Master thesis : Simulating LISP with NS3

Auteur : Marechal, Emeline
Promoteur(s) : Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en informatique, à finalité spécialisée en "computer systems security"
Année académique : 2018-2019
URI/URL : http://hdl.handle.net/2268.2/6737

University of Liège - Faculty of Applied Sciences

Master Thesis

Simulating LISP with NS-3

A thesis submitted in fulfillment of the requirements for the degree of Master "Civil Engineer in Computer Sciences"

Author:
Emeline Marechal

Supervisor:
Pr. Benoit Donnet

Academic year 2018-2019

Abstract

The *Locator/Identifier Separation Protocol* (LISP) is a novel routing architecture for the Internet that is based on the *locator/identifier separation* paradigm. The principle is to split the current address space into the *identifier* address space, which is composed of locally routable addresses used for identification; and the *locator* address space, composed of globally routable addresses used to route traffic in the network. The main objective is to solve the current Internet's routing scalability problems threatening the network performance. Besides that, LISP also brings several new interesting benefits, in particular for Traffic Engineering (TE), and multi-homing.

Originally, LISP architecture had no support for node mobility. *LISP Mobile Node* (LISP-MN) has thus been developed to introduce mobility extensions to LISP and provide scalable and fast mobility. LISP-MN allows a node to roam from one site to another (whether the site is a LISP site or not), while maintaining ongoing communications at the transport-level. However, there was still no support for sites using Network Address Translation (NAT). Therefore, NAT extensions (LISP+NAT) have been established to define a NAT traversal mechanism for LISP mobile nodes. These extensions enable a node to send and receive LISP traffic, even though it is situated behind a NAT.

So far, little is known about the performance of LISP-MN, and even less about the impact of NAT traversal for LISP traffic. Indeed, both propositions are mainly limited to the theoretical level and lack any experimentation. We aim to provide a first look at this aspect of LISP through simulations on the ns-3 Network Simulator.

To do so, we adapted the existing LISP implementation in ns-3 to add several functionalities to the model. Our main contribution consisted into adding a NAT model, proxy features (interworking mechanism used for communication between LISP sites and non-LISP sites), as well as the NAT extensions (LISP+NAT) to the LISP model. Additionally, we wrote a LISP-MN Helper, meant to help the script writer to easily setup a simulation scenario with mobile nodes and handovers. Finally, several unit tests have been integrated into the ns-3 testing framework for the NAT and LISP models.

After investigation, we saw that all works on NAT traversal for LISP only focused on static scenarios, i.e., scenarios with no roaming and no handover. As such, some important aspects of mobility (the update of remote nodes' state after a roaming event) have been left completely unspecified. We thus propose an extension of the protocol to take handovers into account and define a novel procedure for those cases.

Our results confirm the intuition that NAT traversal has a negative impact on path stretch and on the handover delay. Indeed, most of the time, the handover delay when roaming into a non-LISP site behind a NAT is superior to the handover delay when roaming into a non-LISP site with no NAT deployment.

Acknowledgements

I want to express my gratitude to Benoit Donnet, my thesis supervisor, who introduced me to LISP, and more generally, to the world of research. He was always present to give valuable pieces of advice, and our conversations helped me determine the path I took for my work. For his availability, his guidance throughout the year, and the opportunities he gave me, I am thankful.

I would also like to thank Luigi Iannone for welcoming me in Paris, at Télécom ParisTech. I really appreciated working with him in a new setting, and exchange with him on LISP. He brought new insights on my work, which were much appreciated and stimulating.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1 Context

For many years now, it has been recognized that the Internet routing architecture and addressing system is facing challenges regarding scalability [1]. This scalability issue is reflected by a considerable growth of the default-free zone (DFZ) routing table (a.k.a Routing Information Base, or RIB), resulting in a global negative impact on the network performance [1].
Multiple factors are driving the growth of the DFZ RIB, such as Traffic Engineering (TE), multihoming, and Provider-Independent (PI) addresses. These factors result in the de-aggregation of address prefixes into more specific prefixes, which in turn results in more entries in the RIB.

In order to cope with these scalability issues, a new type of network-level protocol arose, based on the *locator/identifier separation* paradigm [2]. The idea of this paradigm is to separate the *identifier* and *locator* roles of the IP address. Indeed, IP addresses currently have a dual semantic: they represent at the same time the identification aspect, as used by end-points for communication; and the location aspect, as used by the routing system to deliver packets.

The *loc/id separation* paradigm thus introduces two distinct address spaces: the *identifier* address space, composed of addresses that are only locally routable. These addresses are used to identify end-points and have no meaning in the core of the Internet. Then there is the *locator* address space, composed of addresses that are globally routable, and used to route traffic in the network as usual.

Many implementations of the paradigm have been proposed. They can be classified into two categories: those that associate the *locators* directly to the host (HIP [3], shim6 [4]), and those that associate the *locators* to routers (LISP [5]). Among the different proposals, the *Locator/Identifier Separation Protocol*, a.k.a LISP [5], is the most widely deployed.

LISP introduces two address spaces, the EID (*identifier*) address space, and the RLOC (*locator*) address space. EIDs are deployed in stub networks, at the edge of the Internet, while RLOCs are used in the DFZ. It is important to notice that EIDs and RLOCs are syntactically equivalent to IP addresses, but that their semantics have changed. For this reason, few changes are required to deploy LISP. Neither end-points nor routers in the core of the Internet will see any change. Indeed, end-systems will be using EIDs to communicate, as they were using classic IP addresses. And core routers will be using RLOCs to forward packets in the network, as they used to do with IP addresses.

With two different address spaces, a *mapping* between EIDs and RLOCs is necessary to bind the two together. This mechanism is called the *Mapping Distribution System* (MDS) and will provide the bindings between a given EID and a set of RLOCs.

LISP uses *encapsulation* to tunnel packets from one border router to the other. In practice, an end-point that wishes to communicate will form a regular IP packet, with EIDs as source and destination addresses. This packet will be forwarded to the border router, which has been upgraded to support LISP encapsulation. These border routers are called *Ingress Tunneling Routers* (ITR) and are in charge of resolving the mapping of the destination EID by asking the MDS. Once the remote RLOC is obtained, the ITR will encapsulate the packet in a new IP header, with RLOCs as source and destination. This packet will then be forwarded in the core of the Internet, until reaching the remote border router. This router has been upgraded as well to support LISP functionality, and is called an *Egress Tunneling Router* (ETR). The packet will then be decapsulated by the ETR and forwarded to the remote end-point.
The encapsulation part of LISP is commonly called the Data Plane and the mapping part is called the Control Plane.

With the separation of the address space, LISP solves the scalability issues faced by the current Internet architecture. Indeed, as EIDs are only locally routable, they do not need to be advertised in the core of the Internet. Only routes towards the RLOCs of the edge routers need to be maintained. As a result, the size of the DFZ RIB is strongly decreased [6]. Moreover, the BGP churn (i.e., the number of prefixes changed, added, and withdrawn as a function of time) will also be reduced, because the core of the Internet is not exposed to the dynamicity of the edges anymore.
Besides solving the scalability issues, LISP also brings new interesting capabilities regarding Traffic Engineering, multi-homing, and mobility [5].

2 Problem Description

In this master thesis, we consider LISP's approach to mobility, in conjunction with Network Address Translation, or NAT.

LISP Mobile Node (LISP-MN) [7] introduces mobility extensions to LISP and provides scalable and fast mobility. It defines a new network element, the LISP-MN, which implements a lightweight LISP router and allows the node to roam from one site to another, while being reachable under the same EID. In turn, this allows transport-level communications to survive roaming events.

LISP, that divides the address space into EIDs and RLOCs, works on the assumption that LISP routers will always be reachable through their globally routable RLOCs on port *4341* (LISP data port). However, when a LISP device (either a LISP router or a LISP-MN) is situated behind a NAT, this assumption does not hold anymore, as nodes behind a NAT are only given private addresses. The draft (LISP+NAT) [8] introduces LISP extensions for NAT traversal. With it, a LISP device is able to detect if it is located behind a NAT, and take the necessary measures to send and receive traffic nevertheless. The NATed device initialises state on the NAT, and then uses a new network element, the *Re-encapsulating Tunnel Router* (RTR) to forward traffic to and from other LISP devices through the NAT.

Both LISP-MN [7] and its NAT extensions (LISP+NAT) [8] are at the early development stage, and little is known about how they behave in the field. However, with the increase of mobile devices in today's Internet, as well as the desire to have continuous network connectivity at any moment, it is important to evaluate this kind of traffic. In particular, evaluating the impact of NAT traversal on LISP traffic is also crucial, as it corresponds to a majority of situations where a user is browsing the Internet at home, behind an operator box.

3 Contributions

Up to now, no investigation of the impact of NAT traversal for LISP traffic has been conducted, to the best of our knowledge. In this master thesis, we aim to provide a first look at this aspect of LISP through simulations on the ns-3 Network Simulator [9]. More precisely, we evaluate several handover scenarios between different types of site (LISP, non-LISP, non-LISP behind NAT) and provide a comparison of the handover delay, as well as the handover overhead.

To perform these simulations, the ns-3 Network Simulator has been adapted to support the different aspects of the scenarios. Our contribution to the LISP implementation in ns-3 consisted into adding a NAT model, proxy functionalities, and NAT extensions to the existing LISP model. Additionally, we also wrote a LISP-MN Helper to help the script writer deal with the setup of a mobile node, without having to worry about the low-level implementation details. Finally, various unit tests for different scenarios (both NAT- and LISP-related) have been added into the ns-3 testing framework.

The different works on NAT traversal for LISP traffic only cover static scenarios, i.e., scenarios in which there is no roaming and no handover. In particular, a certain procedure, called the SMR procedure, used to update the remote nodes' state after a roaming event, is left completely unspecified. Therefore, we propose an extension to the protocol to allow a LISP-MN to update the remote nodes' state, even when it is situated behind a NAT.

Our implementation in ns-3 respects the NAT extensions (LISP+NAT) [8], with the addition of this novel SMR procedure.

4 Results

The results of our simulations highlight the negative impact of NAT traversal on LISP traffic, compared to classic scenarios with no NAT deployment. From a theoretical point of view, the control message overhead is higher for NAT traversal, with at least 1172 bytes of control messages, compared to 612 bytes with no NAT deployment. From the simulation itself, we also observe a larger handover delay most of the time, as expected, due to the complexity introduced by NAT traversal mechanisms.

Having compared handovers towards NATed sites with handovers towards non-LISP sites, we also conduct a simulation with a handover towards a LISP site, in order to have a reference for the negative impact introduced by NAT traversal. The results show that most of the time, the handover delay is higher for LISP sites than for NATed sites. This is a surprising result, as we expected NAT traversal to have more impact on the handover delay. However, we explain this by the fact that simulations are inherently limited, compared the real experimentation, and that even though we took great care to setup a realistic scenario, some effects could not be taken into account with the available data we used for the simulations.

5 Roadmap

This master thesis is organized as follow. In Chapter 2, we present the scalability issues the current Internet architecture and addressing system is facing. We review the reasons behind this scalability problem as well as the implications for the network performance.
In Chapter 3, we introduce the *Locator/Identifier Separation Protocol*, or LISP [5], and explain its design. We also present the benefits of LISP: how it solves the scalability problem encountered in

today's Internet, and what extra routing capabilities it brings to networks, especially for Traffic Engineering and mobility.
Chapter 4 first presents how LISP has been extended to provide mobility features to the protocol, and allow a mobile node to roam in and out of sites, while maintaining ongoing communications at the transport-level. Then, we have a look at NAT considerations for LISP traffic and review the NAT extensions added to the protocol in order to enable a LISP device to send and receive traffic, despite being NATed. Finally, we present the novel SMR procedure we defined for handovers.
In Chapter 5, we briefly present the ns-3 Network Simulator and its LISP model. Then we detail our implementation of NAT, proxies, and LISP+NAT, as well as the unit tests and the Helper for mobile nodes.
Finally, Chapter 6 studies several handover scenarios between different types of site. We detail our methodology for simulating a realistic network, the simulation setup itself, as well as the limitations of the simulations. Then we show the results of our ns-3 simulations and draw conclusions about NAT traversal for LISP traffic.

Chapter 2

Motivations for a new network-level protocol

For many years now, it has been recognized that the Internet routing architecture and addressing system is facing challenges regarding scalability [1]. In this chapter, we will review the different scalability problems with the Internet current implementation. More precisely, we will see that the scalabilty issue we are currently facing in the network is reflected by a considerable increase in the size of the DFZ routing table (a.k.a the Routing Information Base, or RIB) [1]. We will survey the different factors having an effect on the RIB size, as well as the impact of such a growth. Then we will talk about the underlying problem of this scalability issue.

1 Problem #1: Scalability of the routing system

The fast growth of the DFZ routing table is a major concern for the current routing architecture. In 2005, the number of entries was approximately 150,000. In 2010, it was 300,000, and as of 2018, it has now reached 700,000 entries, confirming a more than linear growth of the table size [10].

There are multiple factors driving the growth of the RIB. There is of course the general increase in the Internet size and its user population, but there are also other reasons, such as multi-homing, Traffic Engineering, business events (such as acquisitions), etc. The underlying component to all these factors is the addition of more prefixes to the table.

Some of these additions are "natural" and result only in the growth of the Internet. Indeed, as new sites are connected to the network, new entries must necessarily appear as well.
However, a large portion of the growth comes from the de-aggregation of address prefixes. By de-aggregation, we mean that address prefixes are split into more specific prefixes, resulting in more entries than initially. This is in fact contradictory to the principle of BGP, which is to topologically aggregate prefixes together in order to have fewer entries.

1.1 Reasons of the DFZ routing table size increase

Let us review in more details the reasons for this de-aggregation of address prefixes. In fact, it appears that the more specific entries found in the DFZ are the result of deliberate actions from operators.

- **Traffic Engineering**:

Traffic Engineering (TE) refers to a set of techniques for influencing where traffic goes for the sake of network performance and traffic delivery optimization [11]. TE can be used to balance the load over multiple links in order to adapt the traffic to capacity or congestion in the network. It can also be used to reduce the cost by re-orienting the traffic towards lower-cost paths. Additionally, TE can be employed to apply some policies, for example, avoiding or privileging certain routes because of political reasons.

Traffic Engineering is crucial for providers to guarantee good performances in their network and if they want to stay competitive and increase their benefit. However, TE is not an easy feast, and network operators do not have a lot of means to achieve their routing goals. One way to proceed is by crafting specific BGP advertisements to serve one's purpose: by advertising more-specific prefixes, that is to say, by de-aggregating some address range, they are able to discriminate between traffic and adjust it on the different links.

Obviously, applying TE in this way results in an increase of the DFZ routing table size.

- **Avoiding renumbering and PI addresses**:

  Renumbering a network means that IP addresses of devices in the network will be changed. This renumbering process can be quite cumbersome, as it may require a certain number of actions. For example, it may be necessary to change host parameters and configuration files, such as files which contain addresses of DNS and other servers. In reality, renumbering can become effectively impossible for an organization.

  For this reason, customers usually choose to adopt PI (Provider-Independent) addresses for their network, which helps them avoid the need to renumber and gives them extra flexibility. PI addresses are globally unique addresses which are not assigned by a provider, but rather by some other organization, such as a Regional Internet Registry (RIR). This means that these blocks of addresses are not associated with any particular location in the network and are thus not topologically aggregatable in the routing system.

  Therefore, routing PI prefixes means that non-aggregatable prefixes will be injected in the table. Consequently, many more entries will be added to the RIB, further increasing the growth issue.

- **Multi-homing**:

  A network that is multi-homed is a network that is connected to several providers. Multi-homing is generally employed to increase connectivity reliability, as well as performance, but it will have an impact on the DFZ routing table, as we will see.

  Multi-homing can be achieved using either PI addresses, or PA (Provider-Aggregatable) addresses. In a way, PA addresses are the opposite of PI addresses: they are assigned by a provider and the block of addresses is a sub-block of the address range of the service provider, resulting in addresses that can be aggregated into the larger block before being advertised into the global Internet [12].

  With PI prefixes, the problem is the same as described earlier: non-aggregatable prefixes will be injected in the tables, and this will be the case for all of the providers of the multi-homed site.

  With PA prefixes, even though these addresses are in theory aggregatable in the 'primary' provider address space, the provider won't be able to effectively aggregate them if it wants its multi-homed site to still be reachable through itself. In other words, for a site to remain

accessible through all of its providers, its address prefixes must necessarily appear in the global routing table.

Let's look more precisely at the reasons for this phenomenon: we know that the customer's prefixes will not be aggregatable for the other providers (as the prefixes are not part of their address space). Because of this, the other providers have to advertise these specific prefixes.
As a result, because of the longest-matching prefix forwarding rule, the customer's traffic will be directed through the non-primary providers. To counter this, the 'primary' provider has no choice but to de-aggregate the customer's prefix in order to keep the customer's traffic flowing through itself, instead of the non-primary providers. If the 'primary' provider doesn't de-aggregate, the site won't be reachable anymore via the 'primary' provider.

To cut a long story short, whether using PA or PI addresses, multi-homing will ruin topological aggregation and leads to a fast growth of the routing tables.

- **IPv6**:

  We saw that the network is already facing serious scalability issues with the current IPv4 address space. It is easy to envision that the RIB and FIB size growth problem of today will be further aggravated by IPv6 much larger address space, if we remain without a more scalable approach to networking.

1.2 Implications of the DFZ routing table size increase

Now that we have reviewed the factors driving the growth of the RIB size, let us see the implications of such a growth. The problem is divided into two aspects: the growth of the Routing Information Base, and the resulting growth of the Forwarding Information Base (FIB).

The RIB and the FIB are conceptually close to each other, but there are differences. The RIB is the table used to store all routing information. It is also independent of the routing protocol: each time that a routing protocol learns a new route, it is injected into the table.
The FIB however is used for forwarding exclusively. It contains the best route toward a destination prefix, whereas the RIB cannot be used directly for forwarding, as it sometimes contains entries with next hops that are not directly connected to the device. That way, the forwarding plane is separated from the control plane.

- **Implications of the RIB growth**:

  It goes without saying that the RIB growth will result in a larger amount of memory required to store it. But this is not the main concern. This prefix de-aggregation used by providers for Traffic Engineering and multi-homing means that these addresses, situated at the edge of the network, will end up having an impact on the core of the network. In other words, the core of the network will be exposed to the dynamic nature of edges.

  Injecting new routes into the DFZ routing table (with BGP UPDATE messages) often results in a re-computation and redistribution of the FIB, which has a certain cost. The problem is that this whole process will occur much more frequently, as the BGP UPDATE churn (that is to say the number of prefixes changed, added, and withdrawn as a function of time) increases.

  The size of the DFZ routing table is currently bounded by the address space, however, the number of BGP UPDATE messages, used to propagate dynamic topology changes, is not.

As a consequence, the routing convergence will be impacted and become slower, because of the frequent re-computations of the RIB.

- **Implications of the FIB growth**:

  As new routes are injected into the system, the forwarding table necessarily grows along with them. Thus, as the amount of routing information that needs to be handled becomes more important, so does the hardware capacities for forwarding.
  The concern is that the costs induced by a rapid technology renewal (in order to keep up with the FIB size) won't be constant, but will become more and more expensive as we go [1].

  Another element is that hardware capacities for forwarding are simply and bluntly limited, and it may be possible that technology will not meet the expectations for routing in the core of the Internet, in terms of heat, power consumption, and heat dissipation.

We now understand better the scalability issues faced by the current routing architecture and how the resulting RIB and FIB growth can impact the performance of the network.

2 Problem #2: The underlying issue

In Section 1, we saw that the scalability problem in the current routing architecture is reflected by a considerable increase of the DFZ routing table size. There are several reasons to this, including, but not limited to, Traffic Engineering, multi-homing, and non-aggregatable address allocations. The concerns about such a growth are numerous: more resources are needed for maintaining state, for computations and for routing traffic in the core of the Internet, resulting in higher costs. The BGP convergence is also endangered with the BGP UPDATE churn increase, overall threatening the performance of routing in the network.

Upon closer inspection, we can notice that there is a recurring problem to all the reasons for the RIB growth: the overloading of the IP address semantic. By overloading, we mean that an IP address has several roles. Indeed, if we look at the semantic of the IP address, we can find two elements:

1. An identification aspect. This is the semantic of the "Who", as used by end-points in the transport layer.
2. A location aspect. This is the semantic of the "Where", as used by the routing system to effectively deliver packets.

Because of this dual semantic, we are faced with two conflicting objectives. In the first place, for routing to be efficient and to have as few entries in the table as possible, addresses must be assigned topologically, in order for them to be aggregatable.
But on the contrary, from an organization and management point of view, addresses shouldn't be bound to the topology, which is quite dynamic, because the desired property here is stability. Indeed, organizations don't want to be forced to renumber because of topological changes.

Therefore, this overloading of the IP address semantic is the underlying cause of the scalability problem, because the same address space can not serve both purposes efficiently. The *locator/identifier overload* problem thus naturally leads to a *locator/identifier separation protocol*, a.k.a LISP.

Chapter 3

Locator/Identifier Separation Protocol

Because of the scalability issues faced by the Internet routing architecture and addressing system, the need for a new network-level protocol arose, leading to the creation of the Locator/Identifier Separation Protocol, or LISP [5]. In this chapter, we will first present the principles and mechanisms of LISP. Then in a second time, we will see the benefits of LISP: how it solves each of the problems presented in Chapter 2 Section 1, and what extra capabilities it brings to network routing.

1 LISP Overview

In this section, we will see the functioning of the LISP protocol. We review the basic principles of LISP by presenting a classic communication scenario between two end-points in the Internet. In addition, we see the encapsulation format of LISP. We also introduce the LISP Mapping System, a key component in the workings of the protocol. Finally, we present the interworking mechanism used for communication between LISP sites and non-LISP sites.

1.1 Operating mode of LISP

As explained in Chapter 2 Section 2, the dual role of the IP address is the inherent problem regarding the exponential growth of the routing tables. Therefore, to solve this underlying problem, the defining characteristic of LISP, as the name indicates, is to separate the *identifier* and *locator* roles of the IP address. LISP thus introduces two different address spaces:

- The End-point Identifier (EID) address space, composed of IP addresses that are only locally routable. These addresses are used to identify end-systems, and are assigned independently from the network topology.
- The Routing Locator (RLOC) address space, composed of IP addresses that are globally routable. These are used to localize the end-systems in the topology, and route traffic through the network.

An example of how LISP is deployed and where the different address spaces are used can be found in Figure 3.1.

EIDs and RLOCs are syntactically equivalent to IP addresses, but the semantics behind them has changed. For this reason, few changes will be required to deploy LISP. Neither end-systems nor routers in the core of the Internet will see any change. Indeed, end-systems will be using EIDs to communicate, as they were using classic IP addresses. And core routers will be using RLOCs to forward packets in the network, as they used to do with IP addresses.

