shaikh, and R. Sitaraman. On the performance benefits of multihoming route control. *IEEE Transactions on Networking*, 16(1):96–104, February 2008.

[AmW99] Daniel O. Awduche and Uunet (mci Worldcom). Mpls and traffic engineering in ip networks. *IEEE Communications Magazine*, 37:42–47, 1999.

[APS04] A. Akella, J. Pang, and A. Shaikh. A Comparison of Overlay Routing and Multihoming Route Control. In *Proceedings of ACM SIGCOMM*, Portland, Oregon, August 2004.

[APY11] R. Alimi, R. Penno, and Y. Yang. ALTO Protocol. Internet Draft (Work in Progress) draft-ietf-alto-protocol-09, Internet Engineering Task Force, June 2011.

[Ava05] Avaya. Adaptative networking software (ANS), 2005.

[BA06] M. Bagnulo and J. Arkko. Cryptographically Generated Addresses (CGA) Extension Field Format. RFC 4581 (Proposed Standard), October 2006.

[Bag09] M. Bagnulo. Hash-Based Addresses (HBA). RFC 5535 (Proposed Standard), June 2009.

[BCC+06] Ruchir Bindal, Pei Cao, William Chan, Jan Medved, George Suwala, Tony Bates, and Amy Zhang. Improving traffic locality in bittorrent via biased neighbor selection. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ICDCS '06, pages 66–, Washington, DC, USA, 2006. IEEE Computer Society.

[BCF+08] Scott Brim, Noel Chiappa, Dino Farinacci, Vince Fuller, Darrel Lewis, and David Meyer. LISP-CONS: A Content distribution Overlay Network Service for LISP. draft-meyer-lisp-cons-04, April 2008. Work in progress.

[BFCW09] Hitesh Ballani, Paul Francis, Tuan Cao, and Jia Wang. Making routers last longer with viaggre. In *USENIX NSDI*, April 2009.

[BFMR10] Kevin Butler, Toni Farley, Patrick Mcdaniel, and Jennifer Rexford. A Survey of BGP Security Issues and Solutions. *IEEE/ACM Transactions on Networking*, 98(1):100–122, January 2010.

[BNC02] A. Broido, E. Nemeth, and K. Claffy. Internet expansion, refinement and churn. *European Transactions on Telecommunications*, January 2002.

[BRISC+07] Pere Barlet-Ros, Gianluca Iannaccone, Josep Sanjuàs-Cuxart, Diego Amores-López, and Josep Solé-Pareta. Load shedding in network monitoring applications. In *USENIX Annual Technical Conference*, 2007.

[Bro04] G. Brown. Internet address space clustering for intelligent route control. see `http://www.cs.indiana.edu/~geobrown/journal-new.pdf`, 2004.

[CAI10] CAIDA. AS-Rank, 2010. See `http://as-rank.caida.org/`.

[CASBDP09] Albert Cabellos-Aparicio, Damien Saucez, Olivier Bonaventure, and Jordi Domingo-Pascual. Validation of a LISP simulator. Technical Report UPC-DAC-RR-CBA-2009-8, UPC, 2009.

[CB08] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: a practical approach to reducing cross-isp traffic in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, SIGCOMM '08, pages 363–374, New York, NY, USA, 2008. ACM.

[CBB00] B Cheswick, H. Burch, and S. Branigan. Mapping and visualizing the internet. In *Proc. USENIX Annual Technical Conference*, June 2000.

[CCRK04] M. Costa, M. Castro, R. Rowstron, and P. Key. PIC: Practical Internet coordinates for distance estimation. In *Proc. 24th International Conference on Distributed Computing Systems*, March 2004.

[CDN$^+$97] Kenneth Calvert, Matthew B. Doar, Ascom Nexion, Ellen W. Zegura, Georgia Tech, and Georgia Tech. Modeling internet topology. *IEEE Communications Magazine*, 35:160–163, 1997.

[CFM99] R. Coltun, D. Ferguson, and J. Moy. OSPF for IPv6. RFC 2740 (Proposed Standard), December 1999. Obsoleted by RFC 5340.

