Master thesis : Implementation and Evaluation of LISP Publish/Subscribe Functionality in NS3

Auteur : Piron, Tom
Promoteur(s) : Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en informatique, à finalité spécialisée en "computer systems security"
Année académique : 2021-2022
URI/URL : https://github.com/tompiron/ns-3-lisp; http://hdl.handle.net/2268.2/16125

Master thesis

Implementation and Evaluation of LISP Publish/Subscribe Functionality in NS3

Academic year 2021-2022

Université de Liège - Faculté des Sciences Appliquées

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

Author :
PIRON Tom

Supervisor :
Donnet Benoit

Abstract

The Locator/Identifier Separation Protocol (LISP) is a protocol designed to solve scalability problems with interdomain routing on the internet. It is based on tunnelling of IP packets through an underlay network, which is also an IP network. A critical component of the protocol is the mapping system allowing to establish such tunnels. In order to improve the time it takes for a tunnel to adapt in case of a change of mapping, an extension to LISP was proposed and is based on a Publish/Subscribe pattern. To our knowledge, no evaluation of this extension has been done yet. In this thesis, we implemented the extension in ns-3, a network simulator, and used it to compare the extension to other existing methods to reduce the handover delay. Our results showed that LISP Publish/Subscribe is indeed more efficient and reliable.

Contents

Chapter 1

Introduction

Nowadays, internet technologies are everywhere and the use of those keeps growing without any slow down in sight. In the midst of this environment, a lot of scalability challenges are posed to enterprises that must manage more and more users and to network operators that need to keep up with a faster and bigger internet every day. Specifically, in the field of interdomain routing, many challenges have appeared. The growth of the internet itself is already a challenge, but in addition many behaviours create more constraints on the network. The growth of the internet routing complexity due to such behaviours motivated the proposition of the LISP protocol. This protocol promotes a separation of the network in two different space according to a Locator/Identifier separation principle to solve those issues. It also allows for improvements upon others of the problematic behaviours, such as traffic engineering and mobility.

In the initial proposition of LISP, few considerations were given to the mobility aspect and the update time of the mapping system, a crucial part of the protocol. In order to improve upon the original protocol, a Publish/Subscribe extension of LISP was proposed. It aims at improving the dynamicity of the mapping system. However, to our knowledge, no evaluation of the actual efficiency of this proposition has been done as yet. We therefore decided to tackle this task.

In order to do that, we made multiple contributions. First, we implemented the extension of LISP Publish/Subscribe in ns-3, a simulator, allowing us and others to evaluate this extension through simulations. Secondly, our implementation is based on an existing implementation by E.Marechal et al. [13] [12] [1]. We improved the existing implementation, primarily regarding the scalability of the implementation. Finally, we compared the Publish/Subscribe extension with other existing solutions thanks to simulations to determine its efficiency.

We were able to conclude that the Publish/Subscribe extension is indeed more efficient and also move the responsibility of handling the propagation of the updates entirely to the mapping system, which is not the case for every alternative.

This thesis is separated in multiple chapters. In Chapter 2, we discuss the LISP protocol, the motivations behind it, as well as its advantages. We also discuss the mapping system for LISP and its possible implementations. In Chapter 3, we go deeper into the proposed Publish/Subscribe extension and how it works. In Chapter 4, we go in detail in the existing implementation of LISP in ns-3, our simulator of choice. We also explain our implementation of the Publish/Subscribe extension and the improvements made to the existing implementation. Finally, in Chapter 5, we present our methodology to compare the extension to other solutions and discuss the results we obtained.

Chapter 2

Locator/ID Separation Protocol

In this chapter, we will present the Locator/ID Separation Protocol, also known as LISP. We will start by discussing the problems that leaded to the proposition of this protocol. Then we will outline the characteristics of LISP and how it solves those issues. We will inspect the protocol first from a data plane point of view and then from a control plane point of view.

2.1 Internet routing scaling problem

In 2006, the Internet community started to worry about a scaling problem that the large backbone operators were seeing arise. The problem was discussed during the Routing and Addressing Workshop of 2006. [14] The number of entries in the routing tables of the backbone, also called Default Free Zone (DFZ), was growing exponentially, and the operators feared they would not be able to keep up. The DFZ is defined as the collection of all the Autonomous Systems (AS) that do not require a default route to route packets. In other words, the DFZ is all the AS that know where to route a packet with any routable address. They form the top of the internet routing hierarchy. At the time, the number of entries in the DFZ FIB was around 200 000 [14]. Today, it has surpassed 900 000 and the size keep growing as we can see on Figure 2.1 [11].

Another related problem is the amount of BGP UPDATE messages that a DFZ router must handle, referred to as the BGP churn. Back in 2006, the number of BGP updates by router was around 500 000 per day with a peak at 1000 per second [14]. Today, those numbers have skyrocketed to 1 500 000 updates per day with a peak at 65 000 per second for IP version 4 [20] and 550 000 updates per day with a peak at 75 000 per second for IP version 6 [21].

Those two problems are responsible for the constant increase in hardware complexity of the routers in the DFZ which cause increase of cost for routing operations. The complexity come from the specialized memory needed to store the growing FIB and the specialized forwarding chips used to ensure fast forwarding operations. Those complex chips also consume a lot of power and produce heat that must be evacuated using yet more energy.