Now, having introduced two different address spaces, a *mapping* between the two is necessary to allow communication. Indeed, when two end-systems wish to communicate over the network, they cannot directly use their EIDs, as they have no meaning in the core of the network. A tunnel from the RLOC of the source EID to the RLOC of the destination EID is necessary. The Mapping System, which is a key component of LISP, will provide the necessary binding between a given EID and a set of RLOCs.

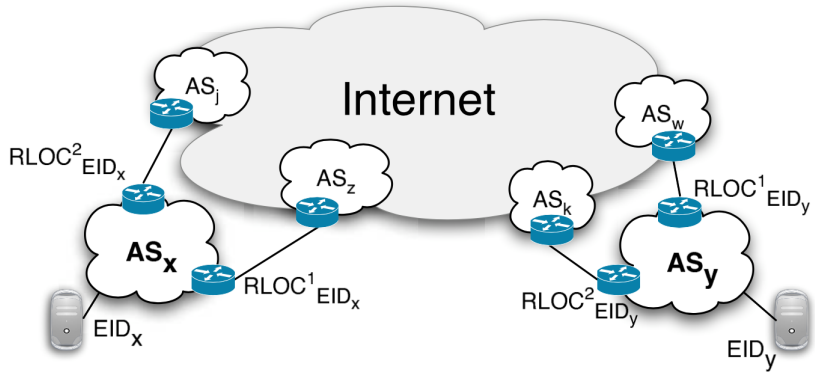


Figure 3.1: Lisp topology
[6]

To wrap all elements together, let us review the procedure when sending a packet from one domain to another, as can be seen on Figure 3.1:
When EID_x wishes to establish a connection with EID_y, a standard IP packet, using both EIDs as source and destination, is created. This packet will be forwarded according to AS_x policy to one of the edge routers ($\text{RLOC}^1_{\text{EID}_x}$ for example), which are called *Ingress Tunneling Routers* (ITR), where the encapsulation will occur. The ITR will perform a look-up of the Mapping System, looking for a set of potential RLOCs associated with the destination EID. Notice that multiple RLOCs (either $\text{RLOC}^2_{\text{EID}_y}$ or $\text{RLOC}^1_{\text{EID}_y}$) can be associated to one EID (EID_y), opening the door to TE capabilities, of which we will talk later.
Once a suitable RLOC ($\text{RLOC}^1_{\text{EID}_y}$ for example) is found for EID_y, the packet is encapsulated and forwarded as usual in the core of the network. This encapsulation mechanism simply consists in adding a new IP header with $\text{RLOC}^1_{\text{EID}_x}$ and $\text{RLOC}^1_{\text{EID}_y}$ as source and destination. Once the packet arrives at the edge router of the destination domain, also called the *Egress Tunneling Router* (ETR), it is decapsulated and forwarded to the destination EID, i.e. EID_y. The encapsulation part of LISP is commonly called the Data Plane, and the Mapping System is called the Control Plane.

We can have a look at Figure 3.2 to get a more precise view of the encapsulation and decapsulation occurring between two LISP sites. When the original packet arrives a the ITR, it is encapsulated in a new IP header with the RLOCs of the xTRs as source and destination addresses. Once it arrives at the ETR, it is decapsulated, and the original packet is forwarded towards its destination.

With this system, only edge routers (xTRs), situated at the boundary between the EID and

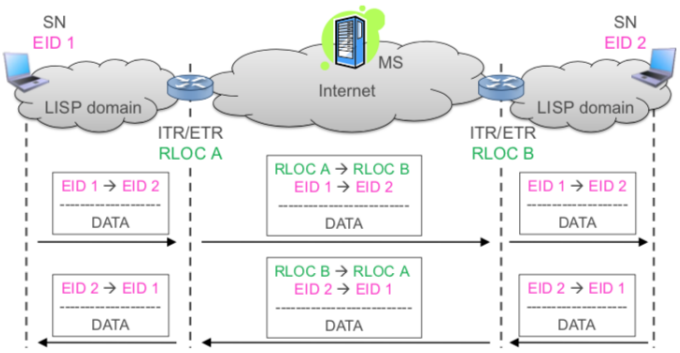


Figure 3.2: Communication between two LISP sites
[13]

the RLOC address space, will need to be upgraded to support LISP and be able to perform all actions associated with encapsulation.

1.2 LISP encapsulation

LISP is an encapsulation protocol that tunnels packets from one xTR to the other in the core of the Internet. To do so, the ITR prepends an IP header to the packet with the RLOCs of the xTRs as source and destination. However, the ITR cannot just add an IP header, as a lot of middle boxes will only forward packets with a classic structure, that is to say IP packets that contain either UDP or TCP.

The format of a packet that has been encapsulated by an ITR towards an ETR can be found in Figure 3.3. The original IP packet is composed of the inner IP header, that uses the EIDs of the two end-points. Then, a LISP header is first appended to the original packet (to carry LISP specific information). Then it is followed by the UDP header, that allows LISP packets to be processed by middle boxes in the Internet. The destination port is set to 4341, which is defined as the LISP data port. Finally we find the outer IP header, that uses the RLOC addresses. For further details about each field, the reader can refer to the LISP RFC [5].

1.3 The EID-to-RLOCs Mapping System

The Mapping System of LISP is truly the core of the protocol. It will determine whether LISP is performing efficiently or not, whether it brings benefits or not. The Traffic Engineering capabilities of LISP will also be determined by the EID-to-RLOCs mappings available.

This Mapping System has voluntarily been designed as a separate module in order to facilitate experimentation with different database designs. To hide the implementation details of the Mapping System to xTRs, they will only be communicating with a service interface, composed of *Map Resolvers* and *Map Servers*. Those two types of devices will be the front-end of the Mapping System for xTRs.

- **Service Interface**:

  LISP operates in a pull mode, that is to say that mappings are retrieved on demand by the ITR. More precisely, when an ITR needs a mapping between an EID and a set of RLOCs, it will send a MapRequest to a *Map Resolver*, whose responsibility is to resolve the mapping.

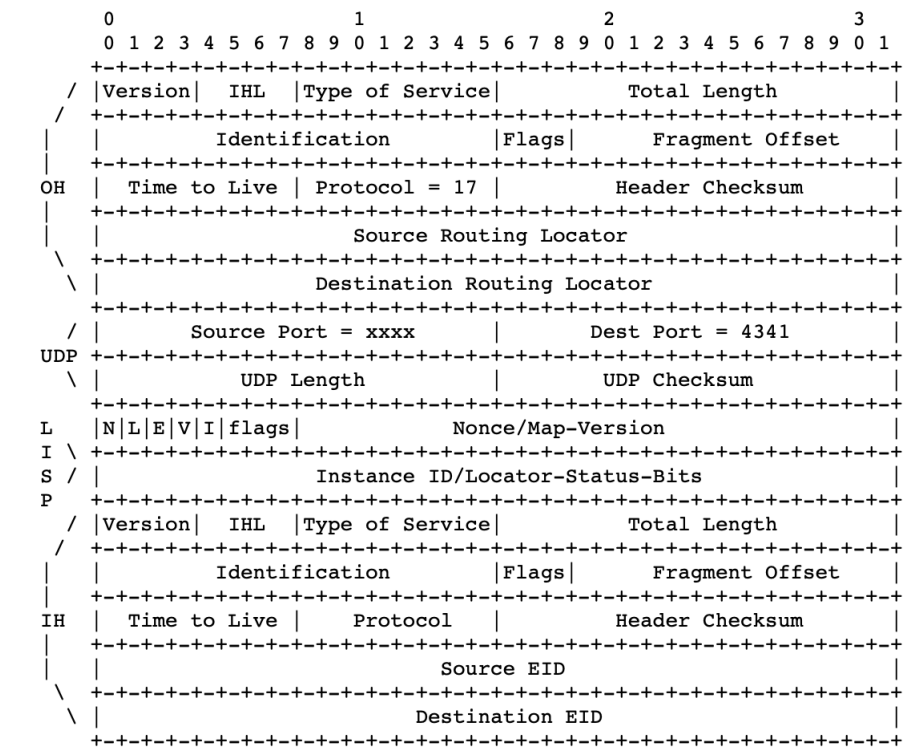


Figure 3.3: Lisp Header format
[5]

Behind the curtains, the *Map Resolver* will actually query the LISP database to find the authoritative ETR responsible for the requested EID. On the other hand, the *Map Server* will learn different mappings from ETRs and publish them in said database.

More precisely, when a *Map Resolver* receives a MapRequest, it will forward the request towards the database, leaving the source address of the ITR unchanged. This means that *Map Resolvers* will not respond to ITRs that sent MapRequests, but rather, they will only forward the requests to a LISP device (either an ETR or a proxy *Map Server*). While the request is being processed, packets for the requested EID are dropped. It is therefore of prime importance for the Mapping System to have good performance.

- **Implementation:**

  As of now, multiple architectures to implement such a distributed database of EID-to-RLOCs mappings have been proposed, but only two have been deployed: *LISP Alternative Topology* (LISP+ALT) [14] and *LISP Delegated Database Tree* (LISP-DDT) [15]. LISP+ALT was the first Mapping System to be deployed, but it was quickly found to be unmanageable and was replaced with LISP-DDT.

  LISP-DDT has been designed with high scalability in mind and is organized as a hierarchical distributed database, mirroring a DNS-like architecture. Each node of the database (called a DDT node) is responsible for a part of the EID address space, where the hierarchy mirrors the hierarchy of the address space. In other words, a child node is only responsible for a sub-part of its parent address space, and the parents maintain a list of all DDT nodes to which they delegated some sub-prefixes. The leaves of the database tree are made of *Map Servers*, which contain the mapping information.

Therefore, like for DNS, a *Map Resolver* seeking a particular binding will query this architecture by starting from the root, and will be redirected from one DDT node to another, until finally reaching the *Map Server* authoritative for the requested EID.

LISP mappings in an xTR can be found in two different places: the LISP *Cache*, and the LISP *Database*. The LISP *Cache* is populated with mappings retrieved from the LISP Mapping System on demand, when a packet from a new flow arrives at the ITR, and that no mapping is found for the requested destination EID. These mappings are used by the ITR to encapsulate packets towards the right ETR. They are removed from the Cache when not used after a certain timeout. The LISP *Database* on the other hand is populated by configuration and is mainly static, contrary to the LISP *Cache*, which is dynamic. The LISP *Database* stores mappings for the EID prefixes it is authoritative for. This is used by the ETR to answer to MapRequests, and this is used by the ITR to select the source RLOC when encapsulating packets.
The complete process when a packet is sent from one LISP site to another can be found on Figure 3.4.

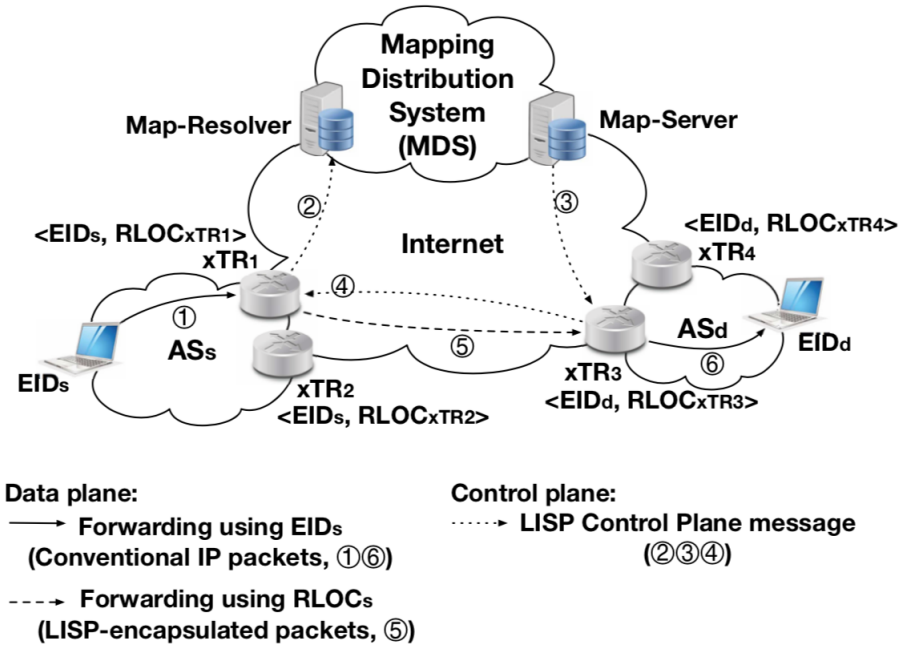


Figure 3.4: Lisp architecture
[16]

1. A classic IP packet with EID source and destination addresses arrives at the ITR.
2. A MapRequest message is sent to a *Map Resolver* for EID_d.
3. The MapRequest message is forwarded towards the authoritative ETR for that EID, thanks to the Mapping System.
4. A MapReply is sent back to the requesting ITR.
5. The IP packet is encapsulated with RLOC source and destination addresses.
6. The packet is decapsulated and forwarded towards its destination.

1.4 Interworking with legacy Internet

LISP design is such that only border routers, at the boundary between the EID and the RLOC address space, need to be upgraded to support LISP. However, it is unlikely that LISP will be deployed uniformly and at the same time all over the world. Therefore, we need an interworking mechanism to allow LISP sites to communicate with non-LISP sites and inversely. This mechanism introduces two new network elements: The *Proxy Ingress Tunnel Router*, or PITR, and the *Proxy Egress Tunnel Router*, or PETR [17]. These new devices are situated outside the edge domains, in the core of the Internet. Both are also known as PxTRs.

The PITR will act as an ITR for non-LISP sites, allowing packets to be encapsulated towards LISP sites. The PETR will act as an ETR for non-LISP sites, allowing packets to be decapsulated and forwarded natively (that is to say, without encapsulation) in the Internet. Both elements allow interworking without any change required at the non-LISP site.

Proxy Ingress Tunnel Router

In order to encapsulate legacy Internet traffic towards LISP sites, the PITR first needs to attract that traffic towards itself. To do so, the PITR will advertise in the core Internet portions of the EID space with BGP, so that packets destined to a LISP site can be routed to the PITR. Once the traffic arrives at the PITR, it will perform the same processing as a classic ITR, i.e., query the Mapping System with a MapRequest to find the associated RLOC of the destination EID and encapsulate the packet towards the returned RLOC.

The announcements made into the DFZ must be highly aggregated in order not to lose the first benefit of LISP: the reduction of the DFZ routing table size. Indeed, the advantage of LISP is that the EID space of a LISP site will not be advertised into the core network anymore. However, PITRs will still advertise that same EID space in order to attract legacy traffic. Therefore, the EID spaces advertised must be highly aggregated in order not to inject too specific routes into the DFZ table. The placement of PITRs in the network is also extremely important from a path stretch point of view. PITRs should be positioned close to non-LISP sites in order to reduce as much as possible the path stretch to reach the PITR [17].

Proxy Egress Tunnel Router

PETRs are used by LISP sites that send traffic towards non-LISP sites in the case where the ITR cannot natively forward packets in the core Internet for policy reasons. When an ITR realises that the destination address is not part of a LISP site (with a negative MapReply), it can either directly forward the original packet, or it can encapsulate it towards a configured PETR.
If the original packet is directly forwarded without encapsulation, it would find its destination because the destination address is not part of a LISP site, hence it is globally routable. In some cases however, the provider AS doesn't allow the original packet to be directly forwarded because the source is part of the EID space, and therefore not globally routable. This is a source address filtering meant to avoid IP spoofing and that is usually implemented by a unicast Reverse Path Forwarding (uRPF) [18] check in the provider Edge Routers. And because the EID source address is not part of the address range of the provider AS, the packet would be dropped. In those cases, the packet must be encapsulated towards a PETR that will act as an ETR for the non-LISP site and decapsulate the packet. Again, to avoid path stretch, the ITR should be configured with a PETR close to it.

Let us review on Figure 3.5 the packet flow for communication between a LISP site and a non-LISP site. When the packet towards IP address 11.13.12.5 arrives at the ITR, it issues a MapRequest for that address and receives a negative MapReply because address 11.13.12.5 is not part of a LISP site. Therefore, it encapsulates the packet towards the PETR with RLOCs A and E in outer header. When the PETR receives the packet, its role is to decapsulate it and forward it natively to the non-LISP site.
In the opposite direction now, the packet towards IP address EID1 will be routed towards the PITR thanks to the advertisements made by the PITR in the DFZ for that EID space. The PITR will then issue a MapRequest, as an ITR would do, and encapsulate the packet with RLOCs A and I in outer header. The packet then arrives at the ETR and is decapsulated and forwarded towards node SN.

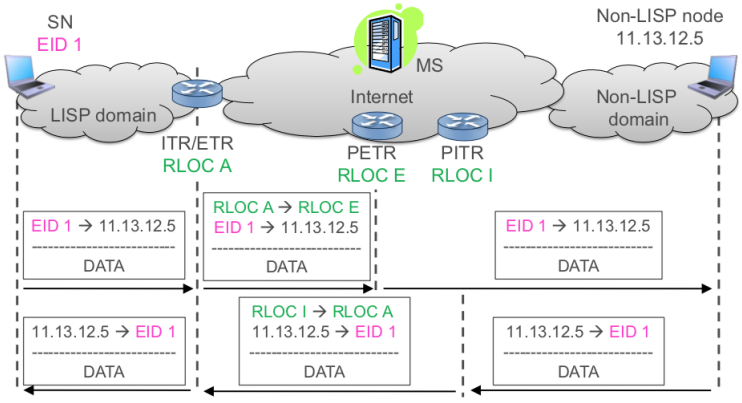


Figure 3.5: Communication between LISP and non-LISP sites [13]

2 LISP Benefits

Besides the fact that LISP already solves the scalability problems in the current Internet architecture, it also offers new interesting capabilities regarding Traffic Engineering, multi-homing, and mobility [5]. Another key component of LISP is that it is an IP-over-IP tunnelling protocol, meaning that it can be incrementally deployed, without the need to modify end-systems or core routers, as already explained in Section 1.
It is commonly agreed ([6], [2]) in the Internet community that this separation of *identifier* and *locator* roles will be essential in the future of the Internet, and LISP appears to be the more promising solution.

2.1 Reduction of the DFZ routing table size

The first benefit of LISP is a decrease of the DFZ routing table size. Indeed, as EIDs are only locally routable, core routers don't need to maintain any route towards them, but only towards the edge routers of those domains, already decreasing the number of entries. The BGP UPDATE churn will also be reduced since fewer routes need to be advertised, and since the core is not exposed to the dynamicity of the edges anymore.

Moreover, by dividing the address space into distinct parts, one for identification (EID), and another one for localization (RLOC), it is now possible to organize addresses efficiently in both of them.

- EID addresses are only locally routable, meaning that they can be assigned independently from the network topology. In other words, the sites are now decoupled from the core topology. This allows to solve numerous problems, such as renumbering and PI addresses.

  Indeed, because these addresses are not bound to the topology anymore, EIDs can be assigned in any manner fitting the requirements of the network operators: they can be assigned hierarchically, to facilitate organization and management from an administration point of view. Or they can also be structured in a manner suitable for local routing within the site.
  As a result, network mobility is facilitated because sites can change provider without the need to renumber, which also means that using PI addresses is not a concern anymore.

- Furthermore, the use of a separate address space for RLOCs allows these addresses to be assigned topologically, meaning that they will be highly aggregatable. In turn, the number of entries in the routing tables will be strongly reduced, resulting in a routing that is more efficient. At the same time though, the customer's freedom to change their provider remains unrestrained.

All in all, dividing the address space into two distinct parts enables for management to be easier on one side, and for routing to be more efficient on the other side, thanks to the reduction of the RIB size in core routers. Studies have shown that, when using only aggregatable addresses, the number of entries has an order of magnitude less than the number of ASes, while in the current Internet the number of entries has an order of magnitude larger than the number of ASes [6].

2.2 New routing capabilities

Traffic Engineering

Traffic Engineering is a major concern for ASes because of economic reasons. Network operators must take care to optimize their traffic to make the largest possible profit. Currently, inter-domain Traffic Engineering is not simple to achieve with the means that are available. It can be performed with MPLS, with segment routing, or with BGP attributes, among other techniques. In Section 1, we also saw that TE can be achieved with BGP and de-aggregation, resulting in an artificial growth of the DFZ routing tables.

Fortunately, the tunneling capabilities of LISP will offer more flexible Traffic Engineering capabilities and allow to take advantage of path diversity, without adding state into the routing system. Indeed, due to the Mapping System of LISP, which can associate multiple RLOCs to a single EID, a source can now choose among several paths to reach its destination.

Indeed, LISP offers the possibility for a site to control incoming traffic by manipulating its mappings. In fact, each RLOC in a mapping is associated with a priority and a weight. This information can thus be used by the ITR to decide which RLOC to send packets to: the RLOC with the highest priority will be selected, and in case of equal priorities, traffic will be balanced among the different RLOCs according to the weights specified.
Therefore, it is possible to use LISP to route different types of traffic on different paths having different capacities, by tuning the list of RLOCs, along with their weights and priorities [19]. Moreover, a mapping-owner can even differentiate its answers to MapRequests depending on the author of said requests, as the EID source is present in the request.

With this newly acquired freedom, alternative paths can be selected based on whatever criteria, such as delays for example. Obviously, not all paths are of the same quality, and choosing

one RLOC over another can have an impact on the traffic performance. For example, studies [6] show that there can be large variations in the delays depending on the chosen path, with differences between the worst and best case larger than 100ms. Even though delays are not the only metric that can be optimized, LISP already shows the possible benefits that can be achieved.

Mobility

End-point mobility occurs when an end-point moves relatively rapidly, therefore changing its IP-layer network attachment point. During such an event, maintenance of ongoing communications at the transport-layer is a primary goal, but requires special mechanism to handle the change of IP address. Indeed, as transport-level communications are defined by the IP addresses of the two end-points communicating, the connection cannot survive a change of IP address. Special mechanisms to deal with mobility already exist, such as Mobile IP [20]. However, these techniques often come at the cost of additional complexity, as well as triangular routing, which is not desirable.

However, with LISP address space separation and Mapping System, mobility capabilities will come naturally with the protocol. We will review this aspect in more details in Chapter 4.

Chapter 4

LISP Mobility and NAT

With the increase of mobile devices in today's Internet, it is important to ensure the ability for nodes to move from one IP subnet to another (a.k.a *roaming*), while being able to maintain ongoing communications at the transport layer. In this chapter, we will review how mobility is achieved with LISP, thanks to the introduction of a new network element, the LISP-MN. Furthermore, we review the particular case where a mobile node roams behind a NAT. This scenario is important as it corresponds to a majority of situations where a user is browsing the Internet at home, behind an operator box. Additionally, we present a refinement of the protocol considering the SMR procedure when a LISP device is NATed.