[CGG$^+$04] Don Caldwell, Anna Gilbert, Joel Gottlieb, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexford. The cutting edge of ip router configuration. *SIGCOMM Comput. Commun. Rev.*, 34:21–26, January 2004.

[CH10] X. Cai and J. Heidemann. Understanding block-level address usage in the visible Internet. In *Proc. ACM SIGCOMM*, August 2010.

[cHKF09] kc claffy, Y. Hyun, K. Keys, and M. Fomenkov. Internet mapping: from art to science. In *Proc. IEEE Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, March 2009.

[Cis] Cisco Systems. Optimized edge routing (EOR).

[CJC$^+$09] Florin Coras, Lorand Jakab, Albert Cabellos, Jordi Domingo-Pascual, and Virgil Dobrota. Coresim: A simulator for evaluationg locator/id separation protocol mapping systems. In *Trilogy Future Internet Summer School poster session*, 2009.

[CL05] R.K.C. Chang and M. Lo. Inbound Traffic Engineering for Multihomed ASs Using AS-Path Prepending. *IEEE Network Magazine*, pages 18–25, March/April 2005.

[Cla04] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004.

[CLH03] K. Cho, M. Luckie, and B. Huffaker. Identifying IPv6 network problems in the dual-stack world. In *Proc. ACM SIGCOMM Workshop on Network Troubleshooting*, September 2003.

[Cor09] Florin Coras. CoreSim: A simulator for evaluating LISP mapping systems. Master's thesis, Technical University of Cluj-Napoca, June 2009.

[D. 09] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. LISP Alternative Topology (LISP+ALT). draft-ietf-lisp-alt-01, May 2009. Work in progress.

[DB08] Benoit Donnet and Olivier Bonaventure. On bgp communities. *SIGCOMM Comput. Commun. Rev.*, 38:55–59, March 2008.

[DCKM04] F. Dabek, R. Cox, K? Kaashoek, and R. Morris. Vivaldi, a decentralized network coordinated system. In *Proc. ACM SIGCOMM*, August 2004.

[DD06] Amogh Dhamdhere and Constantinos Dovrolis. Isp and egress path selection for multihomed networks. In *IEEE INFOCOM*, 2006.

[DD08] Amogh Dhamdhere and Constantine Dovrolis. Ten years in the evolution of the internet ecosystem. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 183–196, New York, NY, USA, 2008. ACM.

[DFC05] B. Donnet, T. Friedman, and M. Crovella. Improved algorithms for network topology discovery. In *Proc. Passive and Active Measurement Workshop (PAM)*, March 2005.

[DFD10] Amogh Dhamdhere, Pierre Francois, and Constantine Dovrolis. A value-based framework for internet peering agreements. pages 1–8, 2010.

[DH98] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998. Updated by RFC 5095.

[DHKS09] Xenofontas Dimitropoulos, Paul Hurley, Andreas Kind, and Marc Stoecklin. On the 95-percentile billing method. In *Passive and Active Measurements Conference (PAM)*, April 2009.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[DKF$^+$07] Xenofontas Dimitropoulos, Dmitri Krioukov, Marina Fomenkov, Bradley Huffaker, Young Hyun, kc claffy, and George Riley. As relationships: inference and validation. *SIGCOMM Comput. Commun. Rev.*, 37:29–40, January 2007.

[dL05] Cédric de Launois. *Unleashing traffic engineering for IPv6 multihomed sites.* PhD thesis, Université catholique de Louvain, October 2005.

[dLBL03] C. de Launois, O. Bonaventure, and M. Lobelle. The NAROS Approach for IPv6 Multi-homing with Traffic Engineering. In *Proceedings of QoFIS, LNCS 2811, Springer-Verlag*, pages 112–121, October 2003.

[dLQB06] C. de Launois, B. Quoitin, and O. Bonaventure. Leveraging networking performance with IPv6 multihoming and multiple provider-dependent aggregatable prefixes. *Computer Networks*, 50(8):1145–1157, June 2006.