1 Mobility

To handle mobility in IP, special mechanisms are required. Indeed, when a node roams into a new network, it changes its attachment point to the Internet, and thus receives a new IP address. As such, transport-level connections cannot be maintained, because they are defined by the IP addresses of the two end-points that are communicating.

Special mechanisms already exist to deal with mobility. Mobile IP [20], for example, defines a mechanism which enables nodes to change their point of attachment to the Internet without changing their IP address, therefore allowing transport-level connections to survive. Mobile IP defines new elements, such as the home agent, the foreign agent, the permanent address, the care-of-address, etc; all of which allow the mobile node to exchange packets while being away from home. However, this mobility mechanism comes at the cost of additional complexity in the IPv4 protocol. Moreover, a major downside is the indirect triangular routing that it causes, where packets are routed through the home agent while the mobile node is roaming, resulting in a path stretch that is often non negligible [21].

Generally, the following requirements are desirable when designing a mobility protocol [7]:

- Allowing TCP connections to stay alive while roaming.
- Allowing the mobile node to communicate with other mobile nodes while either or both are roaming.
- Allowing the mobile node to multi-home (i.e., use multiple interfaces concurrently).
- Allowing the mobile node to be a server. That is, any mobile node or stationary node can find and connect to a mobile node as a server.

- Providing shortest path bidirectional data paths between a mobile node and any other stationary or mobile node.
- Not requiring fine-grained routes in the core network to support mobility.
- Not requiring a home agent, foreign agent or other data plane network elements to support mobility. Note since the LISP mobile node design does not require these data plane elements, there is no triangle routing of data packets as is found in Mobile IP [20].

1.1 LISP-MN

LISP Mobile Node (LISP-MN) [7] introduces mobility extensions to LISP, providing scalable and fast mobility. It defines a new network element, the LISP-MN, which implements a light-weight xTR and allows the node to roam in and out of LISP sites. LISP-MN answers all of the requirements presented above. Indeed, with LISP address space separation and encapsulation, these capabilities come naturally with the protocol: the EID address is permanent, and allows transport connections to survive roaming events, while the RLOC changes when the device roams into another network and receives a new IP address.
LISP-MN leverages on three existing components: a classic LISP implementation running on the mobile node, *Map Servers*, and interworking mechanisms (PxTRs).

The Mobile Node will act and look like a complete LISP site. Each time it receives a new RLOC for roaming into another network, it will issue a MapRegister to its assigned *Map Server* to register it, since it is the authoritative ITR for its EID. The packets originating from the LISP-MN will already be encapsulated, with the inner header containing the EID, and the outer header containing the received RLOC. It will also issue MapRequests for traffic it needs to encapsulate. Additionally, LISP-MN doesn't suffer from triangular routing, as in Mobile-IP.
LISP-MN EIDs are provisioned from specific blocks reserved for mobiles nodes, so that they don't overlap with other EID spaces. Things are envisioned to work much like a subscription to a telephony company: users receive their EID for the whole subscription period (along with a designated *Map Server*), meaning that these EIDs will change very infrequently. This property allows transport communications to survive roaming events, since the EID isn't changed.

When a LISP-MN roams into a new network, there are two possible cases: either the network is a LISP site, or it is a regular site.

LISP-MN in a non-LISP site

In this case, the LISP-MN will act as a complete LISP site, and operations will flow as usual. The LISP-MN receives a new RLOC (assigned through DHCP) which is globally routable and registers it to its *Map Server*. Packets exit the LISP-MN already encapsulated, arrive at the border router, that won't modify them, and simply forwards them natively into the Internet. Packet flow and encapsulation can be seen in Figure 4.1.

LISP-MN in a LISP site

In this case however, things are a little bit more complex, as a LISP-MN behind an xTR is exactly equivalent to a LISP site within another LISP site. The RLOC that is assigned to the LISP-MN comes from the EID prefix of the LISP site, which means that it is not globally routable. These RLOCs are sometimes called *Local* RLOCs (LRLOCs) to make the distinction with classic RLOCs.

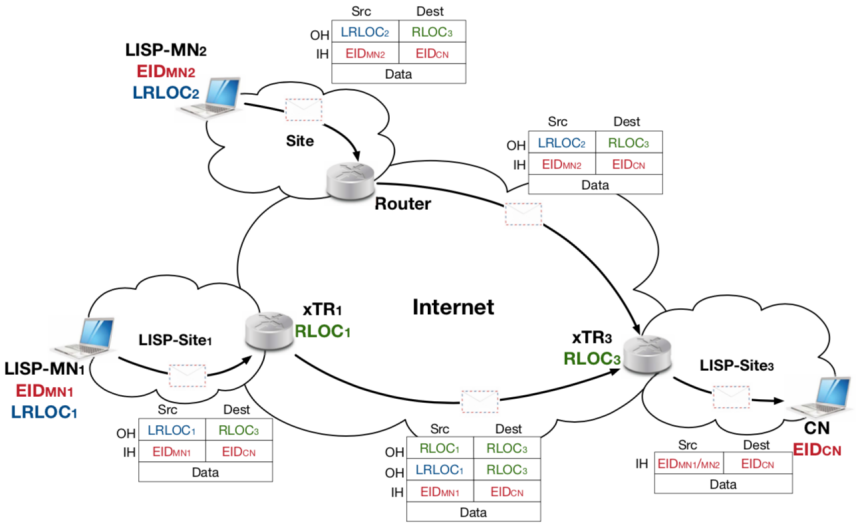


Figure 4.1: LISP-MN in LISP site and in non-LISP site
[16]

Looking at Figure 4.1, we can see that packets leave the LISP-MN already encapsulated, with the outer header source address as LRLOC_1. This address is in fact an EID, which means that returning packets cannot simply use that address as destination in the outer header: another level of encapsulation is needed.
The ITR will therefore encapsulate the LISP-MN packet again before forwarding it to the Internet. Returning packets also need to be encapsulated twice.

1.2 Handover procedure

Let us now review the general handover procedure when a LISP-MN roams between two sites. When a LISP-MN arrives in a new subnet, it receives a new IP address that will serve as its (L)RLOC. The first step is thus to register this new (MN_{EID}, (L)RLOC) mapping to the Mapping System, through its designated *Map Server*. This procedure allows to update the Mapping System with the new location of the LISP-MN, so that new connections can reach the LISP-MN. However, ongoing connections flowing through remote (P)ITRs also need to be updated, otherwise, the returning packets will be encapsulated towards the previous location of the LISP-MN. To do so, the LISP-MN has at its disposal several mechanisms to update the remote caches. The one that is used for a roaming event is the SMR procedure [5]. SMR stands for *Solicit Map Request* and is a way for a LISP device to tell a remote LISP device that it should refresh the mappings it has cached. Upon receiving an SMR for a certain EID, the LISP device will send a MapRequest for that EID, in order to refresh its mappings. Therefore, to allow packets to be encapsulated towards the new location of the LISP-MN, the LISP-MN sends SMRs to all sites it has been receiving encapsulated packets from.

Scalability of the Mapping System

With frequent roaming events that modify the mappings of the mobile EID and its corresponding RLOCs, one can ask oneself if the performance of the Mapping System will remain good.
In fact, *Map Servers* are assigned a range of EID prefixes they are responsible for, meaning that

roaming events (i.e., assignment of new RLOCs) will be confined to the *Map Server*, as well as to the (P)ITRs that have cached the mapping because they have been communicating with the LISP-MN. With such an architecture, the Mapping System scalability remains good, as it doesn't require additional state in the rest of the Mapping System.

On the other hand, LISP-MN has an important shortcoming: it requires (P)xTRs to perform a double mapping lookup for *all* traffic, even if that traffic is not concerned with mobile nodes. The double encapsulation that is sometimes needed when a LISP-MN is in a LISP site is already a drawback, but at least it is only limited to mobile nodes. However, double lookup will concern all traffic. Let us take back the example from Figure 4.1, when a LISP-MN is behind an xTR, to better understand why. When the returning packet arrives at xTR_3, the xTR needs to perform a first lookup for destination EID EID_{MN1}. The returned RLOC, LRLOC_1, is itself an EID that is not routable. The xTR thus needs to perform a second lookup for LRLOC_1 to finally get RLOC_3, and perform a double encapsulation of the packet. The problem is that the xTR has no way of knowing when a returned RLOC is globally routable or not. This means that even if a conventional RLOC is returned, the xTR must still check that with a second lookup.
For the interested reader, Menth et al. [13] present the different shortcomings of LISP-MN and introduce some improvements to remedy these issues, including the problem of double lookup.

2 NAT extensions

LISP, that divides the address space into EIDs and RLOCs, works on the assumption that xTRs will always be reachable through their globally routable RLOCs on port *4341*. However, when a LISP device (either an xTR or a LISP-MN) is situated behind a NAT, this assumption does not hold anymore, as nodes behind a NAT are only accessible through the NAT public address. The draft [8] introduces LISP extensions for NAT traversal. With it, a LISP device is able to detect if it is located behind a NAT, and take the necessary measures to send and receive traffic nevertheless. The NATed device initialises state on the NAT, and then uses a new network element, the *Re-encapsulating Tunnel Router* (RTR) to forward traffic to and from other LISP devices through the NAT.

2.1 NAT Traversal Overview

When a LISP device is situated behind a NAT, its RLOC(s) are typically private addresses that are neither unique nor globally routable. Moreover, a NAT usually requires to first initialize state with outgoing packets before it is possible to receive incoming packets. Additionally, LISP requires xTRs to encapsulate data with destination UDP port *4341*, which is not possible anymore with the NAT translated address and port. Finally, depending on the type of NAT that is used, the mapping state of the NAT can be more or less restrictive. The more restrictive NATs use the full 5 tuples (IP src, IP dst, UDP/TCP src, UDP/TCP dst, protocol), meaning that even when an outgoing mapping is established for the NATed xTR, traffic from various other xTRs may be blocked because it doesn't match the entire tuple.

For all these reasons, a LISP-specific NAT traversal mechanism needs to be introduced. Firstly, a LISP device that just received an RLOC has to discover if it is situated behind a NAT or not. To do so, the device will query its *Map Server* with the help of two new LISP control messages, the InfoRequest and InfoReply, in order to discover its global address. If the device discovers it is behind a NAT, it will use the RTR to proxy its registration process. Additionally, state will be initialised in the NAT during this registration process, so that data packets can flow through the

NAT in the future. Once the registration process is done, inbound and outbound packets for the NATed device will flow through the RTR, whose main function is to serve as a proxy to relay control and data traffic through the NAT.

We will now review in more details the different adaptations that need to be made to LISP in order to allow for NAT traversal. LISP processing in the case of a NATed device is composed of three main steps: NAT discovery, Registration Process, and Data Forwarding.

2.2 NAT discovery

Each time that a LISP device (in particular a LISP-MN) receives a new RLOC, it must check whether it is behind a NAT or not. For this, two new control messages have been created: the InfoRequest and the InfoReply. When a *Map Server* receives an InfoRequest, it answers it with an InfoReply with the global address and port as seen by the *Map Server*, as well as a list of RTRs if the requesting device ever needs relaying. That way, the LISP device can compare the returned global address and port with its address and port and determine whether it is NATed or not.

2.3 Registration Process

The registration process, that is used by xTRs to publish their EID-to-RLOCs mappings, must be adapted, as the RLOC(s) of a NATed device are not globally routable and can thus not be published in the database. The NATed device will therefore use the RTR to proxy its MapRegister messages, and establish state into the NAT at the same time.

The registration process is carried out in four steps, also presented in Figure 4.2:

- The LISP device crafts a MapRegister message that contains the RTR RLOC(s), in order to register those RLOC(s) as the RLOC(s) where the LISP device EID prefix is reachable. As a result, all packets destined to this EID will be encapsulated towards the RTR. This MapRegister message is encapsulated into a LISP ECM header destined to the RTR's RLOC. It is important that the the outer header source port is set to *4341* in this step, as it will initialise state on the NAT so that the NATed LISP device can receive traffic on port *4341*.
- Upon reception of the ECM'ed MapRegister, the RTR strips the ECM header and re-originates the message towards the *Map Server*.
  Additionally, it must also record some information about the EID prefix and LISP device that just registered to it, so that it can later forward LISP data traffic towards the NATed LISP device. A new entry in the cache will be created that contains the following information: The outer header source RLOC and source port, that correspond to the address and port modified by the NAT. The inner header source RLOC will also be recorded as it corresponds to the LISP device local NATed RLOC. Finally, the outer header destination RLOC (i.e., the RTR's own RLOC) will be recorded as well to use the same address in returning packets and be consistent.
- When the *Map Server* receives an ECM'ed MapRegister, it stores the mapping, and issues an ECM'ed MapNotify towards the RTR. When MapRequests will come to the *Map Server* to ask for the EID, they will usually be forwarded towards the RTR (see 2.4).
- Finally, the RTR that receives an ECM'ed MapNotify message destined towards one of its registered EID prefixes, will encapsulate the MapNotify in a LISP data header and sends it to the associated LISP device. This MapNotify inside a LISP data header is referred to as a Data-Map-Notify message.

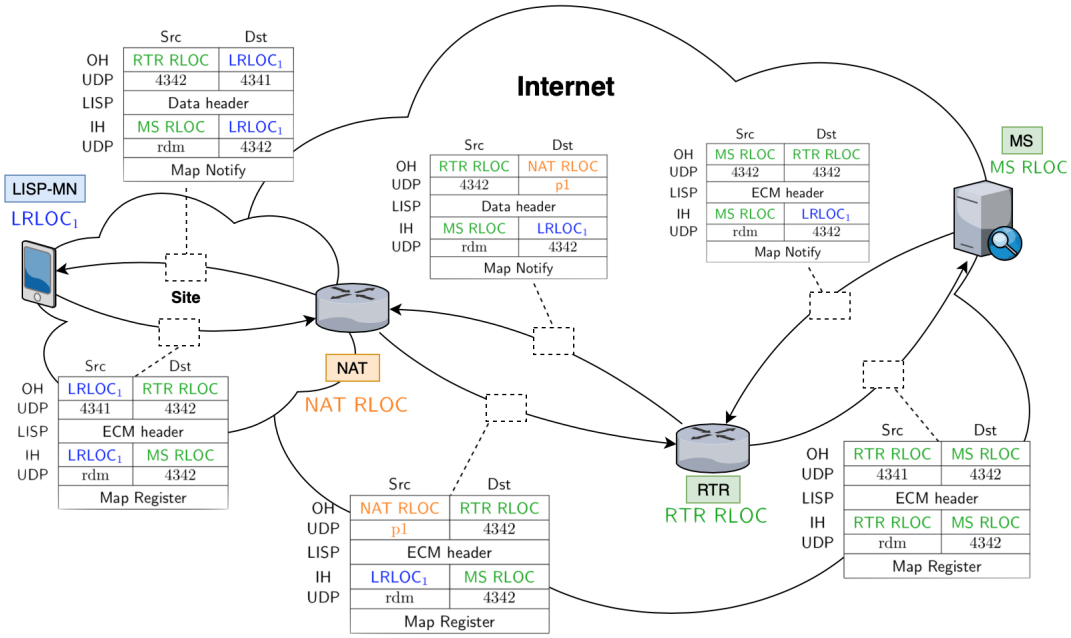


Figure 4.2: NATed Registration Process

2.4 Data Forwarding

Now that the registration process has been carried out, the RTR can serve as a relay for LISP data packets between a NATed LISP device and other LISP devices.

Data Forwarding

When a LISP packet encapsulated towards the RTR's RLOC arrives at the RTR, the RTR first checks whether the source or destination is a previously registered EID.

If the source is a previously registered EID, this means that the packet comes from a NATed LISP device. In that case, the RTR will act as a PETR, i.e., the RTR will strip the outer header and process the packet based only on the inner header. The packet will either be forwarded natively in the Internet, or it will be LISP encapsulated towards an xTR. NATed LISP devices will always encapsulate all outbound traffic towards the RTR, and don't need to issue MapRequests for the purpose of finding EID-to-RLOC mappings anymore. Therefore, it is the responsibility of the RTR to issue MapRequests if necessary.
If the destination is a previously registered EID, this means that the packet is destined to a NATed LISP device. In that case, the RTR will strip the LISP data header and re-encapsulate the packet in a new LISP data header. The outer header destination address and port are filled based on the cache entry created during the registration process (see 2.3) in order for the packet to go through the NAT. The outer header source RLOC is filled with the RTR RLOC from the cache entry as well, and the source port is set to 4342.

The entire process is illustrated in Figure 4.3.

Handling MapRequests/MapReplies

Handling MapRequests for a NATed EID space can be done in several ways.

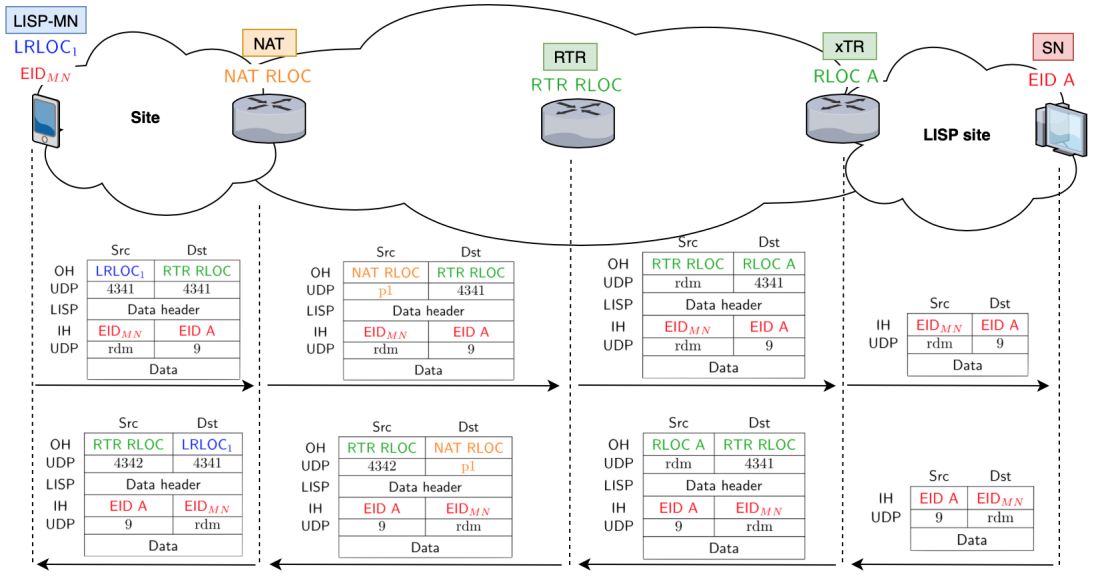


Figure 4.3: NATed Data Forwarding

- If the proxy bit in the ECM'ed MapRegister is set, the MS will answer itself with a MapReply.
- If the proxy bit is not set in the ECM'ed MapRegister, the MS forwards MapRequests to the registered RLOC(s) (i.e., the RTR RLOCs). Note that it is also possible for a NATed LISP device to receive directly MapRequests from the *Map Server* by registering the NATed xTR translated RLOC, but it requires to maintain state in the NAT. Moreover, if an ETR behind a NAT chooses to receive MapRequests from the *Map Server*, it must also send MapReplies to the requesting ITRs. Therefore, this configuration is not recommended as it will result in excessive state in the NAT. The LISP device (either the *Map Server*, the RTR, or the NATed xTR) must include the RTR RLOC(s) as the locator set in the MapReply in order for traffic to be encapsulated towards the RTR.

3 SMR procedure for a NATed LISP device

After investigation, we saw that the different works on NAT Traversal for LISP (LISP+NAT [8], among others) only cover static scenarios, i.e., scenarios in which there is no roaming and no handover. In particular, the SMR procedure to update remote caches after a roaming event is left completely unspecified. Therefore, we propose an extension to the protocol to allow a LISP-MN to update remote caches, even when NAT is implicated.

As we already explained in Section 1.2, upon a roaming event, the LISP-MN must update the remote caches of the LISP devices it has been communicating with. This is crucial for packets of ongoing communications to be encapsulated towards the new (L)RLOC of the LISP-MN. To update the caches upon a roaming event, the SMR procedure [5] is used. As a quick reminder, the LISP-MN will send an SMR to the remote LISP device, in turn this device will send a MapRequest for the LISP-MN EID, causing its cache to be refreshed.
We will review two aspects of the SMR procedure when a LISP device is NATed: the procedure in itself, to send SMRs through the RTR and get remote caches to be updated. And the mechanism

that is used to put state into the NATed LISP device, in order for it to know which (P)ITRs to send SMRs to.

3.1 Message exchange specification

In this section, we will define the message exchange for the complete SMR procedure when a LISP device is NATed. Here, we make the assumption that the LISP-MN already knows exactly all the LISP devices that are encapsulating towards itself, thus all the LISP devices that need to have their caches updated. However, we will see in the next section (Section 3.2) that this is not automatically the case, and we will define a mechanism to remedy this issue.

When a LISP device is NATed, it sends SMRs encapsulated in an ECM header towards its RTR, in exactly the same way that its MapRegisters are encapsulated towards the RTR. More precisely, the inner header source and destination are the LRLOC of LISP-MN and the RLOC of the remote device respectively. Then this packet is encapsulated in an ECM header. The outer header source and destination are the LRLOC of LISP-MN once more, and the RTR RLOC this time.

The RTR, upon receiving an ECM encapsulated SMR, will remove the outer header and the ECM header, and re-originate the message to send it to the remote ITR. The ITR RLOC field in the SMR is set to the RTR RLOC, so that the SMR-invoked MapRequest is sent directly to the RTR.

The remote ITR receives the SMR, and sends back an SMR-invoked MapRequest to the RTR, which directly answers with a MapReply. Notice that the SMR-invoked MapRequest is not forwarded towards the NATed LISP device. Indeed, it would only add overhead to the communication, while the RTR is perfectly able to answer directly, on behalf of the NATed device. The entire process can be found in Figure 4.4.

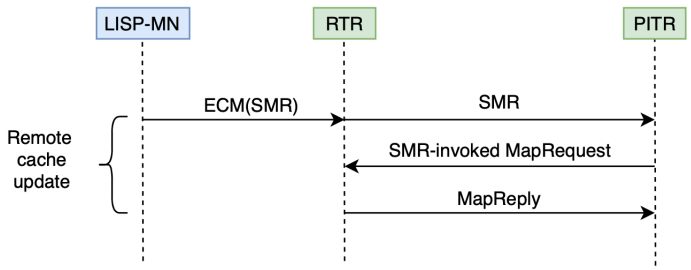


Figure 4.4: NATed SMR Procedure

3.2 NATed LISP device awareness

Earlier, we made the assumption that the LISP-MN already knew all the RLOCs that are encapsulating towards itself. Actually, when situated behind a NAT, this is not true anymore, and the LISP-MN is not necessarily aware of all the remote (P)ITRs. We will first review why the LISP-MN may not be aware of all the remote (P)ITRs, then we will see the mechanism to give the LISP-MN that information, so that the SMR procedure can be started.