[dLUB05] C. de Launois, S. Uhlig, and O. Bonaventure. Scalable route selection for IPv6 multihomed sites. In *Proc. IFIP Networking*, May 2005.

[DOK92] Peter B. Danzig, Katia Obraczka, and Anant Kumar. An analysis of wide-area name server traffic: a study of the internet Domain Name System. *SIGCOMM Comput. Commun. Rev.*, 22(4):281–292, 1992.

[Dra03] R. Draves. Default address selection for Internet protocol version 6 (IPv6). RFC 3484, Internet Engineering Task Force, February 2003.

[DRFC05] B. Donnet, P. Raoult, T. Friedman, and M. Crovella. Efficient algorithms for large-scale topology discovery. In *Proc. ACM SIGMETRICS*, June 2005.

[EdGV98] H. Eidnes, G. de Groot, and P. Vixie. Classless IN-ADDR.ARPA delegation. RFC 2317 (Best Current Practice), March 1998.

[EJLW01] Anwar Elwalid, Cheng Jin, Steven Low, and Indra Widjaja. Mate: Mpls adaptive traffic engineering. pages 1300–1309, 2001.

[FF09] Dino Farinacci and Vince Fuller. LISP Map Server. draft-fuller-lisp-ms-00, March 2009. Work in progress.

[FFM03] M. J. Freedman, E. Freudenthal, and D. Maziéres. Democratizing content pulication with coral. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2003.

[FFM$^+$07] Angelo Fanelli, Michele Flammini, Domenico Mango, Giovanna Melideo, and Luca Moscardelli. Experimental evaluations of algorithms for ip table minimization. In *Proceedings of the 6th international conference on Experimental algorithms*, WEA'07, pages 324–337, Berlin, Heidelberg, 2007. Springer-Verlag.

[FFML10a] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID separation protocol (LISP). Internet Draft (Work in Progress) draft-ietf-lisp-08, Internet Engineering Task Force, August 2010.

[FFML10b] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis. LISP Alternative Topology (LISP+ALT). Internet Draft (Work in Progress) draft-ietf-lisp-alt-05, Internet Engineering Task Force, October 2010.

[FGL$^+$00] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Transactions on Networking*, 9:265–279, 2000.

[FJP$^+$99] P. Francis, S. Jamin, V. Paxson, L. Zhang, D. F. Gruniewicz, and Y. Jin. An architecture for a global Internet host distance estimator service. In *Proc. IEEE INFOCOM*, March 1999.

[FL06] V. Fuller and T. Li. Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan. RFC 4632 (Best Current Practice), August 2006.

[FLYV93] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy. RFC 1519 (Proposed Standard), September 1993. Obsoleted by RFC 4632.

[FMS10] D. Farinacci, D. Meyer, and J. Snijders. LISP Canonical Address Format (LCAF). Internet draft, draft-farinacci-lisp-lcaf-04, work in progress, October 2010.

[FRH$^+$11] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), March 2011.

[FRT02] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *IEEE Communications Magazine*, 40:118–124, 2002.

[FT00] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. 19th IEEE Conf. on Computer Communications (INFOCOM)*, pages 519–528, 2000.

[FT02] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, 2002.

[Gao01] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9:733–745, December 2001.

[GCLC04] F. Guo, J. Chen, W. Li, and T. Chiueh. Experiences in Building a Multihoming Load Balancing System. In *Proceedings of IEEE INFOCOM*, March 2004.

[GCX$^+$05] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, analysis, and modeling of bittorrent-like systems. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, IMC '05, pages 4–4, Berkeley, CA, USA, 2005. USENIX Association.

[GDZ05] Ruomei Gao, Constantinos Dovrolis, and Ellen W. Zegura. Interdomain ingress traffic engineering through optimized aspath prepending. In *In Proceedings of IFIP Networking*, pages 647–658, 2005.

[GDZ06] R. Gao, C. Dovrolis, and E. Zegura. Avoiding oscillations due to intelligent route control systems. In *Proc. IEEE INFOCOM*, April 2006.

[GR00] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. *SIGMETRICS Perform. Eval. Rev.*, 28:307–317, June 2000.

[GSW99] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. Policy disputes in path-vector protocols. In *Proceedings of the Seventh Annual International Conference on Network Protocols*, ICNP '99, pages 21–, Washington, DC, USA, 1999. IEEE Computer Society.

[GW99] Timothy G. Griffin and Gordon Wilfong. An analysis of bgp convergence properties. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 277–288, New York, NY, USA, 1999. ACM.

[HHA$^+$] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, and k. claffy. The CAIDA IPv4 Routed /24 Topology Dataset - August 2008. `http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml`.

[HHK03] Mark Handley, Orion Hodson, and Eddie Kohler. Xorp: an open platform for network research. *SIGCOMM Comput. Commun. Rev.*, 33:53–57, January 2003.

[Hop00] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000.

[HS05] N. Hu and P. Steenkiste. Exploiting Internet route sharing for large-scale available bandwidth estimation. In *Proc. ACM/Usenix Internet Measurement Conference (IMC)*, October 2005.

[Hus] Geoff Huston. Growth of the BGP table - 1994 to present.

[HWLR08] Cheng Huang, Angela Wang, Jin Li, and Keith W. Ross. Measuring and evaluating large-scale cdns paper withdrawn at mirosoft's request. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 15–29, New York, NY, USA, 2008. ACM.

[IB07] L. Iannone and O. Bonaventure. On the cost of caching locator/id mappings. In *Proc. ACM CoNEXT*, December 2007.

[Int05] Internap. Premise-base route optimisation, 2005.

[ISB11] Luigi Iannone, Damien Saucez, and Olivier Bonaventure. Implementing the locator/id separation protocol: Design and experience. *Computer Networks*, 2011. To appear, http://dx.doi.org/10.1016/j.comnet.2010.12.017.

[JCAC$^+$10] Loránd Jakab, Albert Cabellos-Aparicio, Florin Coras, Damien Saucez, and Olivier Bonaventure. LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System. *IEEE Journal on Selected Areas in Communications*, 28(8):1332–1343, 2010.

[JSBM02] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS performance and the effectiveness of caching. *IEEE/ACM Transactions on Networking*, 10(5):589–603, October 2002.

[KCGR09] Changhoon Kim, Matthew Caesar, Alexandre Gerber, and Jennifer Rexford. Revisiting route caching: The world should be flat. In *PAM*, April 2009.

[KIF11] Juhoon Kim, Luigi Iannone, and Anja Feldmann. Deep dive into the lisp cache and what isps should know about it. In *IFIP International Conference on Networking*, volume 6640 of *Lecture Notes in Computer Science (LNCS)*, pages 267–278, Berlin / Heidelberg, Germany, May 2011. Springer.

[KKK07] Nate Kushman, Srikanth Kandula, and Dina Katabi. Can you hear me now?!: it must be bgp. *SIGCOMM Comput. Commun. Rev.*, 37:75–84, March 2007.

[KKSB07] Srikanth Kandula, Dina Katabi, Shantanu Sinha, and Arthur Berger. Dynamic load balancing without packet reordering. *SIGCOMM Comput. Commun. Rev.*, 37:51–62, March 2007.

[KKY03] R. Katz, K. Kompella, and D. Yeung. Traffic Engineering (TE) Extensions to OSPF Version 2. Internet Engineering Task Force, RFC3630, September 2003.

[KMC$^+$00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18:263–297, August 2000.

[KPS$^+$11] S. Kiesel, S. Previdi, M. Stiemerling, R. Woundy, and Y r. Yang. Application-Layer Traffic Optimization (ALTO) Requirements. Internet Draft (Work in Progress) draft-ietf-alto-reqs-08, Internet Engineering Task Force, March 2011.

[KRP05] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet service providers fear peer-assisted content distribution. In *Proc. Internet Measurement Conference (IMC)*, October 2005.