NATed LISP-MN's empty LISP Cache

When a LISP-MN is not NATed, traffic flows between both ends uninterrupted. In particular, traffic from the LISP-MN towards the remote (P)ITR is encapsulated based on a mapping stored in the LISP Cache at the LISP-MN. The normal SMR procedure is based on that cache, and specifies that an SMR is to be sent to each RLOC stored in the cache.

When a LISP-MN is NATed however, the situation is different. The problem comes from the connections that are established while the LISP-MN is behind the NAT. While it is behind the NAT, the LISP-MN has no state about the (P)ITRs with which it is communicating. Indeed, the entire load is put on the RTR, who is in charge of forwarding traffic for the LISP-MN. The LISP-MN doesn't send any MapRequests anymore, as all the traffic is automatically encapsulated towards the RTR. Therefore, its LISP Cache will never be populated with remote ITRs. Furthermore, the LISP-MN doesn't receive any MapRequests either, as the RTR is in charge of delivering MapReplies on behalf of the NATed LISP device. As such, the LISP-MN is not aware of the communications that are occurring.

As long as the LISP-MN stays behind the NAT, and communicates through the RTR, all is well. But this is problematic when the LISP-MN roams in another site with no NAT, and must thus send SMRs to remote (P)ITRs: it has no state in its LISP Cache about the different remote (P)ITRs, and thus can not start the SMR procedure.

Solution and message exchange specification

Therefore, the solution that we propose defines a mechanism to make the NATed LISP-MN aware of the various communications it is having with remote LISP devices.
To make the NATed LISP-MN aware of the remote (P)ITRs, the solution is for the RTR to forward the MapRequests for the LISP-MN EID to the LISP-MN. That way, the LISP-MN knows about all the (P)ITRs that ever requested its mapping and can record their RLOCs in a list, that we will call the RemoteItr Cache. Therefore, when the LISP-MN roams into another site, it can send SMRs to each RLOC in this list.

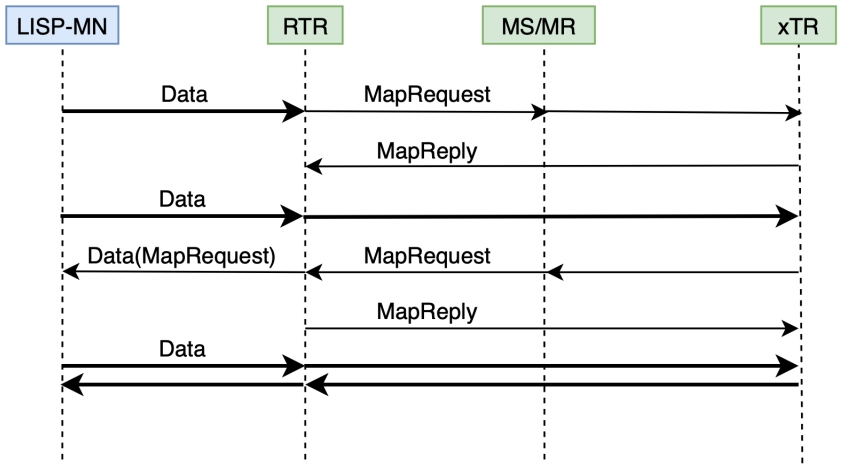


Figure 4.5: NATed LISP-MN: state establishement

This mechanism can be found in Figure 4.5. Upon connection establishment, the LISP-MN doesn't send any MapRequests and automatically encapsulates all data towards the RTR. The

RTR is in charge of resolving the mapping for the requested EID and sends a MapRequest to the Mapping System. After receiving a MapReply, data can flow towards the xTR. For the returning flow, the xTR will also query the Mapping System to resolve the LISP-MN EID. As the RTR RLOC is registered for the LISP-MN EID, the MapRequest arrives at the RTR.
Previously, the RTR would only send a MapReply back to the xTR, and that would be sufficient for the flow to be established in both directions. However, the LISP-MN would not be aware of the remote xTR. Therefore, the RTR will also forward the MapRequest to the LISP-MN, encapsulated in a LISP Data header in order to cross the NAT. Upon reception of the MapRequest, the NATed LISP-MN will record the RLOC in the RemoteItr Cache. As the RTR is already in charge of sending back MapReplies, the LISP-MN doesn't answer the MapRequest. Indeed, this mechanism is not used to resolve mappings, but only to make the NATed LISP-MN aware of its correspondents.

The SMR procedure in itself (i.e., the message exchange) is unchanged, except that it is now based on the RemoteItr Cache, and not on the LISP Cache anymore (as currently defined in LISP RFC). The RemoteItr Cache is populated based on the MapRequests received by the LISP-MN, whether the device is NATed or not.

3.3 RTR processing

All in all, the RTR performs the same operations as a classic xTR, with some specificities for being used as a relay (among others, the novel SMR procedure we just reviewed). The state transition diagram of the RTR can be found in Figure 4.6, where we have a look at the additional functions of the RTR in the Control Plane.

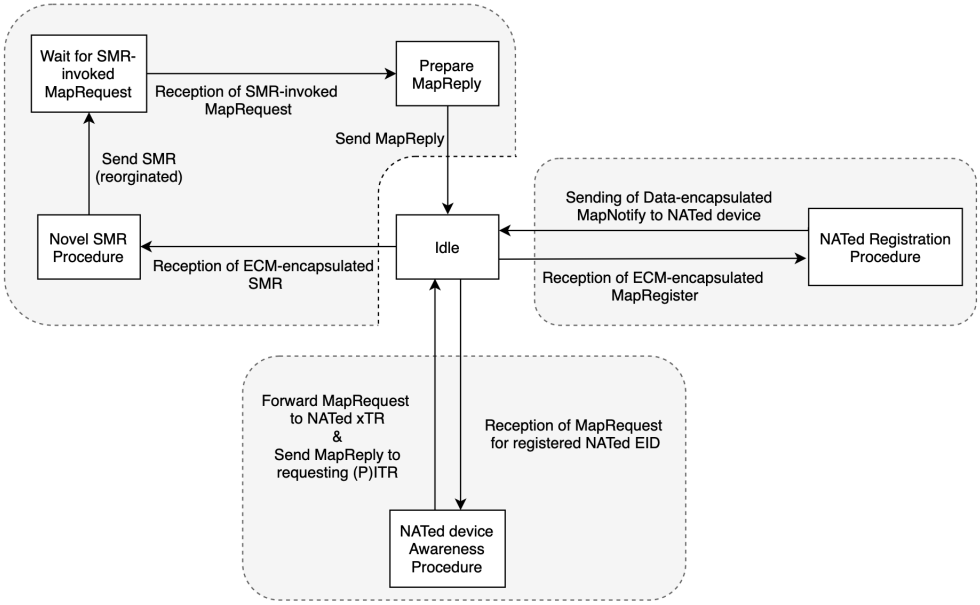


Figure 4.6: State transition diagram of RTR

There are three triggers, each starting a different procedure.
The reception of an ECM encapsulated MapRegister means that a NATed device wishes to use the RTR as a relay. The NATed Registration Procedure is thus started, and the necessary information is recorded at the RTR.

The reception of an ECM encapsulated SMR means that a NATed device wishes to update its correspondents remote caches through the RTR. The novel SMR procedure is henceforth started: the RTR reoriginates the SMR, sends it, waits for the corresponding SMR-invoked MapRequest, and sends a MapReply.
The reception of a MapRequest for a registered NATed EID triggers what we call the NATed device Awareness Procedure, which consists into forwarding the MapRequest, encapsulated in a data header, back to the NATed device, for it to be aware of its correspondents.

Chapter 5

NS-3 Implementation

LISP-MN [7], as well as the NAT extensions [8], are still at the early development stage. LISP-MN has already been the object of several studies ([13], [16]). But to the best of our knowledge, no investigation of the NAT extensions has been conducted. This proposal is only limited to the theoretical level, and lack any implementation and experimentation, as well as results. For this reason, we decided to implement the NAT extensions in the ns-3 Network Simulator [9], in order to evaluate it. Using simulation and ns-3 suits our purpose, as it would have been difficult to perform studies with real systems. Moreover, simulation has become more and more important to evaluate new technologies, or as a proof of concept of new protocols.

This chapter presents briefly the ns-3 Network Simulator, as well as the LISP implementation in it. Then we describe our extension of that implementation, namely, NAT, PxTRs, and LISP+NAT. Our implementation is freely available at https://github.com/Emeline-1/ns-3_LISP_NAT

1 The ns-3 Network Simulator

The ns-3 Network Simulator is a discrete-event network simulator targeted primarily for research and educational use. The simulator provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments [9]. ns-3 is open-source and allows researchers to develop their own model, extend the ns-3 libraries, and bring their contribution to the project, as we did with LISP and its NAT extensions. The project is very well documented, and has an active community, which encourages people to contribute to the project by adapting existing code or by writing new models.

ns-3 developers wanted to put the focus on realism, and therefore designed an architecture that is close to a real Linux machine, with channels, net devices, a TCP/IP protocol stack, socket programming, and applications. The project is entirely written in C++ (although some bindings to python exist) for ease of use, maintainability, and performance. The simulation scripts written by the user to conduct experiments are also written in C++, unlike other simulators that use a pseudo-language for that purpose.

2 LISP implementation in ns-3

The first LISP model in ns-3 has been written by L. Agbodjan [22] under ns-3.24, and implements basic LISP functionalities: the Data Plane and the Control Plane, and the *Map Server* and *Map Resolver*, as well as the LispHelper. Then this work has been further refined by Y. Li [16], who

adapted it to ns-3.27, and implemented the LISP-MN extensions. Additionally, she covered some shortcomings of the original code. The implementation respects LISP RFC 6830 [5] and LISP mobility standards [7].

We will now review the basis of the architecture of the LISP implementation in ns-3. LISP has been implemented in the internet module of ns-3, as it is heavily dependent on IP. The model is two-fold: the Data Plane (i.e., packet forwarding), and the Control Plane (i.e., mappings resolution). A communication mechanism between the two is also necessary for the Data Plane and the Control Plane to interact with each other.

2.1 LISP Mapping Socket

Because the LISP implementation is divided into two distinct parts, the Data Pane, and the Control Plane, communication between both is required. This is necessary for example when a cache miss occurs during the encapsulation of a packet in the Data Plane. In that case, the Control Plane must be triggered to send a MapRequest and retrieve the requested mappings. Conversely, when a MapReply arrives at the xTR in Control Plane, the Data Plane must be notified to insert the new mapping information into the LISP Cache. Other types of events can also happen and need to be transmitted between the two Planes.

For this communication to be possible, the example of OpenLisp [23] has been followed. To allow the Data Plane and the Control Plane to interact, OpenLisp provides a Mapping Socket API. Formatted messages can thus be exchanged between the Data Plane and the Control Plane to notify each other that a particular event occurred. The implementation of a Mapping Socket is convenient because it allows the two Planes to be developed completely independently of each other.

2.2 Data Plane

The role of the Data Plane of LISP is to encapsulate/decapsulate packets.

LISP Database and LISP Cache

A LISP device maintains a LISP Cache and a LISP Database where EID-to-RLOCs mappings are stored. The Cache is populated on demand, when the first packet of a new flow arrives and that no mapping for the destination EID is found in the Cache. The LISP Database is populated by configuration and holds the mappings for which the LISP device is authoritative. The Cache is used by ITRs to encapsulate traffic, and by ETRs to perform basic anti-spoof checks. The Database is used by ITRs to choose the source RLOC of the encapsulated packet, and by ETRs to answer MapRequests.

In ns-3, both the Cache and the Database are implemented by the class MapTables, which provides an interface that must be implemented in subclasses. The various operations of MapTables (create, update, search, delete mappings) are implemented in SimpleMapTables, with a simple list of mappings that can be iterated.

LISP operations

LISP operations consist in encapsulating and decapsulating packets. To do so, a first class, LispOverIp, and its extended classes, have been implemented to perform LISP specific tasks. This

class is in charge of determining if a packet must be processed by LISP (NeedEncapsulation(), NeedDecapsulation()), and if so, to encapsulate/decapsulate the packet, with the methods LispOutput() and LispInput(). This class also holds a pointer to MapTables to have access to the different mappings. The architecture of the Data Plane can be found in Figure 5.1. Colored classes correspond to already existing classes in ns-3. Blank classes are new classes added for the LISP implementation.

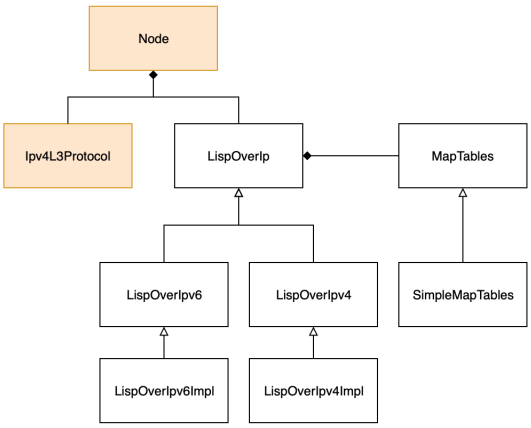


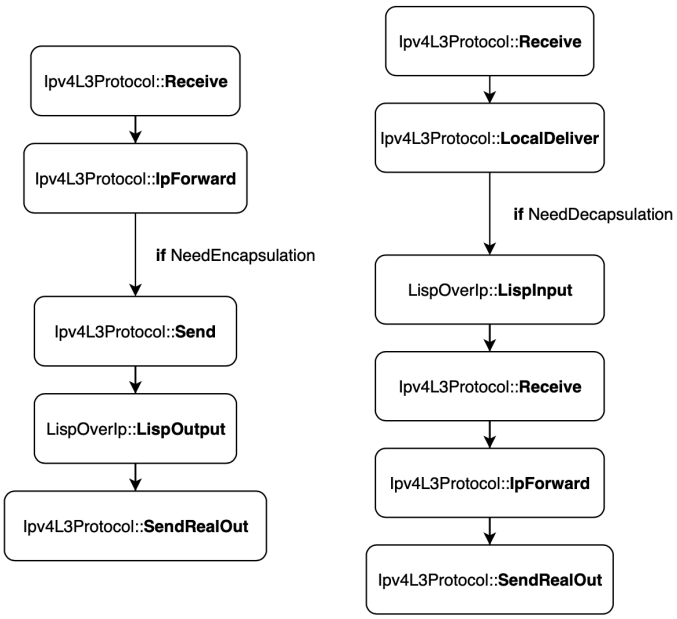
Figure 5.1: Data Plane UML diagram

Next, the most difficult part was to integrate LISP functionalities into a classic IP stack. As we already explained in Section 1, the IP protocol stack of an ns-3 node is very close to the protocol stack of a real Linux machine. As such, the following functions to process a packet at the IP level are in use:

- Receive: This method is called by lower layers (typically a net device) upon reception of a packet.
- IpForward: As the name indicates, this function is called after Receive when the packet is not destined to the current node, but must be forwarded through an outgoing device.
- Send: This method is called by higher layers (typically a socket of an application) to send a packet down the IP stack.
- LocalDeliver: When the packet is destined to the local node, it will be delivered to one of the applications, or an ICMP unreachable message will be sent back.

These functions are implemented in the class Ipv4L3Protocol (see Figure 5.1) and each of those methods has been adapted to add LISP processing to it. To better understand the encapsulation and decapsulation workflow, let us review a classic example of a packet exiting a subnet through an ITR; and of a that same packet arriving at the destination ETR (see Figure 5.2).

In Figure 5.2a, we review the encapsulation workflow at an ITR. When the packet arrives at the ITR, the Receive method of Ipv4L3Protocol is called by the net device. As the packet is not destined to the ITR (but to a remote EID), control is passed to the IpForward method. This method has been slightly modified to perform basic checks and determine if the packet needs LISP encapsulation or not. If it does, control is passed to the Send method of Ipv4L3Protocol. The Send method has been adapted to retrieve the different mappings needed for the encapsulation, as well as to perform additional checks. Once the mappings are retrieved, they are passed to



(a) Encapsulation workflow (b) Decapsulation workflow

Figure 5.2: Data Plane workflow

LispOutput, whose role is to perform the actual encapsulation of the packet, i.e., adding the LISP header, followed by the UDP header, followed by the outer IP header. Finally, when the packet is fully formed, it is passed to SendRealOut that will forward it down the stack to the second layer. This is during the encapsulation of a packet that a cache miss can occur if the mapping for the destination EID is not found in the cache. This is the place where the LISP Control Plane will be triggered through the LISP Mapping Socket (see Section 2.1) to retrieve the requested mapping.

In Figure 5.2b, we review the decapsulation workflow when a LISP packet arrives at the ETR. Once again, the Receive method of Ipv4L3Protocol is called by the lower layers. This time however, the packet is destined to the ETR itself, as it is encapsulated with the ETR RLOC address. As such, control is passed to the LocalDeliver method of Ipv4L3Protocol. This method has been partially rewritten to check if the packet is a LISP encapsulated packet and should be decapsulated or not. If so, control is passed to LispInput, whose job is to decapsulate LISP packets. Note that control packets addressed to the ETR are delivered to the xTR application, of which we will talk later. LispInput performs consistency checks (regarding mappings for example) on the packet and control if it is well formed or not. Once it is decapsulated, the inner packet is passed to the Receive method again. This time however, it is destined to the remote EID, meaning that control will be passed to IpForward. As the packet doesn't need encapsulation, it will directly be passed to SendRealOut and exit the ETR.

2.3 Control Plane

The main role of the Control Plane is to retrieve mappings from the Mapping System upon the event of a cache miss. More generally, the Control Plane implements all functions related to setting the Mapping System up and the operations that allow to retrieve mapping information from this MDS. This includes xTRs and MR/MS operations, as well as the different control messages exchanged between them. The diagram of the Control Plane can be found in Figure 5.3.

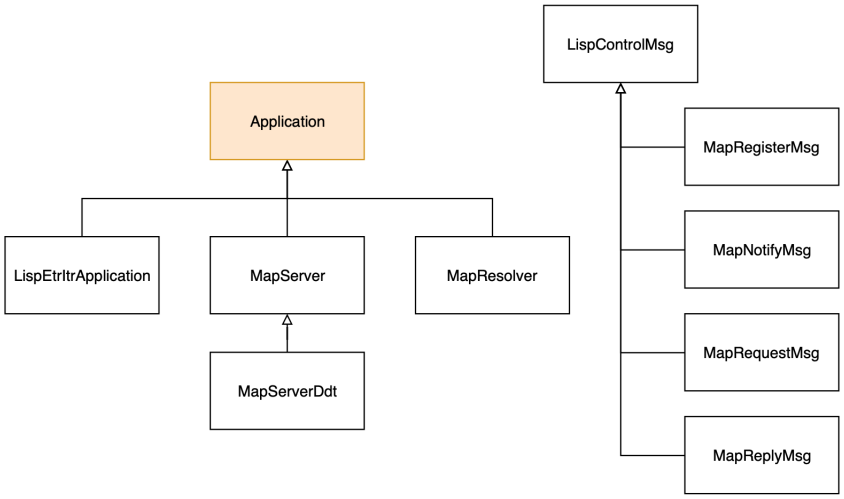


Figure 5.3: Control Plane UML diagram

xTR operations

The functionalities of an xTR have been implemented in the class LispEtrItrApplication, which extends the ns-3 class Application. A LispEtrItrApplication is meant to be installed on a node with a LispOverIp Data Plane installed. An ns-3 node with a LispOverIp Data Plane and a LispEtrItrApplication is a fully functional xTR.

The xTR operations are quite straightforward: upon starting the application, the xTR sends MapRegister messages to its assigned *Map Server* based on the content of its LISP Database. The xTR is also able to handle the different LispControlMsg and respond accordingly.
Upon reception of a MapRequest, the LispEtrItrApplication executes as LISP database lookup to answer with a MapReply.
Upon reception of a MapReply, the LispEtrItrApplication notifies the Data Plane through the LISP Mapping Socket of the new mapping information, so that it can be inserted into the Cache.
Upon a roaming event, the Control Plane is notified by the Data Plane through the LISP Mapping Socket that it must register its newly assigned (L)RLOC, and send SMR messages to its correspondents.

MR/MS operations

The entire LISP Mapping System is implemented with two classes: MapServer and MapResolver. In particular, MapServerDdt extends class MapServer to implement the workings of the *LISP Delegated Database Tree* (LISP-DDT) [15].

The role of the MapResolver is simply to receive MapRequests from the different xTRs and to forward them to the MapServer.
The Map Server operations are also quite simple. This application always listens on port 4342 (LISP control port) for control messages. Upon reception of a MapRegister, the MapServer updates its database to store the corresponding mapping information, and answers with a MapNotify if necessary. This information will allow the MapServer to forward MapRequests to the correct authoritative ETR. Upon reception of a MapRequest (forwarded by the MapResolver), the MapServer will lookup its database to see if any xTR has registered for the required prefix, and if so, forwards the MapRequest to the registered xTR.

Note that this implementation doesn't form a complete LISP Mapping System with a DNS-like architecture. Indeed, it is only composed of two nodes that simulate the entire Mapping System. We will see in Chapter 6 how to remedy this issue.

2.4 LISP Helpers

In ns-3, when developing a model, it is good practice to also develop a Helper for it. A Helper can be seen as a "wrapper" class to make the implemented features more user-friendly. Its goal is to deal with the low-level details, so that the script writer doesn't need to know the implementation details of the model, and can very quickly and easily configure an entire network.

There are many Helpers in ns-3. For example, there is the InternetStackHelper, that allows one to install the entire IP stack (TCP, UDP, IP, etc) on a node without knowing anything about the implementation of this stack. Another example is the PointToPointHelper, which does the low-level work required to put a link between two nodes together. There is also the Ipv4AddressHelper, that manages the allocation of IP addresses on nodes; and many others that are here to make the life of the script writer easier.

Various Helpers have also been written for the LISP model, to install the LISP Data Plane on a node, to install the LISP Control Plane on a node, and to install the MR/MS applications.

3 LISP extensions in ns-3

Our contribution to the LISP implementation in ns-3 consisted into adding a NAT module, PxTRs functionalities, and NAT extensions to the existing LISP model. Our implementation respects the NAT extensions (LISP+NAT) [8], with some extensions to the protocol for a particular case that had not been covered by the draft, as explained in Chapter 4 Section 3. Additionally, we also wrote a LISP-MN Helper to help the script writer deal with the set up of a mobile node. Moreover, various unit tests for different scenarios (both NAT- and LISP-related) have been added into the ns-3 testing framework.