[KTCI04] R Keralapura, N Taft, C N Chuah, and G Iannaconne. Can isps take the heat from overlay networks? In *Proceedings of the 3rd Workshop on Hot Topics in Networks (HotNets-III)*, San Diego,, 2004.

[KW00] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proc. ACM SIGCOMM*, August 2000.

[KW01] B. Krishnamurthy and J. Wang. Topology modeling via cluster graphs. In *Proc. ACM SIGCOMM Workshop on Internet Measurement (IMW)*, November 2001.

[Lea08] Eliot Lear. NERD: A Not-so-novel EID to RLOC Database. draft-lear-lisp-nerd-04, April 2008. Work in progress.

[LGP^{+}05] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft. On the accuracy of embeddings for Internet coordinate systems. In *Proc. USENIX Internet Measurement Conference (IMC)*, October 2005.

[LGS07] J. Ledlie, P. Gardner, and M. I. Seltzer. Network coordinates in the wild. In *Proc. USENIX Symposium on Networked System Design and Implementation (NSDI)*, April 2007.

[LHC03] H. Lim, J. C. Hou, and C.-H. Choi. Constructing internet coordinate system based on delay measurement. In *Proc. ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2003.

[LHC05] H. Lim, J. C. Hou, and C-H. Choi. Constructing Internet coordinate system based on delay measurement. *IEEE/ACM Transactions on Networking*, 13(3):513–525, June 2005.

[Li10] T. Li. Recommendation for a routing architecture. Internet draft, draft-irtf-rrg-recommendation-14, work in progress, September 2010.

[Lig] Light Reading. Controlling P2P Traffic. `http://www.lightreading.com/document.asp?site=lightreading&doc_id=44435&page_number=3`.

[LIJM^{+}09] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, and M. Karir. Atlas internet observatory 2009 annual report. Technical report, Arbor Networks, the University of Michigan and Merit Network, 2009.

[LIJM^{+}10] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 75–86, New York, NY, USA, 2010. ACM.

[Lim] Limelight Networks. High performances content delivery network for digital media. See `http://www.limelightnetworks.com/`.

[LL08] D. Leonard and D. Loguinov. Turbo King: Framework for large-scale Internet delay measurements. In *Proc. IEEE INFOCOM*, April 2008.

[LL10] D. Lee and W. Lawton. IPv6 at Facebook. Google IPv6 Implementors Conference, June 2010.

[LLB02] Kurt J. Lidl, Deborah G. Lidl, and Paul R. Borman. Flexible packet filtering: providing a rich toolbox. In *Proceedings of the BSD Conference 2002 on BSD Conference*, BSDC'02, pages 11–11, Berkeley, CA, USA, 2002. USENIX Association.

[LPS06] J. Ledlie, P. Pietzuch, and M. I. Seltzer. Stable and accurate network coordinates. In *Proc. International Conference on Distributed Computing Systems*, July 2006.

[LQZ09] Hongbin Luo, Yajuan Qin, and Hongke Zhang. A DHT-based identifier-to-locator mapping approach for a scalable Internet. *IEEE Transactions on Parallel and Distributed Systems*, February 2009.

[MAK$^+$06] Harsha V. Madhyastha, Thomas Anderson, Arvind Krishnamurthy, Neil Spring, and Arun Venkataramani. A structural approach to latency prediction. In *IMC*, October 2006.

[Max02] MaxMind. Geolocation and online fraud prevention from maxmind, 2002. See `http://www.maxmind.com/`.

[MD95] Paul V. Mockapetris and Kevin J. Dunlap. Development of the domain name system. *SIGCOMM Comput. Commun. Rev.*, 25(1):112–122, 1995.

[MEC$^+$11] F. Maino, V. Ermagan, A. Cabellos, D. Saucez, and O. Bonaventure. LISP-Security (LISP-SEC). Internet draft, draft-ietf-lisp-sec-00.txt, work in progress, July 2011.

[Mey08] D. Meyer. The locator identifier separation protocol (LISP). *Internet Protocol Journal*, 11(1):23–36, March 2008.