3.1 NAT model

As our work focuses on NAT traversal for LISP traffic, we needed some NAT model in ns-3. Unfortunately, ns-3 is quite bare-bones regarding the different models implemented in the internet module. As a matter of fact, no implementation of NAT was available in ns-3. Therefore, we had to write our own to provide basic NAT functionalities in our simulations.

We found an implementation of the NAT model for ns-3 written by V. Sindhuja [24] as part of the Google Summer of Code. We therefore took his implementation under ns-3.14 and adapted to ns-3.27 in order to integrate it with our implementation. The goal of this project was to introduce a NAT model into the ns-3 framework, but they also worked on building the basis for a larger framework that supports connection tracking and other firewall features. The work is thus divided into two main parts: the Netfilter framework, and the NAT model.

Netfilter Framework

This implementation is modeled on the Netfilter framework in Linux. Netfilter consists in a set of *hooks* placed in the IP stack to intercept packets flowing through a node. A *hook* is a place in

the networking stack where a packet will be handed over to the Netfilter framework. Different modules can thus register *callbacks* in the networking stack at these *hooks*, allowing for various functions and operations for packet filtering, network address translation, port translation, firewalling, etc.

There are five places in the IP stack where *hooks* have been placed:

- In Ipv4L3Protocol::**Receive**, with the NF_INET_PRE_ROUTING hook.
- In Ipv4L3Protocol::**LocalDeliver**, with the NF_INET_LOCAL_IN hook.
- In Ipv4L3Protocol::**Send**, with the NF_INET_LOCAL_OUT hook.
- In Ipv4L3Protocol::**IpForward**, with the NF_INET_FORWARD hook.
- In Ipv4L3Protocol::**SendRealOut**, with the NF_INET_POST_ROUTING hook.

A model such as NAT can thus register various callbacks at the appropriate *hooks* in order to perform its address translation.

Integrating the NetFilter framework into our version of ns-3 required the addition of 23 files, as well as the modification of the InternetStackHelper (to aggregate the Netfilter object with the networking stack), and of the Ipv4L3Protocol class, where the *hooks* have been placed, as can be seen in Figure 5.4. Orange classes correspond to classes already present in ns-3, blue classes correspond to additions made to integrate the Netfilter framework into ns-3. This Figure only shows the *hooks* and callbacks aspect, not the connection tracking aspect.

The first step was to effectively place the different *hooks* at the correct places into the IP stack functions (Receive, Send, LocalDeliver, IpForward, and SendRealOut) in the Ipv4L3Protocol class. In terms of code, placing a hook consists into calling ProcessHook on the Ipv4Netfilter object with the right *hook* number. Then, we realised that the implementation of the Ipv4L3Protocol class changed a lot in its structure from version 3.14 to 3.27. More precisely, the processing of a packet was still the same, but regarding the code itself, the arguments to the functions had been changed. Therefore, we rewrote the adaptation for Netfilter in the networking stack, mostly playing with adding and removing headers from packets, in order to respect the new structure of the code.
We can have a quick look at the implementation of the Netfilter framework (see Figure 5.4). The class Ipv4NetFilter contains a NetfilterCallbackChain object for each *hook*. The NetfilterCallbackChain class contains a list of the different Callbacks (Ipv4NetfilterHook) that have been registered for the *hook* by other modules. By calling ProcessHook in the networking stack, IterateAndCallHooks is called on the corresponding NetFilterCallBackChain, resulting in each Ipv4NetfilterHook being called.
This implementation was not fully functional, in particular, the decision made about a packet by hooks (drop or accept) was not transmitted back to the IP stack, who thus accepted all packets by default. This issue has been identified and fixed.

Network Address Translation

The NAT model proposed by V. Sindhuja implements both static NAT and dynamic NAT features. The methods DoNatPreRouting() and DoNatPostRouting() are registered at the *hooks* NF_INET_PRE_ROUTING and NF_INET_POST_ROUTING respectively, allowing for packets to be processed by the NAT model. This is done by calling the RegisterHook method on the

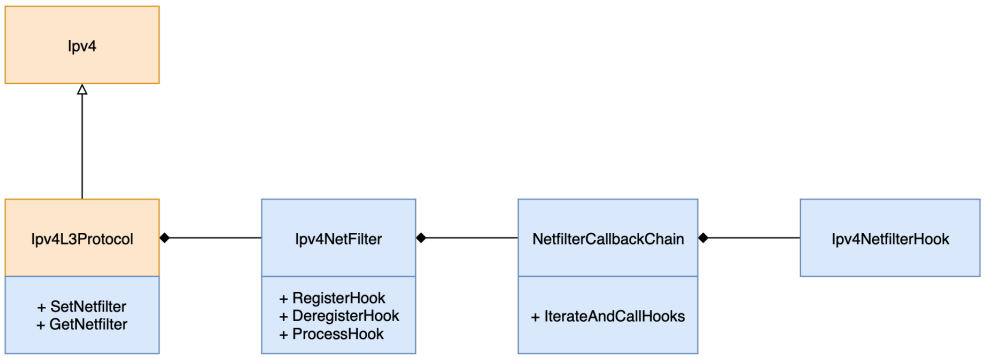


Figure 5.4: Netfilter integration into ns-3

Ipv4NetFilter object.

Integrating the NAT model into our version of ns-3 required the addition of 4 files. Moreover, the implementation suffered some errors, in particular, with regards to the dynamic NAT implementation.
First, the matching-rule logic was wrong: the implementation of the dynamic rule itself was erroneous. Indeed, the local and global (i.e., translated) ports were not taken into account, therefore preventing the possibility to set the correct local port upon arrival of a packet into the NATed subnet. On the other hand, even when a packet was matched to a rule, the address and port were not set correctly, resulting in inconsistencies.
Additionally, the dropping of a packet trying to cross the NAT with no match to a rule was not implemented: every packet was accepted by default, even those without a match to a rule.

All these issues have been identified and fixed, providing a consistent NAT model into the ns-3 simulator as a result. To guarantee the correctness of the NAT implementation, we wrote some unit tests for the NAT model, that we review in Section 3.5. The UML diagram of the NAT module can be found in Figure 5.5.

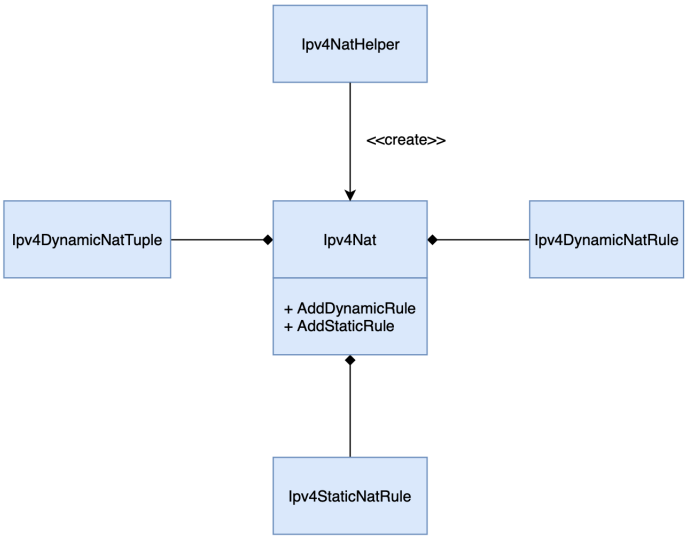


Figure 5.5: NAT module

3.2 PxTRs

Proxy xTRs perform essentially the same operations as classic xTRs, except that the PITR will encapsulate all[1] traffic coming to it (while the ITR only encapsulate traffic for the EID space it is responsible for), and that the PETR will decapsulate all traffic coming to it (while the ETR will only accept traffic for the EID space it is responsible for).

Therefore, our implementation doesn't create any dedicated class for PxTRs, such as LispOverIp-Proxy, but rather works off of the existing LispOverIp. A LISP device is made aware that it is either a PETR or a PITR with a boolean variable. The LispHelper has been rewritten to allow a script writer to specify which devices are supposed to be proxies, and set the corresponding variable in the devices. The main of the implementation consists into bypassing checks made on the LISP Database in the case where the device is a proxy.

More precisely, the Ipv4L3Protocol class has been slightly modified to bypass the check of a non-empty LISP Database in the case of a PITR. Indeed, a PITR encapsulates traffic from all destinations, and is not responsible for any EID space in particular, its LISP Database is thus empty. However, for a classic ITR, a check of a non-empty LISP Database is always performed, because an empty Database means that the device is not responsible for any address space, and should therefore not encapsulate anything.

Additionally, the LispOverIpv4Impl class has also been slightly adapted to bypass the check performed at the ETR to only decapsulate traffic destined for an EID registered into the Database. The class SimpleMapTables has also been modified for the same purpose.

Last but not least, the negative MapReply has been implemented on the *Map Server*. Indeed, up until now, the *Map Server* did not answer with a negative MapReply in the case of a non-LISP site. As a result, because the xTR never received any kind of reply for its requests, all packets for the non-LISP site were dropped. Moreover, at the xTR itself, the case of a negative mapping in the cache was considered as having no mapping at all, causing all packets to be dropped.
Now, with the reception of a negative MapReply, the LISP device can store in its cache a negative mapping. When having to encapsulate packets towards the non-LISP site, and upon seeing that the mapping is negative, packets are not dropped anymore. Rather, the process of encapsulating the packet towards the configured PETR can begin. Indeed, each LISP device is configured with the address of a PETR (specified by the script writer) to which it must encapsulate all traffic for non-LISP sites.

All in all, it is quite easy for a script writer to set up a scenario with PxTRs, as can be seen in Listing 5.1. After creating the topology and the different nodes of the simulation, all that is needed to set up the proxies is to specify to the LispHelper which devices should be made PETR or PITR (lines 10 and 12 respectively). Additionally, the address of the PETR must be configured in the same way with the LispHelper (line 7). As a side note, it is interesting to know for the script writer that the static routes for routing in the nodes must be modified in order to redirect all traffic for a LISP EID towards the PITR, in order for legacy traffic to be encapsulated towards LISP sites.

```
1 [...]
2
3 /* ---- LISP Data Plane ---- */
4 LispHelper lispHelper;
5
```

[1] Rather, all traffic destined to the EID space it has been advertising, on behalf of ITRs

```
// Configure PETR address for all xTRs
lispHelper.SetPetrAddress(iR1_iPETR.GetAddress (1));

// Set PETRs
lispHelper.SetPetrs(NodeContainer(nodes.Get (9)));
// Set PITRs
lispHelper.SetPitrs(NodeContainer(nodes.Get (10)));
// Set RTRs
lispHelper.SetRtrs(NodeContainer(nodes.Get (8)));

// Classic LISP setup
lispHelper.BuildRlocsSet ("./lisp_rlocs.txt");
lispHelper.Install(lispRouters);
lispHelper.BuildMapTables2("./lisp_rlocs_config_xml.txt");
lispHelper.InstallMapTables(lispRouters);

[...]
```

Listing 5.1: Simulation script with PxTRs and RTR

3.3 NAT extensions

Our implementation of the NAT extensions respects the draft LISP+NAT [8], that has been thoroughly explained in Chapter 4 Section 2, with the addition of the novel SMR procedure.

As a reminder, NAT traversal is carried out in three steps: NAT discovery, NATed Registration Process, and Data Forwarding, all of which have been implemented in ns-3. Once again, the different operations of an RTR stay roughly the same than for a classic xTR. Therefore, our implementation focuses on adapting the existing classes and methods to those cases, instead of creating dedicated classes for the RTR.

Control Plane adaptation

The first step was to implement the two new control messages: the InfoRequest and the InfoReply, that allow a LISP device to discover if it is NATed or not. Two new classes, InfoRequestMsg, and InfoReplyMsg have been implemented in the Control Plane, as can be seen in Figure 5.6. The orange classes are classes that were already a part of ns-3, the white ones are classes added for the LISP implementation of L. Agbodjan and Y. Li, and the blue ones are the classes added in our implementation to support LISP+NAT. The workflow of the xTR application has also been modified: instead of sending its MapRegister messages upon starting the application, or upon a roaming event, it first sends an InfoRequest to its *Map Server*. Upon reception of the InfoReply, the LISP Data Plane is notified through the LISP Mapping Socket. Additionally, if the device is NATed, a new mapping is inserted into the LISP Cache for all traffic to be encapsulated towards the RTR. As a result, the LISP-MN will never issue a MapRequest again. For the purpose of notifying the Data Plane, a new type of message has also been added to the LISP Mapping Socket implementation. Lastly, to respect the novel SMR procedure defined in Chapter 4 Section 3, a NATed LISP device doesn't answer anymore to MapRequests. Moreover, the SMR sending is not based on the LISP Cache anymore, but on the RemoteItr Cache.

Then, the second step was to implement the LISP ECM header, which is used for the NATed registration procedure. A new class, LispEncapsulatedControlMsgHeader has been implemented, as can be seen in Figure 5.6. This class extends the ns-3 class Header, and inherits its methods, which allows to add and remove the header to/from a packet very easily.

The methods for creating the MapRegister messages have also been adapted to cover the case where the LISP device is NATed, and where the RLOC advertised in the message should be the RTR RLOC, and not the LRLOC of the device.

The rest of the Control Plane of the xTR hasn't been modified. The main of the implementation concerns the NATed Registration Procedure, as well as the novel SMR procedure, which are implemented in the Data Plane.

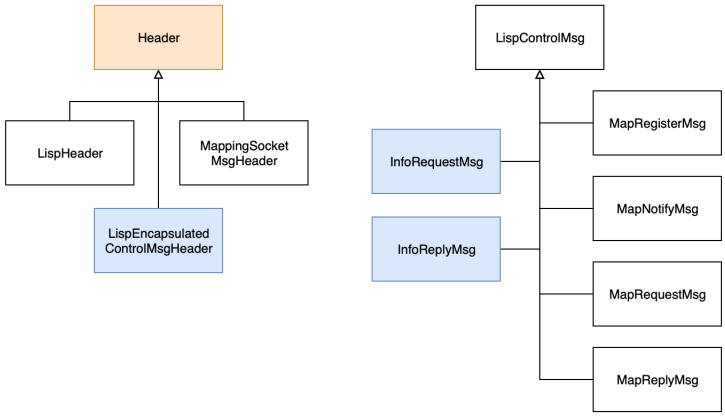


Figure 5.6: Addition to the LISP Control Plane

Data Plane Adaptation

As we just said, the Control Plane workflow of an RTR or of a LISP NATed device is not much different than that of a classic xTR. Indeed, the LISP control messages exchanged between them stay the same. The main of the implementation concerns the various encapsulations of packets, which is the concern of the Data Plane.

First of all, in the same way that a LISP device is made aware that it is a PETR or a PITR, a LISP device is made aware that it is an RTR or a NATed device with a boolean variable. The LispHelper has been rewritten to allow a script writer to specify which devices are supposed to be RTRs, and set the corresponding variable in the devices. The NATed variable is set dynamically after the exchange of InfoRequest/Reply.

We will now review in detail the most important modifications of the workflow of the LISP Data Plane, both in the NATed case, and in the RTR case. More precisely, we will survey four cases:

- The ECM encapsulation of MapRegisters and SMRs at the NATed LISP device.
- The ECM MapRegister processing and forwarding at the RTR.
- The ECM SMR processing and forwarding at the RTR.
- The MapRequest processing for a NATed EID at the RTR.

The white functions correspond to the classic workflow of a packet, and the blue functions correspond to the modifications that have been made in order to encapsulate correctly the packet.

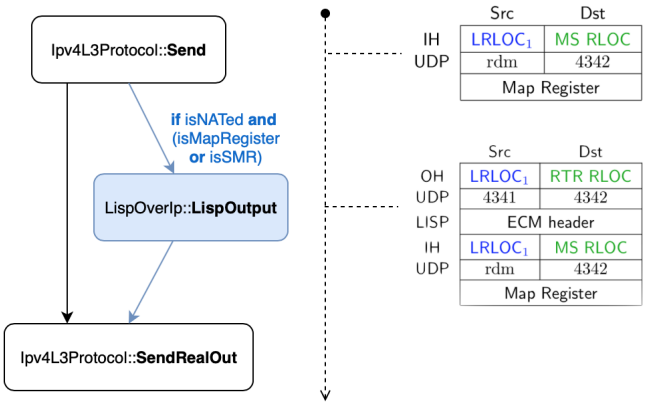


Figure 5.7: ECM encapsulation of MapRegister and SMR at the xTR

We advise the reader to have a look back at Figures 4.2 and 4.3 to get a quick reminder of the packets exchanged during the NATed Registration Procedure, as well as during the Data Forwarding. Figures 4.4 and 4.5 are also useful to remember the novel SMR procedure.

In Figure 5.7, we review the ECM encapsulation of the MapRegister/SMR at the NATed device. When a MapRegister/SMR is sent by the LispEtrItrApplication, the Send method is called. In the normal course, the MapRegister/SMR doesn't need encapsulation if the source and destination are both RLOCs. In that case, the packet is directly passed to SendRealOut and pushed down the stack. In the case of a NATed device however, the MapRegister/SMR needs to be encapsulated into an ECM header towards the RTR. For that purpose, the Send method has been modified: we use DPI techniques to check that the message is a MapRegister/SMR and if so, hand control over to the LispOutput method, in charge of encapsulating packets.
The LispOutput method has also been rewritten to deal with ECM encapsulation. Before it was only able to encapsulate packets in a LISP data header.

In Figure 5.8, we review the processing and the forwarding of the ECM MapRegister sent by the NATed device at the RTR. Two modifications have been made in the workflow: the first one is to intercept the packet and record information about it. The second one is to intercept it again, and encapsulate it in en ECM header towards the *Map Server*.
As usual when receiving a packet, the first method being called is the Receive method. Then, as the MapRegister is encapsulated towards the RTR, control is passed to the LocalDeliver function. This function has now been modified to be able to handle ECM encapsulated packets (previously, it was only able to handle data encapsulated packets). Therefore, if the packet is an ECM encapsulated packet, if the device is an RTR, and if the message is a MapRegister, we know that a remote NATed device wants to use the RTR as a proxy. As explained in Chapter 4 Section 2.3, the RTR must record a certain number of information about the NATed device in order to later be able to reach it. These operations are conducted by the SetNatedEntry method, that will add a special kind of entry into the LISP Cache, recording all that information.
Once this step is carried out, the flow is resumed as usual and LispInput is called. This method is in charge of decapsulating packets, and has been modified to handle ECM encapsulated packets, while previously it was only able to deal with data encapsulated packets.
Once the packet is decapsulated, it will continue its journey in the IP stack. It is passed to the Receive method again. As can be seen in the Figure, this packet comes from the NATed device and is addressed to the *Map Server*, therefore, control will be passed over to IpForward. Then, as the packet needs to be encapsulated, it is passed to the Send method.

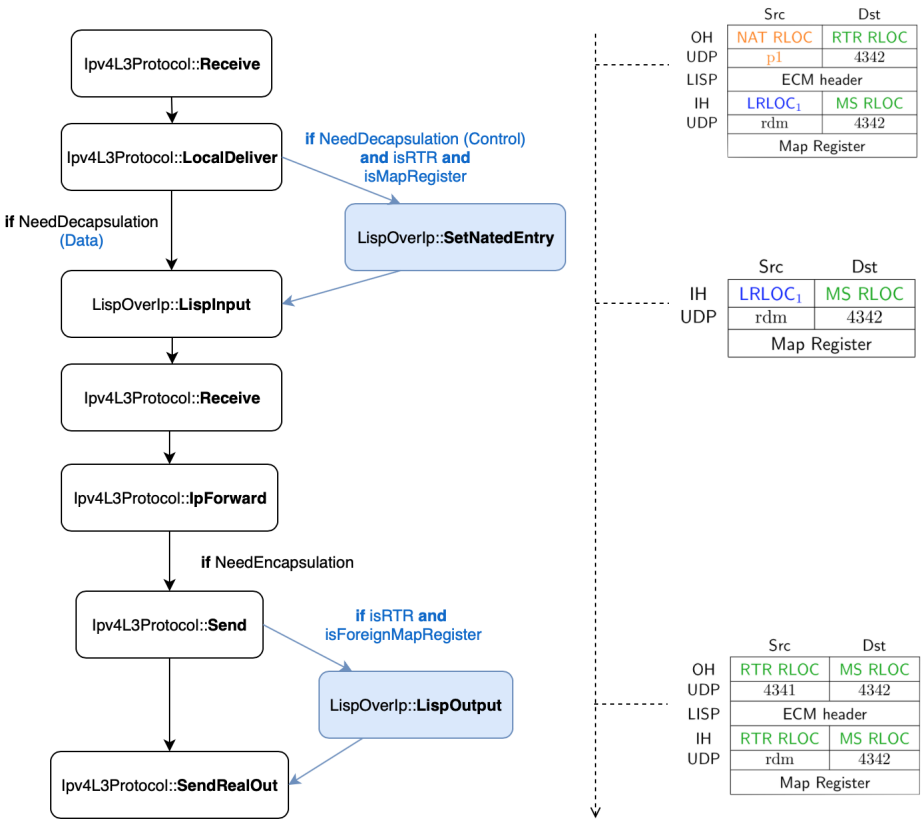


Figure 5.8: ECM MapRegister processing at the RTR

Once again, the packet will be intercepted in the stack with DPI techniques to determine if the message is a MapRegister coming from a foreign device. This can be determined simply by looking at the source address of the packet, which is LRLOC_1, and not the RTR RLOC. If that is the case, it must be ECM encapsulated towards the *Map Server* and the packet is thus passed to LispOutput, that will encapsulate it correctly. Finally, SendRealOut is called to forward the packet down the stack as usual.

Now that we have seen the main of the processing for the NATed Registration Procedure, let us have a look at the novel SMR procedure. The RTR must handle correctly ECM encapsulated SMRs, i.e., change the ITR RLOC field inside the message, and reoriginate it towards the remote ITR, as described in Chapter 4 Section 3. To do so, the procedure is roughly the same as in Figure 5.8. Up until LocalDeliver, the process is carried out in the same way. A new condition is inserted there to check that the message is an SMR in order to change the ITR RLOC field. Once this is done, the workflow is resumed by calling LispInput as usual. Then the packet follows the same path except that it won't be redirected towards LispOutput as it musn't be encapsulated, but rather sent as is to the remote ITR.

Lastly, let us review the MapRequest processing for a NATed EID at the RTR, that can be found in Figure 5.9. As explained in Chapter 4 Section 3, the RTR must do two things: answer with a MapReply on behalf of the NATed LISP device, and transmit the MapRequest, encapsulated in a Data header, back to the NATed device, for it to be aware of its correspondents. To do so, the workflow presented above has been adapted a new time, as can be seen in Figure 5.9. The MapRequest that arrives at the RTR is destined to the RTR RLOC, control is thus passed to the method LocalDeliver. Here, we check if the device is an RTR, and if the MapRequest is

for an EID that is NATed. If so, the IP destination is modified and put to the NATed LISP device RLOC, LRLOC_1. Then control is directly passed to the method Receive. The packet will follow its journey into the stack as usual, and arrive at the Send method. This method has been modified to check if the message is a MapRequest for a NATed EID, and if so, redirect the flow towards LispOutput, that is in charge of encapsulating the packet in a data header.
At the same time, a copy of the packet is made for it to be delivered at the RTR application, and for a MapReply to be sent as usual.

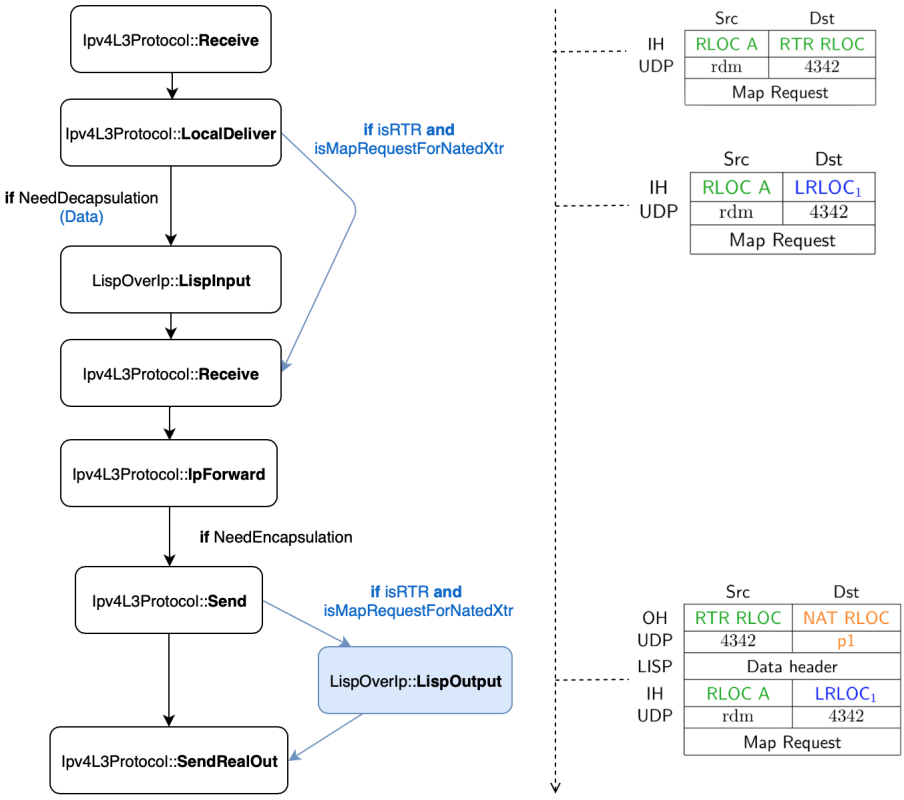


Figure 5.9: MapRequest processing for NATed EID at the RTR

Now that we have reviewed the encapsulation/decapsulation process for the control messages exchanged (during the NATed Registration Procedure, and the novel SMR procedure), let us review the process for the Data Forwarding. The workflow is actually the same as the decapsulation workflow (presented in Figure 5.2b) followed by the encapsulation workflow (presented in Figure 5.2a) that were already in use for classic Data Forwarding.

However, the LispOutput method has been modified again to deal with the NATed Data Forwarding. Indeed, there are two cases of figure when the device is an RTR: either it receives packets from a NATed device, or it receives packets towards a NATed device. In the first case, the encapsulation procedure is not modified. The RTR will perform the necessary MapRequests if no mapping is present in its Cache and encapsulate packets towards remote ETRs in the usual fashion. For the second case however, the RTR must deal with NAT traversal. It will thus use the public IP address and the public port of the NAT to encapsulate the packets, instead of the usual LISP data port, 4341.

As a final step, let us review the state of the LISP Cache and of the LISP Database at the RTR

when it is serving as a relay for a NATed device. Special entries need to be inserted in both of them in order for NATed Data Forwarding to happen seamlessly and to integrate well with the existing workflow.
Upon reception of an ECM MapRegister at the RTR, we already said (Chapter 4 Section 2.3) that the RTR is recording a certain number of information about the NATed device in order to be able to later reach it. We also said that these operations are carried out by the SetNatedEntry method. The different entries that are inserted by SetNatedEntry and their use can be found in Table 5.1.

Location	Entry	Purpose
Cache	(EID $\longrightarrow$ NAT translated address)	Forward data packets towards NATed device.
Database	(EID $\longrightarrow$ RTR RLOC)	Answer MapRequests on behalf of NATed device.
Cache	(LRLOC $\longrightarrow$ NAT translated address)	Forward control packets (addressed to LRLOC) to NATed device.

Table 5.1: RTR LISP Cache and Database

The two entries inserted in the Cache are specific type of entries meant to deal with NAT traversal. The class MapEntry has been adapted to record the necessary fields, namely, the NAT public port, the RTR own RLOC, and the NATed LRLOC.

3.4 LISP-MN Helper

As we explained in Section 2.4, Helpers are useful in ns-3 to deal with the low-level details of models and make the life of the script writer easier. Helpers to install the LISP Data Plane and Control Plane, as well as the MR/MS applications already existed, but no helper for the LISP-MN had been written yet. Therefore, we implemented a LISP-MN Helper for setting up LISP mobile nodes, and manage handover procedures from one attachment point to another in the network.

The LISP-MN Helper exposes a certain number of methods that the script writer can use to easily set up the simulation. Let us review in Listing 5.2 the usage of the Helpers. On line 4, the LISP-MN Helper is instantiated with the node we want to be a LISP-MN, and its EID address. Then, on line 11, we define what nodes in the network should be the attachment points for the LISP-MN, and we set them on line 12 with the SetRoutersAttachmentPoints () method. On line 13 and 15, we define the subnet for each router, which will be used for DHCP considerations.
Then, on line 20, a handover is scheduled from the first attachment point (node 3) to the second attachment point (node 5). Finally, the script writer calls the Install () method to bind everything together.

```
1  [...]
2
3  /* ---- LISP-MN Setup ---- */
4  LispMNHelper lispMnHelper(nodes.Get (MN), Ipv4Address ("172.16.0.1"));
5  lispMnHelper.SetPointToPointHelper (p2p);
6
7  lispMnHelper.SetDhcpServerStartTime (StartTime);
8  lispMnHelper.SetEndTime (EndTime);
9
10 // Set attachment points
11 NodeContainer attachmentPoints = NodeContainer (nodes.Get (3), nodes.Get (5));
```

```
12 lispMnHelper.SetRoutersAttachmentPoints (attachmentPoints);
13 lispMnHelper.SetRouterSubnet (Ipv4Address ("10.1.2.0"),
14                              Ipv4Mask ("255.255.255.0"));
15 lispMnHelper.SetRouterSubnet (Ipv4Address ("10.1.3.0"),
16                              Ipv4Mask ("255.255.255.0"));
17
18 // Schedule handover from attachment point 0 to attachment point 1
19 // (at time HandTime)
20 lispMnHelper.ScheduleHandover (0, 1, HandTime);
21 lispMnHelper.Install ();
22
23 [...]
```

Listing 5.2: LISP-MN Helper usage

It took only 10 simple lines of code to set up the LISP-MN as well as the whole handover procedure between attachment points. Previously, when no Helper existed, the script writer needed roughly 100 lines of code, and had to deal with low-level details about NetDevices and VirtualNetDevices, Interfaces, DHCP servers and clients, IPv4 and MAC address assignments, routing tables, etc.

3.5 Unit Tests

ns-3 is a major piece of software, and as such, it requires various tests to ensure the quality, the correctness, the performance, and the robustness of its implementation. ns-3 provides a fully-featured testing framework to encourage its contributors to write tests for the models and code they add into the system. This tool allows for model validation, i.e., the verification that a particular model is implemented according to its specifications.

We wrote different tests for both the NAT model and the LISP model, as none had been written yet. Testing is of the utmost importance to verify the implementation of a model and find any remaining errors. Additionally, testing is also useful once a model has been verified to avoid any regression, and keep the system maintainable. Indeed, the system must remain valid over its lifetime, and it is important to detect any break of functionality when a change is integrated into the system. These tests will be useful for future developers who wish to modify the code without introducing regressions in existing functionalities.

In the next sections, we review the different unit tests we implemented, as well as a brief summary of the features they cover, for both NAT and LISP models. The source files for the unit tests can be found in src/internet/test and src/internet/test/lisp-test respectively.

NAT model

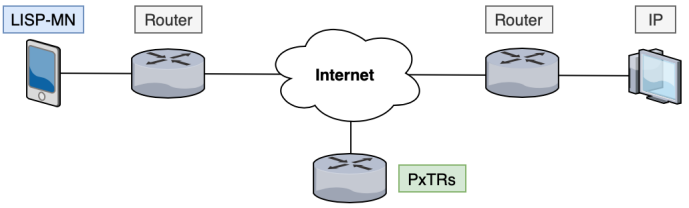
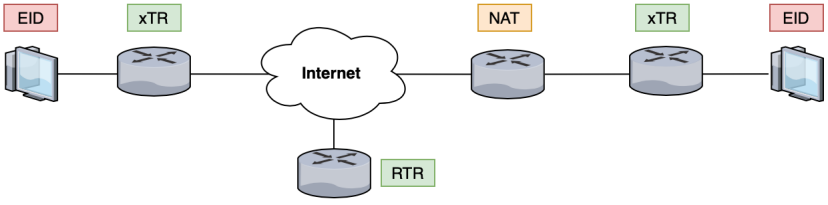
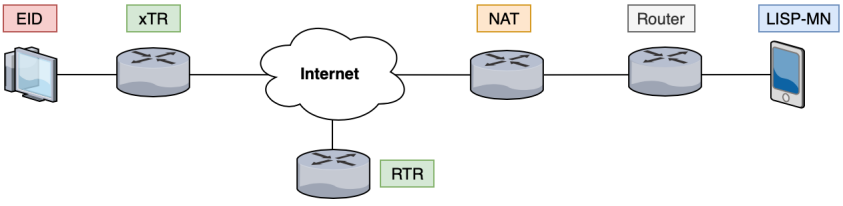
The unit tests implemented for NAT can be found in Table 5.2.

LISP model

For the LISP model testing, as we arrived quite late into the development stage, we decided to implement broad tests for different LISP configurations and scenarios, rather than to test each feature individually. For each configuration, we check that the flow is established between the two ends and that packets are flowing correctly between them. We review in Table 5.3 the different use cases that are covered.

Unit Test	Description
NatDynamic	Tests that a packet exiting the subnet through NAT is correctly modified, and that its returning packet is also correctly modified (and forwarded) according to the dynamic rule.
NatDynamicDrop	Tests that no packet can enter the NATed subnet when no dynamic rule is defined for them, and that such packets are dropped.

Table 5.2: NAT Unit Tests

Unit Test	Description
Static Scenarios	
SimpleLisp	
MnLisp	
PxTRs	
MnBehindXtr	
XtrBehindNat	
MnBehindNat	
Handover Scenarios	

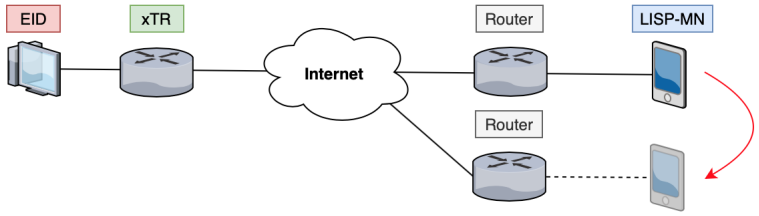
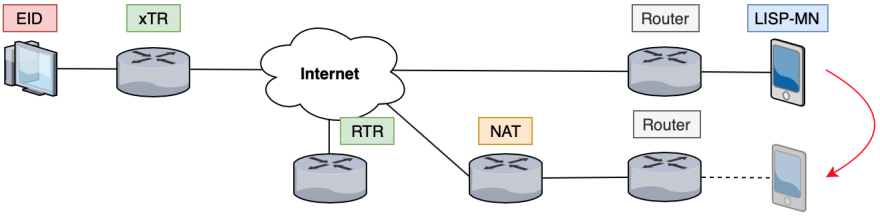
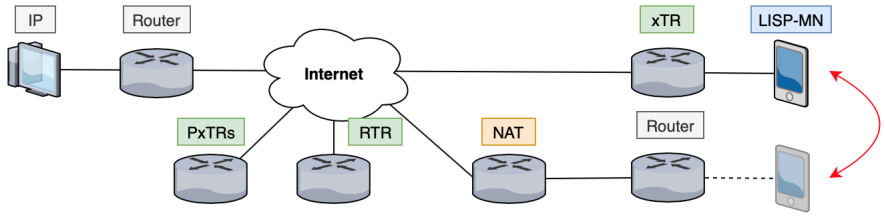
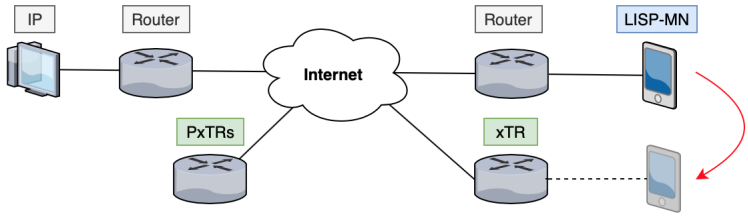
SimpleHandover	
NonLisp-NatHandover	
LispNatProxyHandover	
NonLispLispProxyHandover	

Table 5.3: LISP Unit Tests

A lot of network configurations have been covered to ensure robustness of the implementation in various cases. Additionally, they provide code examples of how to set up the various scenarios. This is useful for the beginner who wishes to setup a scenario, as LISP scripts require quite a lot of configuration for them to work properly, in particular regarding the static routes, LISP Database, and RLOCs configuration.

Chapter 6

Analysis of LISP mobility with NAT

In this chapter, we review several handover scenarios between different types of sites and compare their performance. In particular, we will have a look at the performance of NAT Traversal for LISP traffic compared to classic traffic with no NAT deployment. We first conduct a theoretical analysis on the different scenarios, and explain our methodology for the simulations we conducted. Then we present the results of the simulations, and draw conclusions.

1 Simulation scenarios and methodology

We are interested in studying the different handover scenarios that can be found in Figure 6.1. The LISP-MN is first situated in a LISP site and is downloading content from the Content Server (typically, the streaming of a video). This server is situated in a non-LISP site, and thus requires the use of PxTRs to communicate with the LISP-MN. In a second time, the LISP-MN roams in a non-LISP site behind a NAT (typically behind an operator box at home) (1). In that case, the use of an RTR is also required in order to transmit traffic through the NAT. Then, the LISP-MN roams again in a second non-LISP site (2). Finally, the LISP-MN arrives in a LISP site (3). We

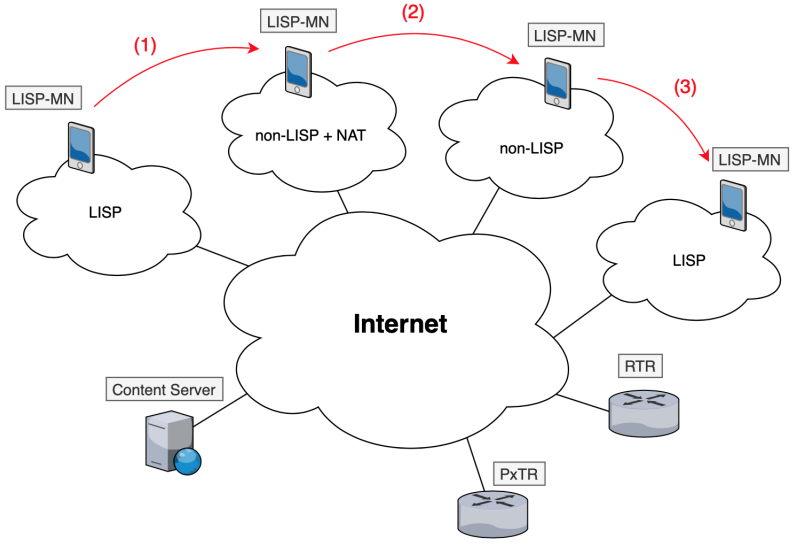


Figure 6.1: Handover scenarios

will study this mobility scenario in terms of control message overhead, and in terms of handover

delay. This handover procedure happens transparently for the transport-level communication and the connection is not interrupted.

1.1 Theoretical analysis

We will now review and compare the control messages exchanged for the three handover scenarios. Note that the handover procedure is fully determined by the site of arrival of the LISP-MN. The starting site has no impact on the handover procedure and the messages exchanged.
There is one exception though: when the LISP-MN starts a communication behind a NAT, and then roams into another site with no NAT, the LISP-MN will have to send an additional MapRequest to the Mapping System to resolve the mapping of its correspondent. Indeed, when NATed, the LISP-MN was encapsulating everything to the RTR, who had the responsibility of resolving mappings on behalf of the LISP-MN.

Arrival into a non-LISP + NAT site (1)

In Figure 6.2, the precise handover procedure when roaming from a LISP site to a non-LISP site behind a NAT can be found.

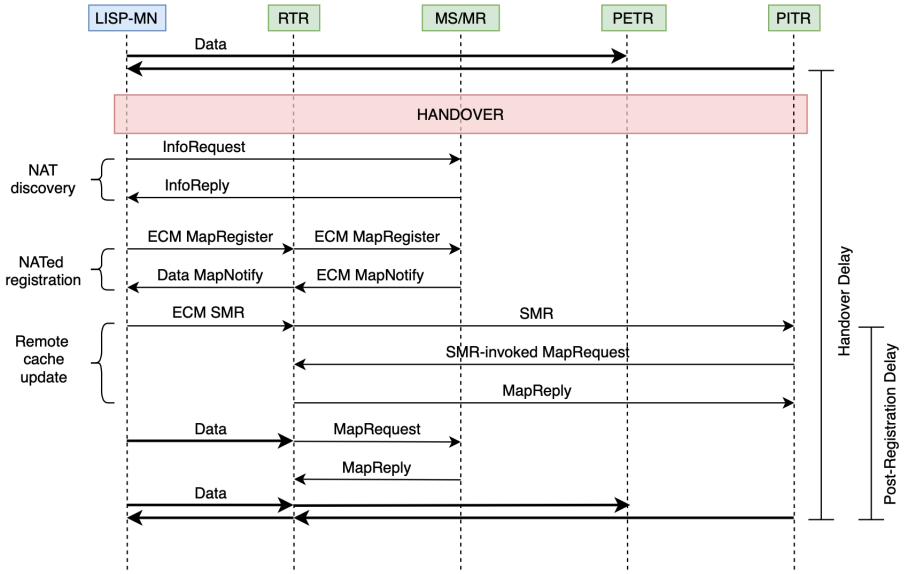


Figure 6.2: Handover procedure when LISP-MN roams behind a NAT

Firstly, the connection is established and data is flowing from the LISP-MN to the content server through the PxTRs. Notice that the traffic is asymmetric: LISP-MN sends its packets to the PETR so that the PETR can decapsulate them and forward them natively in the Internet towards the content server. Conversely, LISP-MN receives traffic from the PITR, because it acts as an ITR for non-LISP sites and encapsulate non-LISP traffic towards LISP sites.
Then the mobile node roams into a non-LISP site that is behind a NAT, and the handover procedure is started.
The first step for a LISP device when receiving a new (L)RLOC is to check whether it is NATed or not. For this purpose, two control messages (InfoRequest and InfoReply) are exchanged between the LISP-MN and the *Map server*. When the LISP-MN realizes it is situated behind a NAT, the

NATed Registration Procedure is triggered (see Figure 4.2 for the message details). Once registered to the RTR and to the *Map Server*, the LISP-MN cannot resume sending data just yet, it must first update all the remote caches that have stored the previous mapping. To do so, it triggers the SMR procedure for all remote sites that have been communicating with it (in this case, the PITR). This remote cache update procedure adds four control messages to the count.
Finally, LISP-MN sends data encapsulated towards the RTR, that is now responsible for encapsulating the packets towards the right ETR. This flow is new for the RTR, meaning that it probably doesn't have mapping information for the destination EID. It issues a MapRequest, answered with a negative MapReply (as the mobile is communicating with a non-LISP site), before the data flow can be resumed between LISP-MN and the content server.

All in all, the control message overhead of the handover procedure is 12 messages. We can also define the overhead in terms of bytes. As the length of the different control messages varies according to the number of records, the number of RLOCs in those records, and the MAC algorithm that is used; we will define a lower bound on the number of bytes exchanged. For each message, we consider the minimum number of bytes for the message to be valid, i.e., a message with no more than one record, and no more than one RLOC per record. The authentication data length is assumed to be 32 bytes. The number of bytes for each type of message can be found in Table 6.1.

Message type	Nb of bytes
MapRequest	32
MapReply	40
MapRegister	76
MapNotify	76
InfoRequest	64
InfoReply	96

Table 6.1: Number of bytes for each control message

Then, to the number of bytes for a message, we need to add the encapsulation overhead. The different values can be found in Table 6.2. For a simple IP packet, the minimun IP header length is 20 bytes, plus 8 bytes of UDP header. For the ECM encapsulation, we have two times that quantity, plus the ECM header itself, which is 4 bytes. And finally, for the Data encapsulation, this is the same computation as for the ECM header, except that the the data header is 8 bytes long.

Encapsulation	Nb of bytes
Simple	28
ECM	56 + 4 = 60
Data	56 + 8 = 64

Table 6.2: Number of bytes for each encapsulation

If we take back Figure 6.2, and apply the values that we just defined for each type of message, and for each encapsulation, we find a lower bound on the message control overhead of 1172 bytes (672 bytes of control messages, 500 bytes of encapsulation).

Let us now define the handover delay that we will be measuring in our simulation. It is defined as the delay between the last packet received by the LISP-MN before the handover procedure, and the first packet received by the LISP-MN after the handover procedure. This is

illustrated in Figure 6.2. This handover delay is composed of the NAT discovery process, the NATed Registration, the remote cache update, as well as the DHCP procedure to acquire a new (L)RLOC.

Arrival into a non-LISP site (2)

By comparison, the handover procedure when roaming into a non-LISP site (without any NAT) can be found in Figure 6.3.