[MFHK08] A. Matsumoto, T. Fujisaki, R. Hiromi, and K. Kanayama. Problem Statement of Default Address Selection in Multi-prefix Environment: Operational Issues of RFC3484 Default Rules. Internet Draft (Work in Progress) draft-ietf-v6ops-addr-select-ps-05, Internet Engineering Task Force, April 2008.

[MHH10] Michael Menth, Matthias Hartmann, and Michael Hofling. Firms: a mapping system for future internet routing. *IEEE J.Sel. A. Commun.*, 28:1326–1331, October 2010.

[MI08] Laurent Mathy and Luigi Iannone. LISP-DHT: Towards a DHT to map identifiers onto locators. In *ReArch*, December 2008.

[MIP$^+$06] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An information plane for distributed services. In *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 2006. See `http://iplane.cs.washington.edu`.

[MN06] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[Moc87a] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592.

[Moc87b] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343.

[Moc89] P.V. Mockapetris. DNS encoding of network names and other types. RFC 1101, April 1989.

[Mor02] Richard M. Mortier. Internet traffic engineering. Technical report, in IEEE Communication Magazine, 2002.

[Moy98] J. Moy. OSPF Version 2. RFC 2328 (Standard), April 1998.

[MS04] Y. Mao and L. Saul. Modeling distances in large-scale networks by matrix factorization. In *Proc. ACM SIGCOMM Internet Measurement Conference (IMC)*, October 2004.

[MTS$^+$02] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: existing techniques and new directions. *SIGCOMM Comput. Commun. Rev.*, 32:161–174, August 2002.

[MWA02] Ratul Mahajan, David Wetherall, and Tom Anderson. Understanding bgp misconfiguration. In *SIGCOMM*, pages 3–16, August 2002.

[MXZ$^+$05] Xiaoqiao Meng, Zhiguo Xu, Beichuan Zhang, Geoff Huston, Songwu Lu, and Lixia Zhang. Ipv4 address allocation and the bgp routing table evolution. *SIGCOMM Comput. Commun. Rev.*, 35(1):71–80, 2005.

[MZF07] D. Meyer, L. Zhang, and K. Fall. Report from the IAB workshop on routing and addressing. RFC 4984, Internet Engineering Task Force, September 2007.

[MZPP08] R. Mahajan, M. Zhang, L. Poole, and V. Pai. Uncovering performance differences among backbone ISPs with NetDiff. In *Proc. Symposium on Network Systems Design and Implementation (NSDI)*, April 2008.

[NB09] E. Nordmark and M. Bagnulo. Shim6: Level 3 Multihoming Shim Protocol for IPv6. RFC 5533 (Proposed Standard), June 2009.

[NZ02] T. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proc. IEEE INFOCOM*, June 2002.

[NZ04] T. S. E. Ng and H. Zhang. A network positioning system for the Internet. In *Proc. USENIX Annual Technical Conference*, June 2004.

[OBFR09] H. Ould-Brahim, D. Fedyk, and Y. Rekhter. BGP Traffic Engineering Attribute. RFC 5543 (Proposed Standard), May 2009.

[Ora90] D. Oran. OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), February 1990.

[Pap07] V. Pappas. Coordinate-based routing for overlay networks. In *Proc. International Conference on Computer Communications and Networks (ICCCN)*, August 2007.

[PCW+03] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti. Lighthouses for scalable distributed location. In *Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2003.

[Pel06] Cristel Pelsser. *Interdomain traffic engineering with MPLS*. PhD thesis, Université catholique de Louvain, November 2006.

[PFA+10] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. Improving content delivery using provider-aided distance information. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 22–34, New York, NY, USA, 2010. ACM.

[PGES05] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The Bittorrent P2P File-Sharing System: Measurements and Analysis. In Miguel Castro and Robbert van Renesse, editors, *Peer-to-Peer Systems IV*, volume 3640 of *Lecture Notes in Computer Science*, chapter 19, pages 205–216. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2005.

[PLMS06] P. Pietzuch, J. Ledlie, M. Mitzenmacher, and M. Seltzer. Network-aware overlays with network coordinates. In *Proc. IEEE International Conference on Distributed Computed Systems Workshops (ICDCSW)*, July 2006.

[PM08] D. Pezaros and L. Mathy. Explicit application-network cross-layer optimisation. In *The 4th International Telecommunication Networking WorkShop (IT-NEWS) on QoS in Multiservice IP Networks (QoS-IP 2008)*, February 2008.

[Pos81] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[PPZ+08] Abhinav Pathak, Himabindu Pucha, Ying Zhang, Y. Charlie Hu, and Z. Morley Mao. A measurement study of internet delay asymmetry. In *Proceedings of the 9th international conference on Passive and active network measurement*, PAM'08, pages 182–191, Berlin, Heidelberg, 2008. Springer-Verlag.

[PUK+11] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. Ip geolocation databases: Unreliable? *ACM SIGCOMM Computer Communication Review*, 41(2), April 2011.

[QIdLB07] B. Quoitin, L. Iannone, C. de Launois, and O. Bonaventure. Evaluating the benefits of the locator/identifier separation. In *Proc. ACM SIGCOMM MobiArch Workshop*, August 2007.

[Quo06] Bruno Quoitin. *BGP-based interdomain traffic engineering.* PhD thesis, Université catholique de Louvain, August 2006.

[QUP+03] Bruno Quoitin, Steve Uhlig, Cristel Pelsser, C. Pelsser, Louis Swinnen, and Olivier Bonaventure. Interdomain traffic engineering with bgp. *IEEE Communications Magazine*, 41, 2003.

[Rad] Radware. `http://www.radware.com`.

[RD10] J. Rexford and C. Dovrolis. Future Internet Architecture: Clean-Slate Versus Evolutionary Research. *Communications of the ACM*, 53(9):36–40, September 2010.

[Riz97] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1):37–41, January 1997.

[RLH06] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[RMK$^+$08] V. Ramasubramanian, D. Malhki, F. Kuhn, I. Abraham, M. Balakrishnan, A. Gupta, and A. Akella. A unified network coordinate system for bandwidth and latency. Technical Report MSR-TR-2008-124, Microsoft Research, September 2008.

[RR06] E. Rosen and Y. Rekhter. BGP/MPLS IP Virtual Private Networks (VPNs). RFC 4364 (Proposed Standard), February 2006. Updated by RFCs 4577, 4684, 5462.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard), January 2001.

[SAA$^+$99] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zhorjan. Detour: Informed internet routing and transport. *IEEE Micro*, 19(1):50–59, January/February 1999.

[SB00] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, August 2000.

[SB09] J. Seedorf and E. Burger. Application-Layer Traffic Optimization (ALTO) Problem Statement. RFC 5693 (Informational), October 2009.

[SB11] D. Saucez and O. Bonaventure. Performance based traffic engineering with idips. In *Proceedings of ACM SIGCOMM 2011 Demo Session*, August 2011.

[SDB09] Damien Saucez, Benoit Donnet, and Olivier Bonaventure. On the impact of clustering on measurement reduction. In *Networking*, pages 835–846, 2009.

[SDIB08] D. Saucez, B. Donnet, L. Iannone, and O. Bonaventure. Interdomain traffic engineering in a locator/identifier separation context. In *Proc. IEEE Internet Network Management Workshop (INM)*, October 2008.

[SGU08] S. Siwpersad, B. Gueye, and S. Uhlig. Assessing the geographic resolution of exhaustive tabulation for geolocating Internet hosts. In *Proc. Passive and Activement Measurement Conference (PAM)*, April 2008.

[She] Shen. HPTP: Relieving the Tension between ISPs and P2P. *IPTPS'07*.

[SIB09a] D. Saucez, L. Iannone, and O. Bonaventure. Openlisp: An open source implementation of the locator/id separation protocol. In *Proceedings of ACM SIGCOMM 2009 Demo Session*, August 2009.