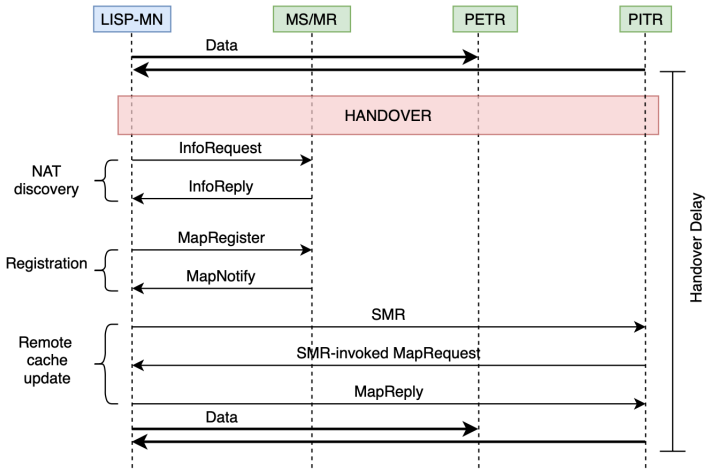


Figure 6.3: Handover procedure when LISP-MN roams into a non-LISP site

As can be seen, the procedure is less complicated, and will most probably result in a shorter handover delay, with only 7 control messages exchanged, instead of 12. In terms of bytes, the overhead is at least 612 (416 bytes of control messages, 196 bytes of encapsulation). The handover procedure is the same as for the first scenario, except that the overhead of the RTR is not present anymore: the messages can flow directly between participants without having to be relayed by the RTR. Moreover, a MapRequest to the Mapping System is also sparred, as the LISP-MN already knows to which ETR to encapsulate, while the RTR had to discover it.
The handover delay is defined in the same way as for the first handover scenario.

Arrival into a LISP site (3)

Finally, let us have a look at the handover procedure when roaming into a LISP site on Figure 6.4.

The procedure is very similar to scenario (2) (see Figure 6.3) but with some additional overhead. The NAT discovery and Registration process happen in the same way, except that each packet is intercepted by the xTR and encapsulated a second time, adding to the delay. This second encapsulation is represented by a red cross at the xTR. The remote cache update procedure is also longer than in scenario (2), due to the fact that the RLOC assigned to the LISP-MN is in fact part of the EID space of the xTR, and is therefore an LRLOC that is not routable. For this reason, the remote ITR (the PITR in this case) must first query the Mapping System to be able to send the SMR-invoqued MapRequest. More precisely, the PITR, upon receiving an SMR from the LISP-MN, wishes to send an SMR-invoqued MapRequest back to the LISP-MN. However, the LRLOC of the LISP-MN is not routable. Therefore, the PITR first sends a MapRequest to the

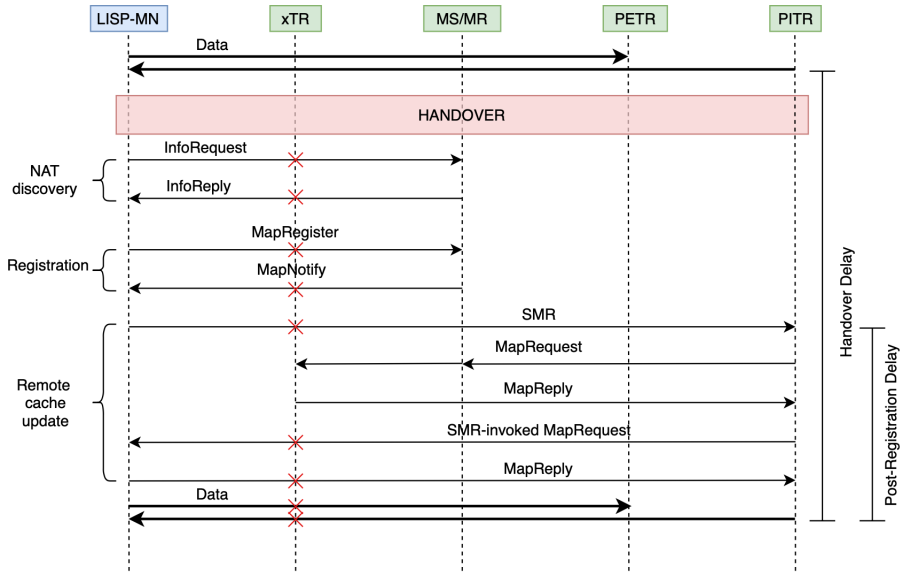


Figure 6.4: Handover procedure when LISP-MN roams into a LISP site

Mapping System. Once it has the necessary mapping, it can send the SMR-invoked MapRequest, encapsulated towards the xTR.

The number of control messages exchanged is 10, slightly more than for scenario (2) and slightly less than for scenario (1). As for the two previous scenarios, let us compute the control message overhead in terms of bytes. This time, all messages from and to the LISP-MN suffer double encapsulation (data encapsulation), because the LISP-MN is situated into a LISP site. After computing, we get an overhead of at least 1052 bytes (520 bytes of control messages, 532 bytes of encapsulation).
The handover delay is defined in the same way as for the first two scenarios.

A summary of the number of control messages exchanged (as well as the number of bytes) in each of the scenarios can be found in Table 6.3.

Scenario	(1)	(2)	(3)
# control messages	12	7	10
# bytes	≥ 1172	≥ 612	≥ 1052

Table 6.3: Handover overhead

1.2 Simulation setup

In this section, we explain the details of the simulation setup and we justify our choices. In particular, we see how the handover procedure between sites is carried out, and what type of traffic is used during the simulation. Additionally, we also justify why we analyse only the handover delay, and why we don't attempt to draw any further conclusions on application-level consequences.

Handover procedure

Regarding the handover procedure between two attachment points (*router1* and *router2*, for example), the LISP-MN is actually physically connected to both of them, and has one interface for each router. First, the interface to *router1* is up, while the interface to *router2* is down. This simulates a situation where the LISP-MN is in the subnet of *router1*.
Then, during the handover, the interface to *router1* is set down, and the interface to *router2* is set up. This simulates the roaming of the LISP-MN in the subnet of *router2*. Of course, this is a very artificial way of simulating a roaming event. In reality, the connection of the LISP-MN to the second subnet would not be immediate, as it is implemented in the simulation. However, this immediate connection allows us to focus solely on the delays introduced by LISP, without any exterior interference.

Type of traffic

As to the type of traffic used for the simulation, we chose to use a constant bit rate traffic over UDP. This may look like an unrealistic assumption to use this kind of traffic to simulate the streaming of a video, but it serves our purpose of evaluating the handover delay as accurately as possible.

The traffic generated by the streaming of a video is far from a constant bit rate traffic. Indeed, streaming is composed of two transmission phases: a buffering phase, or initial burst, where a certain amount of video is downloaded as fast as possible; and a throttling phase, where parts of the video are requested on demand by the client, when its buffer gradually becomes empty as the video is played out.
Roughly, this kind of traffic can be modeled with an On-Off application, which is characterized by periods of data transmission followed by periods where no traffic is generated.

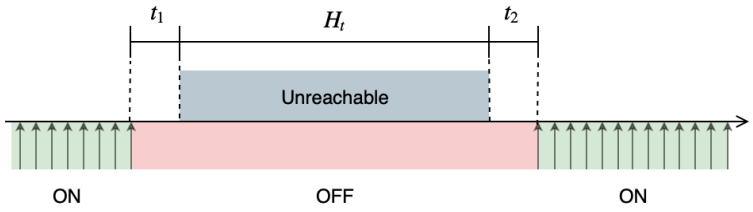
However, this kind of traffic doesn't allow us to measure precisely the handover delay. Indeed, what we want to know is the time necessary for the LISP-MN to be reachable after a handover. This time has been defined as the delay between the last packet received by the LISP-MN before the handover procedure, and the first packet received by the LISP-MN after the handover procedure. If we use an On-Off application, it may happen that the LISP-MN's unreachability period overlaps with an Off period of the traffic, as can be seen in Figure 6.5a. In such a case, we would measure a handover time of $H_t + t_1 + t_2$, while the real handover time (i.e., unreachability period) is H_t.

Conversely, by using a constant bit rate traffic (1024bytes/0.01s), we can detect more accurately the handover time H_t with a maximum error of 2ϵ, where $\epsilon = 0.01$ s (assuming perfect conditions, where the network doesn't introduce additional delays, and where packets arrive at a constant rate at the LISP-MN).

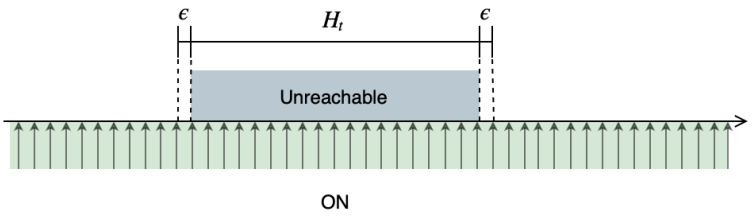
In practice, even with imperfect conditions, the handover time measure with a constant bit rate application still stays more accurate than with an On-Off application. In conclusion, by choosing a cbr traffic, we remove from the equation other considerations (such as streaming considerations), which allows us to only study the effect that interests us, namely, the handover delay.

Application-level consequences

The basic use case that we introduced earlier is the streaming of a video. Therefore, it could have been interesting to have a look at the content quality, as observed by the mobile user when



(a) On-Off traffic



(b) Cbr traffic

Figure 6.5: Handover time measure methodology

watching the video. We can define the content quality as the number of stalling events (i.e., the number of times the video freezes - and for how long - because of an empty buffer). However, we will see that these effects are not significant to study in the context of a LISP handover, and that no strong conclusions can be drawn from simulations, or even from real observations.

Indeed, we can be faced with different scenarios. It could happen that the handover occurs during a period where the buffer is full, and thus where no traffic is generated. In that case, the handover would occur completely transparently to the user, without any bad effect on the content quality (even though the handover delay may be quite long). Or, on the other hand, it could happen that the handover occurs right at the time where the buffer becomes empty and that new video chunks are requested. In that case, a stalling event would be more likely.
What situation the user will be in most of the time is completely random, and doesn't depend on any parameter, or anything linked one way or another to LISP. Indeed, the time at which the handover happens (buffer full, or buffer empty) is completely random. Therefore, it would be meaningless to affirm that 15% of the time (for example), the user experiences a stalling event. The only thing we can say is that, in the worst case (buffer empty), a user would experience a stalling event of H_t seconds.

In conclusion, the only significant parameter that we can study is the handover delay. Trying to conclude anything else about application-level consequences makes no sense.

1.3 Simulating a realistic network

Simulation is useful to evaluate new technologies, to verify and analyse protocols and systems, without the need to develop a complete test bed, which is most often very costly. This is why we decided to use the ns-3 Network Simulator. However, although ns-3 provides robust models of how packet data networks work and perform, we must take care to set up a realistic simulation, whose results are meaningful and can be applied to reality.

Although ns-3 is very scalable and efficient, it is of course impossible to model the entire Internet in it. Our simulation must necessarily use a reduced number of nodes and do some abstractions. For the simulation to be as realistic as possible, we will be using some real data that we

introduce in the simulator for several aspects of the scenario, namely, the Mapping Distribution System, the use of proxies, the use of RTRs, and the SMR exchange between xTRs.

We review in the next sections the datasets that we use, and how we model the delays introduced by the use of those elements. We also provide a visual summary of the places where artificial delays have been introduced.

Simulating the Mapping Distribution System

We already talked about the Mapping Distribution System (MDS) of LISP in Chapter 3 Section 1.3, saying that LISP relies on such a Mapping System to obtain mappings between EID and RLOC(s). The current Mapping System is *LISP Delegated Database Tree* (LISP-DDT) [15] and is organised as a distributed hierarchical database, mirroring a DNS-like architecture. Such a distributed database is composed of multiple nodes and organised in multiple levels.

In ns-3 however, the entire Mapping System is not composed of a full DNS architecture. Rather, it is simulated with only two nodes: a unique *Map Resolver*, to query the Mapping System, and a unique *Map Server*, to forward MapRequests to authoritative ETRs. Therefore, it is necessary to simulate the response time of the Mapping System by introducing artificial delays in the *Map Server* response time, in order to simulate the entire process of the Mapping System lookup. To do so, a Random Variable is used to draw the response time from.

In order to choose a fitting distribution for the response time, we base our choice on the work of Coras et al. [25]. They have evaluated the performance of the LISP Beta Network [26] regarding the Control Plane (i.e., the Mapping System) and found that the Mapping System typically provides reliable performance and relatively low resolution delays. The resolution delay is defined as the RTT a packet requires to travel to the *Map Resolver*, cross the Mapping System, reach the authoritative ETR, and return to the requesting ITR. They found the median to be less than 200ms, with only 10% of the requests exceeding 500ms.

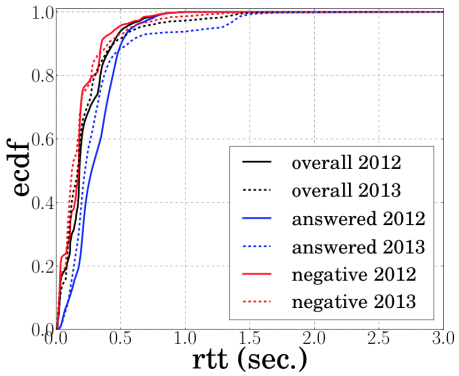


Figure 6.6: RTT distribution
[25]

The Cumulative Distribution Function (CDF) of the RTT can be found in Figure 6.6 for 2012 and 2013. answered corresponds to real mappings, and negative corresponds to negative MapReplies. The RTT for negative MapReplies is often lesser than for classic MapReplies because the negative MapReply typically comes from a DDT node, while the classic MapReply comes from an ETR, and thus travels a bit further into the Mapping System hierarchy.

In ns-3, there is a convenient way to define a Random Variable based only on its CDF, with the EmpiricalRandomVariable class. Therefore, we used the overall 2013 curve of Figure 6.6 to

define the Random Variable used for the response time of the *Map Server*.

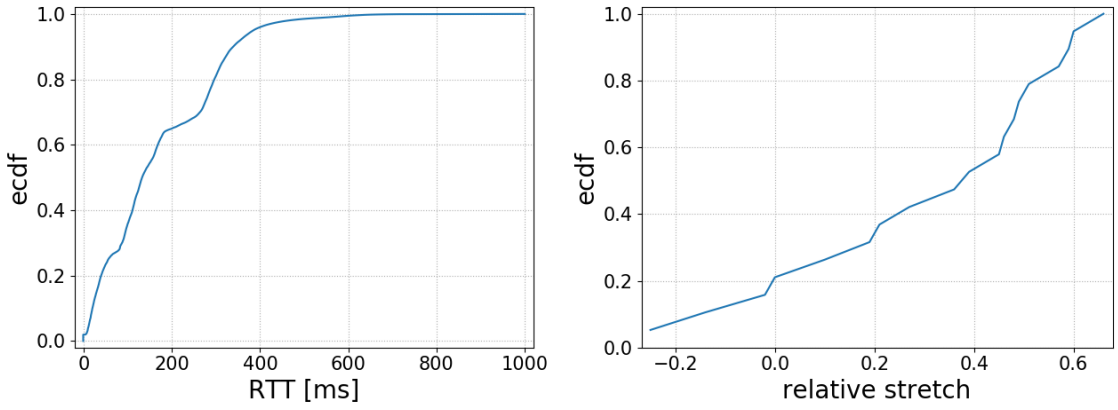
Simulating PxTRs

In Chapter 3 Section 1.4, we introduced the interworking mechanism that is used for communication between LISP sites and non-LISP sites. This mechanism introduces two network elements, the PITR and the PETR, that act as ITR and ETR for non-LISP sites. These elements are deployed worldwide, outside the edge domains, in the core of the Internet, and are geographically spread apart.

Several studies have already been conducted on PxTRs ([16], [25]) and show that this interworking mechanism has a global negative impact on performance. Indeed, their use introduces a path stretch as well as additional delays in the traffic. In particular, Coras et al. [25] show that there is an increase in the delay by at least 20% for 70% of the EID space. They also confirm the intuition that the placement of PxTRs in the topology, as well as the BGP policies, are the determining factors in the path and delay stretch introduced by the use of proxies.

In ns-3 however, we cannot simulate the entire Internet topology, neither the placement of PxTRs in that topology, nor the BGP policies. Therefore, we must necessarily introduce artificial delays, as we did for the MDS, to simulate the use of PxTRs. This artificial delay will be introduced in the processing time of the PITR, to simulate the time required to travel to the PETR, reach the remote host, arrive at the PITR, and finally return to the origin xTR.

We searched for a ready-to-use dataset giving the distribution of delays when using proxies, but unfortunately, we were unable to find one. Therefore, in order to choose a fitting distribution, we will combine two datasets, that can be found in Figures 6.7a and 6.7b:



(a) RV1 (b) RV2

Figure 6.7: PxTRs delays distribution

- RV1: A dataset of RTTs from a vantage point to several locators, from Saucez et al. [27].
- RV2: A dataset of relative delay stretch due to interworking, from Coras et al. [25].

The relative delay stretches of RV2 are computed according to the following formula:

$$\rho = \frac{\hat{d}_p - d_n}{d_n}$$

where d_n is the delay between two xTRs without the intervention of proxies, and where $\hat{d}_p$ is the delay obtained if the traffic transits through proxies.

Obviously, these relative stretches alone can not directly be used for the simulation. Indeed, they don't give real delays that we can introduce in the simulator, but rather, they only give the increase in the delay when using proxies. Unfortunately, we do not dispose of the raw data used to compute ρ, i.e., we do not dispose of the distributions of $\hat{d}_p$ nor d_n. This is why we need the second dataset, RV1, that gives us RTTs of RLOCs, or in other words, the delay between two xTRs without the intervention of proxies.

In practice, to compute the delay introduced by PxTRs in ns-3, we will use two EmpiricalRandomVariables, each representing the distributions of dataset RV1 and RV2. We draw a time from RV1, which gives us the time between two xTRs without the intervention of proxies, and we multiply it by a coefficient drawn from RV2, which represents the delay stretch. That way, we are able to simulate the time that is needed for traffic to transit through proxies.

Simulating RTRs

As we explained in Chapter 5, the NAT extensions [8] proposed for NAT traversal of LISP traffic are still at the early development stage. To the best of our knowledge, no investigation of NAT traversal has been conducted. Therefore, we do not dispose of any data we could introduce in ns-3 for the use of RTRs. Therefore, we will use a uniform random variable in the interval [90-110]ms.

Simulating the SMR exchange

In Chapter 4 Section 1.2, we introduced the SMR procedure [5] that is used to update remote caches upon a roaming event. As a reminder, this procedure is composed of 3 messages exchanged between two xTRs: first an SMR is sent to the remote LISP device (whose mappings must be updated). In turn, this device will send a MapRequest, that will then be answered with a MapReply.

To simulate the delay for each packet of the exchange, we can use the dataset RV1 from Saucez et al. [27], which represents the distribution of RTTs between xTRs. The RTT is defined as the time needed for a packet to reach the remote xTR and come back to the sender. As such, we cannot directly use the data, because we need the delay from one end to the other, and not the RTT. We thus make the assumption that the delays to the remote xTR and back are symmetric, and divide the values by two.

In practice, in ns-3, we define an EmpiricalRandomVariable based on the RV1 dataset. To simulate the time needed for a packet to reach the remote xTR, we draw a delay from the random variable and divide it in two.

Artificial delays deployment

We now present in Figure 6.8 a summary of the random variables that have been deployed in the simulation in order to recreate a realistic environment.

Random variables are represented by red labels. We first have the MDS random variable that is used at the *Map Server* to simulate the lookup time of the entire Mapping System. Then, at the proxies, we have two random variables, RV1 and RV2. Combined, they represent the delay introduced by the use of proxies. The random variable RV1 is also deployed in each LISP device

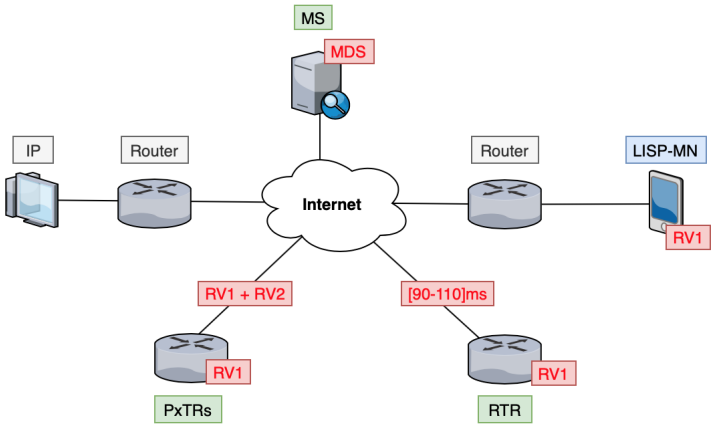


Figure 6.8: Random Variables deployment in simulation scenarios

to simulate the SMR exchange procedure. Finally, there is a uniform random variable deployed at the RTR, to simulate the time needed to make a detour through the RTR.

1.4 Independent replications of the simulation scenario

To evaluate the handover delay, we launched multiple runs of the scenario in order to get meaningful results. When using a simulator to evaluate a scenario, it is important to make sure the different instances of the scenario have been run independently of each other for the results to be significant. We will now briefly introduce the ns-3 Random Number Generator (RNG) and how it was used to convince the reader of the validity of the results, and to allow them to reproduce the results obtained.

ns-3 uses a (pseudo) RNG that produces a long sequence of random numbers based on the *seed* that it is given. The length of this sequence is called the *period*, and it is important in a simulation to guarantee that we will never cycle and repeat ourselves. This long sequence of numbers is divided into *streams* that are uncorrelated to each other. Each *stream* is itself partitioned disjointedly into uncorrelated *substreams*. The RNG produces 1.8×10^{19} *streams*, each partitioned into 2.3×10^{15} *substreams*, each containing 7.6×10^{22} random numbers. In total, the RNG produces 3.1×10^{57} uncorrelated random numbers before it cycles and repeats itself. A summary of this can be found in Figure 6.9.

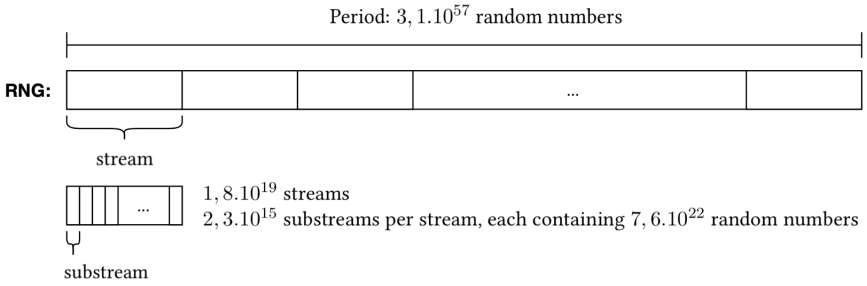


Figure 6.9: ns-3 Random Number Generator

The way ns-3 works is by assigning a *stream* to a Random Variable (RV) each time that a RV is created. Conceptually, this means that each RV has its own "virtual" RNG. Additionally, this means that in a simulation it is possible to instantiate the same number of RVs as there are

streams, i.e., 1.8×10^{19}. In our simulations, we used between 7 and 8 random variables, as can be seen in Figure 6.8, which is by far inferior to 1.8×10^{19}.

The Random Variable is then configured to use one of the *substreams* found in the *stream*, with the *run* number. In a simulation run, this means that each RV can be used to draw at most 7.6×10^{22} random numbers from it. In our case, random variables have been used for the MDS response time to MapRequests, for the SMR exchange, and for the RTT of data packets. The number of requests made to the MDS is clearly inferior to 7.6×10^{22}. The SMR exchange is limited to three messages. And the number of data packets exchanged during the simulation (less than a thousand packets) is also inferior to that number. Therefore, we are assured of the statistical soundness of a simulation run.

Finally, to actually run multiple independent replications of the same simulation scenario, the way to proceed is to advance the *run* number, which causes the RV to use a different *substream* from its main *stream*. As the different *substreams* are uncorrelated, the independence property of the multiple runs of the simulation is ensured, and this allows us to compute statistics on a large number of independent runs.

In our simulation, the RNG has been configured with a seed of 5, and 30 independent replications have been run through a bash script that advances the run number at each run.

2 Simulation results

In this section, we present the results we obtained for the different handover scenarios we introduced in Section 1. Finally, we have a look at the limitations of the simulations.

2.1 Analysis

Figures 6.10a, 6.10b, and 6.10c depict the global handover delay for each simulation scenario, namely, a handover between a LISP site and a non-LISP site behind a NAT, a handover between a non-LISP site behind a NAT and a non-LISP site, and a handover between a non-LISP site and a LISP site.

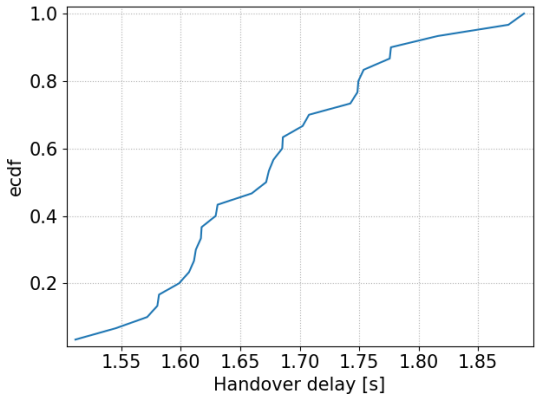
(a) Regarding the roaming into a non-LISP site behind a NAT, the handover delay is less than 1.75 seconds 80% of the time.

(b) For the roaming into a non-LISP site, the delay is less than 1.43 seconds 80% of the time.

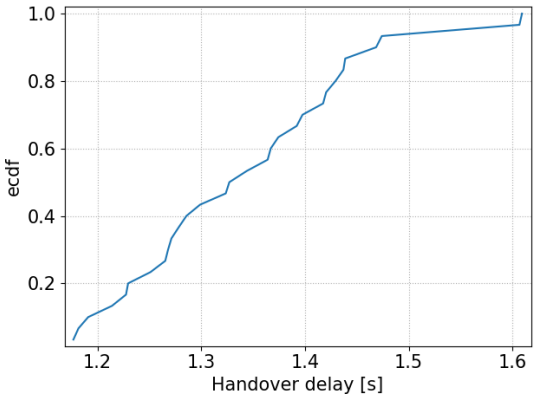
(c) And for the roaming into a LISP site, the delay is less than 1.9 seconds 80% of the time.

Let us first have a look at the difference between the roaming in a non-LISP site behind a NAT (1) and the roaming in a non-LISP site (2), in order to evaluate the impact of NAT traversal on LISP traffic. Globally, the roaming in a non-LISP site (2) has the smallest handover delay. This is not surprising as this scenario is the simplest one and suffers least overhead. Indeed, from the theoretical analysis (see Figure 6.3), scenario (2) has the smallest number of control messages exchanged, as well as the smallest number of bytes. The main part of the handover is composed of the SMR exchange.

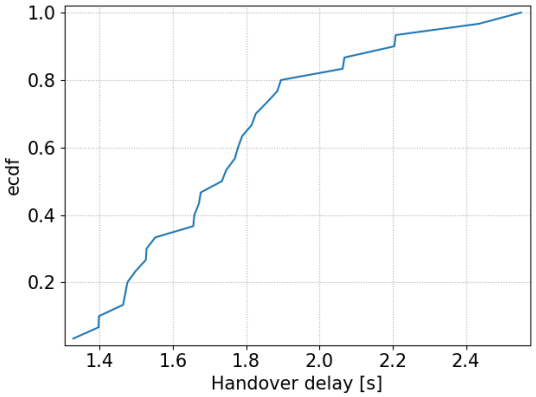
On the other hand, the roaming into a non-LISP site behind a NAT (1) suffers the most overhead in theory (see Figure 6.2), with 12 control messages exchanged, for a total of at least 1172



(a) Arrival in a non-LISP site behind a NAT (1)



(b) Arrival in a non-LISP site (2)



(c) Arrival in a LISP site (3)

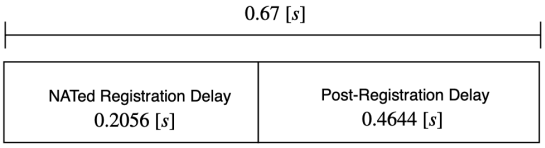
Figure 6.10: Simulation results

bytes. This is reverberated in the handover delay that we observe in Figure 6.10a, which is higher than for scenario (2). As expected, the NAT deployment has a negative impact on the handover delay, due to the additional complexity of the handover procedure in itself (NATed Registration procedure), as well as the RTR relay that must be used for all traffic, and that introduces a path stretch, as well as additional delays.

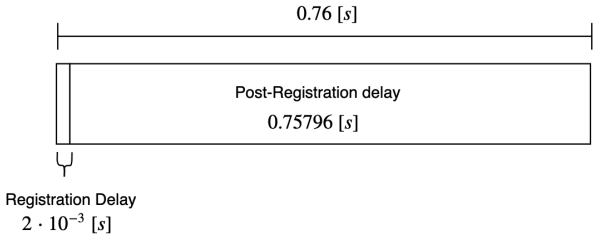
Now that we reviewed the impact of NAT traversal for LISP traffic, we can compare with the roaming of a LISP-MN in a LISP site (3). From the theoretical analysis (see Figure 6.4), scenario (3) suffers from quite some overhead as well, with 10 control messages exchanged, for a total of at least 1052 bytes. By comparison, scenario (1) had 12 control messages for at least 1172 bytes. Although scenario (1) suffers the most overhead in theory, the handover towards a LISP site (3) has the largest handover delay most of the time, as can be seen in Figure 6.10c, with a median of 1.76 seconds for scenario (3), and a median of 1.67 seconds for scenario (1).

This is surprising, because we expected NAT traversal to have a larger impact on the handover delay than scenario (3). However, these results can be explained from the simulation. If we look back at Figures 6.2 and 6.4, which depict the control message exchange for scenario (1) and (3) respectively, we see that both scenarios are almost identical: There is one InfoRequest/Reply exchange, one SMR exchange, and one MapRequest made to the MDS. Both the SMR exchange and the request to the MDS are part of the Post-Registration Delay that is represented in the

Figures. The main difference is the Registration Procedure, which is NATed in one case, and not in the other.
From their similitude, we would expect both scenarios to have the same delays for each aspect of the handover (InfoRequest/Reply, SMR, MapRequest), except with a larger delay for the Registration Procedure in case of a NATed device, overall resulting in a larger global handover delay for scenario (1). However, this is not what we observed in the simulation results.



(a) Roaming into a non-LISP site behind a NAT



(b) Roaming in a LISP site

Figure 6.11: Handover delay split into its main contributions

In order to investigate further these results, we present in Figure 6.11 the mean global handover delay split into its main contributions, for scenario (1) and scenario (3). We only represent the mean Registration delay and the mean Post-Registration delay because all other contributions (DHCP, InfoRequest/Reply, proxies) are identical for both scenarios. The first thing that we can observe from Figure 6.11 is that the NATed Registration Procedure is indeed longer than a classic Registration Procedure with no NAT deployment. This was expected of course, since the RTR introduces a stretch.
Then we can see that the Post-Registration delay (composed of an SMR exchange and a MapRequest to the MDS) is shorter in the NATed case than in the classic case. This is unexpected, because the Post-Registration delay is made up of the same events in both scenarios. From this basis, we could thus expect similar results. However, this is not the case, and the delay is shorter in the NATed case than in the classic case.
This surprising result can actually be explained by the fact that in practice, for scenario (1), the MapRequest to the MDS is made in parallel to the SMR procedure. The delay for the MapRequest thus overlaps with the SMR exchange delay. While for scenario (3), the SMR procedure can only complete once the MapRequest to the MDS has been resolved, making these events sequential and not parallel.
This explains why the global handover delay is shorter most of the time for scenario (1), conversely to what was predicted.

To conclude our analysis of the simulation results, we observed that the roaming into a LISP site (3) suffers the largest handover delay most of the time, followed by the roaming into a non-LISP site behind a NAT (1), followed by the roaming into a non-LISP site (2).

2.2 Limitations

As we already explained, simulation is a great tool to evaluate new protocols and systems, but one must take care to setup a realistic scenario for the results to be meaningful. In Section 1.3, we reviewed the datasets that we used to introduce artificial delays in the simulation in order to capture the ins and outs of real LISP traffic. However, for the NAT traversal part (NATed Registration Procedure, relaying of all traffic through the RTR), we did not dispose of any datasets, simply because the NAT extensions of the LISP protocol have never been studied in the field, to the best of our knowledge. We thus fell back to a simplistic uniform random variable to try to introduce some delays due to the use of an RTR.
However, such a variable probably fails to capture the delays introduced by this new network element. Indeed, the impact of RTRs probably depends on their placement in the topology, much like the impact of PxTRs does. Additionally, for a scenario where both RTRs and PxTRs are used, the path stretch (and thus additional delays) will also depend on the placement of PxTRs and RTRs relatively to each other. As no study of RTRs has been done, and even less studies of the use of RTRs and PxTRs jointly, the effects of both elements combined are difficult to evaluate with a simple uniform random variable.
From the results of the simulations, we found that NAT traversal (1) has a smaller impact on the handover delay most of the time than for scenario (3). However, in a real environment, it would be reasonable to expect the opposite, with scenario (1) having a larger impact than scenario (3). Indeed, proxies already introduce some form of triangular routing. Using RTR besides proxies may deteriorate even further the performance, as yet another detour is introduced in the traffic.

Finally, regarding the artificial delays introduced by the use of PxTRs, the datasets that were used are not perfect either. Indeed, we combined two datasets, RV1 and RV2, because we did not find a ready-to-use dataset giving the distribution of delays when using proxies. It would have been more accurate to directly use such a distribution. However, even though these datasets have some limitations, we can say that they manage to model the effects of proxies reasonably well.

Chapter 7

Conclusion

Internet routing protocols have not evolved much over the years, despite the Internet huge growth and its deep changes in terms of network usage, such as Traffic Engineering (TE), mobility, or multi-homing. These factors are now threatening the scalability of the current Internet architecture and addressing system. To remedy these issues, a new kind of network-level protocol arose, based on the *identifier/locator separation* paradigm. LISP is one of those protocols and is the most widely deployed nowadays. LISP advocates for the separation of the *identifier* and the *locator* roles of the IP address, introducing two distinct address spaces: the EID (*identifier*) address space, and the RLOC (*locator*) address space. In order to bind the two address spaces together, and to stay compatible with legacy Internet, LISP uses encapsulation to tunnel packets from one border router to another. A Mapping System is used to map an EID to a set of RLOCs, and effectively encapsulate packets in the core of the Internet.

Afterwards, LISP-MN and LISP+NAT have been introduced to bring mobility extensions to LISP, and provide a NAT traversal mechanism for LISP mobile nodes, respectively. However, these propositions remain mostly limited to the theoretical level and, to the best of our knowledge, no investigation of their combined performance has ever been conducted.

1 Contributions

In this master thesis, we aimed to provide a first look at NAT traversal for LISP mobile nodes through simulations on the ns-3 Network Simulator. We studied several handover scenarios between different types of site (LISP, non-LISP, non-LISP behind NAT) and compared their performance regarding the handover delay, and the overhead of the handover procedure. Our experiments led us to believe that NAT traversal does have a negative impact on LISP traffic, which is not surprising due to the complexity brought by LISP+NAT.

To perform our experiments, we adapted the ns-3 Network Simulator to incorporate a NAT model, proxy functionalities, and LISP+NAT extensions to the existing LISP model. Our work provides PxTRs and RTRs implementation, as well as the NATed Registration Procedure, and NATed Data Forwarding. Additionally, we wrote a LISP-MN Helper, meant to help the script writer to easily setup a scenario with mobile nodes and handovers. Various unit tests for the NAT and LISP models have also been added to the ns-3 testing framework.

We also defined the novel SMR procedure used to update the remote caches after a handover, in case the LISP node is NATed. Indeed, all works about NAT traversal focused on static scenarios, with no roaming and no handover. This important aspect of mobility, i.e., the update of remote nodes' state after a handover, was left completely unspecified.

2 Future work

All aspects of the work to evaluate LISP mobility with NAT have been achieved: NAT, proxy, and LISP+NAT models have been implemented in the ns-3 Network Simulator. The analysis itself has been conducted in the most realistic way possible, with the use of datasets to simulate real LISP traffic. However, one can always find refinements to bring to one's work.

Regarding the implementation in ns-3, one idea would have been to integrate LISP processing into the Netfilter framework. Indeed, as of now, a lot of LISP-related code has been written inside the IP stack functions (Receive, Send, Forward, and LocalDeliver). This is of course functional, but it is not the most modular way of doing things, as IP code shouldn't be mixed with LISP code. However, as it was already implemented there when we started our work, we only extended it. One idea would have been to rework the entire structure, leveraging on the Netfilter framework that we integrated into ns-3. Indeed, Netfilter allows a module to register callbacks at certain places in the IP stack. Based on this approach, the LISP module would then have been completely separated from IP, and only register callbacks to the Netfilter framework. However, a first study to determine the feasibility of this solution must be done, as LISP processing is more complex than simple packet filtering, or network address translation. Indeed, it requires to jump from one place to another into the stack, as we reviewed in the encapsulation/decapsulation workflow, and the various Data Plane operations. If the integration is possible however, the code would really benefit from greater clarity and modularity.

In this master thesis, we studied LISP mobility with NAT. However, initially, we had considered some other fields of possible study, that were unfortunately unrealizable due to ns-3 limitations. We briefly present these areas for further analysis here.

Outgoing Traffic Engineering

As we already explained, LISP brings new routing capabilities, especially for Traffic Engineering. A LISP site has the possibility to easily control its incoming traffic by manipulating the mappings it distributes. This concerns interdomain *incoming* Traffic Engineering, i.e., how traffic enters the network. However, we didn't find any work on *outgoing* Traffic Engineering, i.e., how traffic leaves the network.

There are two aspects to outgoing Traffic Engineering: the first one is to determine which xTR/RLOC should be used for the outgoing traffic, based on whatever criteria (performance, load balancing, customer or peer link, etc). The second one is to effectively enforce the decision that is made in the LISP site, so that packets leave the network through the chosen xTR/RLOC.
Currently, outgoing Traffic Engineering is achieved with BGP and intradomain TE, where the chosen best routes learned via BGP are distributed in the internal routers of the AS. With LISP however, one cannot simply distribute routes learned via BGP, as the paradigm has completely changed.

This raises several questions: How to best choose the xTR/RLOC? Could an operator easily define several criteria for this choice? For example, one criteria could be the performance from the host to the xTR (i.e., we choose which xTR is used based on the delays from the host to the xTR). Another one, probably more interesting, would be to choose the xTR/RLOC based on the upstream provider that we want to use for the outgoing traffic, depending on the provider performance. To go even further, the choice of the xTR/RLOC could also be based on both the source and the destination at the same time. For example, we could imagine that the choice of the best ISP also depends on the destination. Would it be possible to take the destination into

account when selecting the xTR/RLOC?
For the enforcement aspect of outgoing TE, how to effectively distribute routes in the internal routers? How to implement it? In any case, iBGP must necessarily be modified since external routes (learned via eBGP) mustn't be distributed to the internal routers anymore, because of the two address spaces of LISP. Should we define a new mechanism to distribute routes inside the AS? Or should we modify iBGP to do it?

Source- VS. Destination-based Traffic Engineering

LISP gives network operators a way to enforce Traffic Engineering decisions, with its mechanism of mappings. However, the control plane, i.e., how those mappings are created, is not defined. Right now, mappings are most often created based on policy criteria (load balancing, type of link, preferred RLOC vs. backup RLOC), and not on performance criteria. The only attempt that we found at performance-based TE is IDIPS [19]. The authors present a solution called IDIPS (*ISP-driven path selection*) allowing an ISP to engineer its interdomain traffic. IDIPS can be seen as a black box ranking paths based on network measurements. It is implemented in a server-client fashion, where IDIPS servers can be queried by LISP routers in order to rank paths between two RLOCs.

In such a scenario, we could ask ourselves who has the responsibility of querying the IDIPS servers for RLOC selection? The ITR, who wishes to find the best path? Or the authoritative ETR, who owns the mappings? Both have their advantages and shortcomings.

- Ideally, if the ETR is responsible for querying IDIPS, it should consider all possible sources, to rank path from the sources towards itself. Indeed, mappings cannot be generic for all sources, as traffic coming from China or from Europe won't follow the same paths. The ETR would then use the possibility offered by LISP to differentiate mappings based on the requester of the mappings, to deliver the correct mapping to the correct source.
  In that case, the granularity of the sources must be determined since it is not manageable to query IDIPS for all possible sources in the world. What would be the granularity to have the best trade-off between management overhead and precision of the Traffic Engineering? What would be the cost of storing all those differentiated mappings? The problem here is that the ETR potentially maintains information for sources that will never encapsulate towards itself. Should we instead only select some LISP sites (those that encapsulate the most towards the ETR) instead of considering all sources, in order to reduce the load?
- The second option is to leave this responsibility to the ITR, and have a collaboration between the ETR (who provides the list of RLOCs) and the ITR (who determines which RLOC to use by querying IDIPS). However, querying the IDIPS servers will introduce additional delays before packets can be sent to the destination. What are those delays? Are they reasonable or will they be too high?

Considering everything, what is the best option to perform performance-based Traffic Engineering? The ITR, or the ETR?

Bibliography

[1] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB workshop on routing and addressing," RFC 4984, Internet Engineering Task Force, September 2007.

[2] T. Li, "Recommendation for a routing architecture," rfc 6115, Internet Engineering Task Force, February 2011.

[3] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," rfc 4423, Internet Engineering Task Force, May 2006.

[4] E. Nordmark and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6," rfc 5533, Internet Engineering Task Force, June 2009.

[5] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The locator/ID separation protocol (LISP)," RFC 6830, Internet Engineering Task Force, January 2013.

[6] B. Quoitin, L. Iannone, C. de Launois, and O. Bonaventure, "Evaluating the benefits of the locator/identifier separation," *in Proc. ACM/IEEE International Workshop on Mobility in the Evolving Internet Architecture (MOBIARCH)*, August 2007.

[7] D. Farinacci, V. Fuller, D. Lewis, and D. Meyer, "LISP Mobile Node. draft-ietf-lisp-mn-05," draft (Work in Progress), Internet Engineering Task Force, March 2019.

[8] V. Ermagan, D. Farinacci, D. Lewis, J. Skriver, F. Maino, and C. White, "NAT traversal for LISP. draft-ermagan-lisp-nat-traversal-15," draft, Internet Engineering Task Force, October 2018.

[9] "ns-3." see https://www.nsnam.org/.

[10] G. Hutson, "BGP routing table analysis reports," 2004. see http://bgp.potaroo.net.

[11] B. Donnet, "INFO0031: Network Engineering," Part 3, Chap. 1., Université de Liège, 2019-2020.

[12] J. Abley, K. Lindqvist, E. Davies, B. Black, and V. Gill, "IPv4 Multihoming Practices and Limitations," RFC 4116, Internet Engineering Task Force, July 2005.

[13] M. Menth, D. Klein, and M. Hartmann, "Improvements to lisp mobile node," *in Proc. ITC, 22nd International Teletraffic Congress (ITC 22)*, September 2010.

[14] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis, "Locator/ID Separation Protocol Alternative Logical Topology (LISP+ALT)," RFC 6836, Internet Engineering Task Force, January 2013.

[15] V. Fuller, D. Lewis, V. Ermagan, A. Jain, A. Smirnov, and D. Farinacci, "Locator/ID Separation Protocol Delegated Database Tree (LISP-DDT)," RFC 8111, Internet Engineering Task Force, May 2017.

[16] Y. Li, *Future Internet Services based on LISP Technology*. PhD thesis, Télécom ParisTech, 26 April 2018.

[17] D. Lewis, D. Meyer, and D. Farinacci, "Interworking between Locator/ID Separation Protocol (LISP) and Non-LISP Sites," RFC 6832, Internet Engineering Task Force, January 2013.

[18] F. Baker, and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704, Internet Engineering Task Force, March 2004.

[19] D. Saucez, B. Donnet, L. Iannone, and O. Bonaventure, "Interdomain traffic engineering in a locator/identifier separation context," *in Proc. Internet Network Management Workshop*, October 2008.

[20] C. Perkins, "IP Mobility Support for IP4," RFC 3344, Internet Engineering Task Force, Augustus 2002.

[21] S. Hussein, I. Ismail, and S. Ahmed, "Triangular routing problem in mobile ip,"

[22] L. Agbodjan, "Towards a lisp simulator," Master's thesis, Université de Liège, 2016. see https://bitbucket.org/Lionel_Agbodjan/tfe__towards_a_lisp_simulator.

[23] L. Iannone, D. Saucez, and O. Bonaventure, "OpenLISP Implementation Report. draft-iannone-openlisp-implementation-01," draft, Internet Engineering Task Force, July 2008. see https://tools.ietf.org/html/draft-iannone-openlisp-implementation-01.

[24] V. Sindhuja, and T. Henderson, "GSOC2012NetworkAddressTranslation," 2012. see https://www.nsnam.org/wiki/GSOC2012NetworkAddressTranslation.

[25] F. Coras, D. Saucez, L. Iannone, and B. Donnet, "On the performance of the lisp beta network," *In Proc. IFIP Networking*, June 2014.

[26] "LISP Beta Network." see http://www.lisp4.net.

[27] D. Saucez, and B. Donnet, "On the dynamics of locators in lisp," *In Proc. IFIP Networking*, June 2012.

[28] V. Fuller and D. Farinacci, "Locator/ID Separation Protocol (LISP) Map-Server Interface," RFC 6833, Internet Engineering Task Force, January 2013.

[29] D. Saucez, L. Iannone, and B. Donnet, "A first measurement look at the deployment and evolution of the locator/id separation protocol," *in ACM Computer Communication Review. 43(1), pg. 37-43.*, April 2013.