[SIB09b] D. Saucez, L. Iannone, and O. Bonaventure. Openlisp: An open source implementation of the locator/id separation protocol. In *Proceedings of IEEE INFOCOM 2009 Demo Session*, April 2009.

[SIB11] D. Saucez, L. Iannone, and O. Bonaventure. LISP Threats Analysis. Internet draft, draft-ietf-lisp-threats-00, work in progress, July 2011.

[SMLN$^+$03] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, February 2003.

[SN09] Damien Saucez and Van Nam Nguyen. Lisp-click: A click implementation of the locator/id separation protocol. 1st Symposium on Click Modular Router, November 2009.

[SPPVS08] M. Szymaniak, D. Presotto, G. Pierre, and M. Van Steen. Practical large-scale latency estimation. *Computer Networks*, 52(7):1343–1364, May 2008.

[ST03] Y. Shavitt and T. Tankel. Big-bang simulation for embedding network distances in euclidean space. In *Proc. IEEE INFOCOM*, March 2003.

[Ste07] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007.

[Tea04] Team Cymru. Internet security research and insight, 2004. See `http://www.team-cymru.org/`.

[THKS03] S. Thomson, C. Huitema, V. Ksinant, and M. Souissi. DNS Extensions to Support IP Version 6. RFC 3596 (Draft Standard), October 2003.

[UBQ03] S. Uhlig, O. Bonaventure, and B. Quoitin. Interdomain Traffic Engineering with minimal BGP Configurations. In *Proc. of the* 18th *International Teletraffic Congress, Berlin*, September 2003.

[Uhl04] Steve Uhlig. *Implications of traffic characteristics on interdomain traffic engineering.* PhD thesis, Université catholique de Louvain, March 2004.

[Uni] University of Oregon. Route views, University of Oregon Route Views project. See `http://www.routeviews.org/`.

[UQLB06] Steve Uhlig, Bruno Quoitin, Jean Lepropre, and Simon Balon. Providing public intradomain traffic matrices to the research community. *SIGCOMM Comput. Commun. Rev.*, 36:83–86, January 2006.

[VB.09] VB.com. Domains Counter - Domain Timeline since 1985 by VB.com, 2009.

[WAR07] Y. Wang, I. Avramopoulos, and J. Rexford. Morpheus: Making routing programmable. In *Proc. ACM SIGCOMM Workshop on Internet Network Management (INM)*, August 2007.

[Wis07] Damon Wischik. Short messages. In *Royal Society workshop on networks: modelling and control*, September 2007.

[WSR09] Yi Wang, Michael Schapira, and Jennifer Rexford. Neighbor-specific bgp: more flexible routing policies while improving global stability. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09, pages 217–228, New York, NY, USA, 2009. ACM.

[WSS05] B. Wong, A. Slivkins, and E. G. Sirer. Meridian: a lightweight network location service without virtual coordinates. In *Proc. ACM SIGCOMM*, August 2005.

[XYK$^+$08] H. Xie, Y. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: Provider portal for applications. In *Proc. ACM SIGCOMM*, Agust 2008.

[YRCR04] Ming Yang, X. Rong, Li Huimin Chen, and Nageswara S. V. Rao. Predicting internet end-to-end delay: an overview. In *in Proc. of 36th IEEE Southeastern Symposium on Systems Theory*, pages 210–214, 2004.

[YSB$^+$06] P. Yalagandula, P. Sharma, S. Banerjee, S.-J. Lee, and S. Basu. S3: A scalable sensing service for monitoring large networked systems. In *Proc. ACM SIGCOMM Workshop on Internet Network Measurement (INM)*, September 2006.

[ZCB96] Ellen Zegura, Kenneth Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *In Proceedings of IEEE INFOCOM*, pages 594–602, 1996.

[ZJUVM07] X. Zhou, M. Jacobsson, H. Uijterwaal, and P. Van Mieghem. IPv6 delay and loss performance evolution. *International Journal of Communication Systems*, 21(6), June 2007.