The scaling problem of internet routing is responsible for important costs in terms of energy and money in the internet infrastructure. It also threatens the growth of the internet usage, as technology may not be able to sustain such a growth.

Intuitively, the size of the FIB is bounded by the number of addresses, 2^{32} for IP version 4. However, the size of the DFZ FIB is a lot smaller thanks to the aggregation of addresses. If all the BGP announcements starting with the same prefix come from the

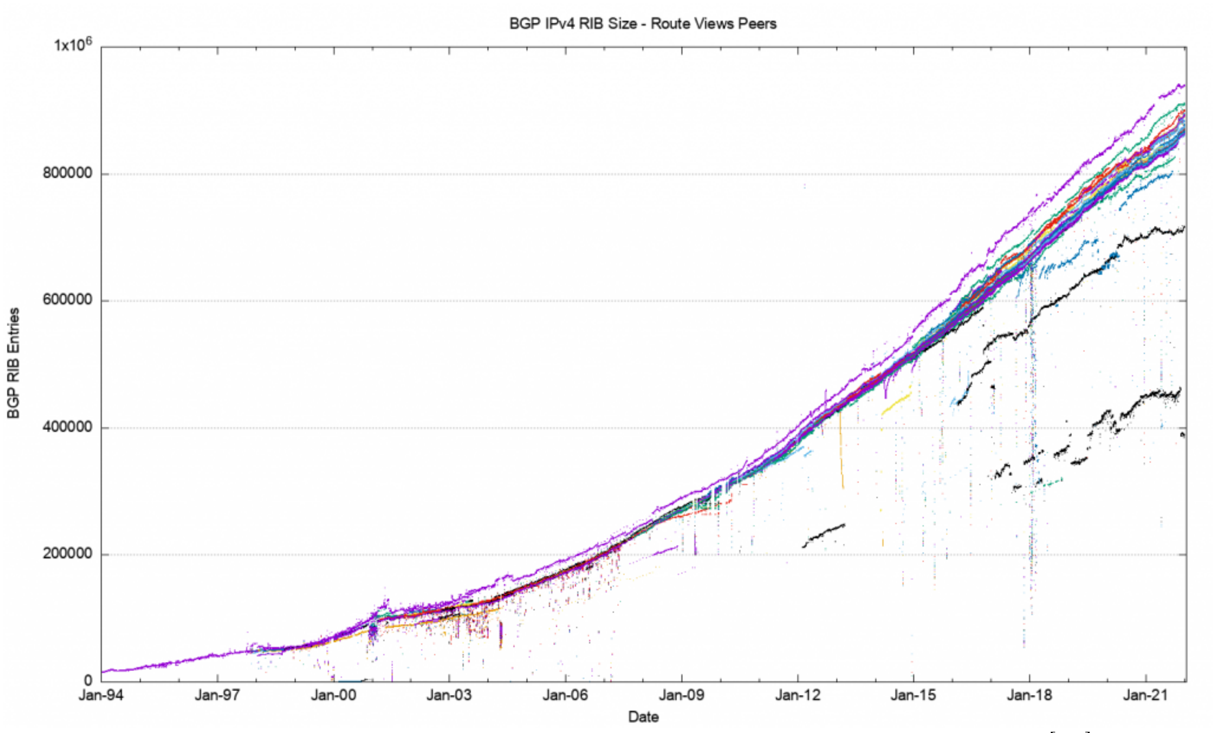


Figure 2.1: IPv4 routing table since 1994 as seen by Route Views peers. [11]

same place, then the routers can aggregate them in a single entry in the FIB. This allows to use a lot less memory.

The main cause of the DFZ FIB growth is therefore the deaggregation of addresses. Meaning that big blocks of addresses cannot be aggregated because the smaller blocks, they are composed of, follow a different route. Which is caused by three big user's behaviours : the use of provider independent (PI) addresses, multihoming and traffic engineering (TE).

2.1.1 Provider Independent Addresses

Usually, IP addresses are provided by the internet provider of an internet user. However, this approach has some problems, especially for large companies. Firstly, you bind yourself to a specific provider and there is a big cost to switch because it would imply to switch the IP addresses of all of your devices. This can be very cumbersome because IP addresses are often used as identification for devices. Therefore, it is used for other features than routing, like access control.

The IP addresses are also commonly hard coded into applications and system's configurations. This allows to bypass DNS in critical applications and make the configuration easier. This means that IP addresses are used directly in multiple places, and changing all of them become impossibly costly for bigger actors.

A solution is therefore used to allow actors more flexibility on their internet providers. It is Provider Independent addresses. The principle is to allow big actors to loan blocks of IP addresses directly to the ICANN. They can then ask their provider to advertise this PI prefix. Switching provider is now easy, the only thing to do is ask another provider to advertise your IP block and forward the traffic to you.

2.1.2 Multihoming

Another cause of the deaggregation of IP prefixes is multihoming, this is the practice of having multiple Internet Service Providers for the same site. This uses the same technique as PI addresses and can be combined with them. The client will simply ask one of its providers to allocate a block of IP addresses for him and ask the other providers to advertise the block of the main provider. Of course, additional routing rules allow controlling through which ISP inbound or outbound traffic is routed.

The main interest of multihoming is the ability to have a backup connection in case of technical difficulties with one ISP. Which allows for better resilience of the services of the client. However, this force deaggregation for all non-primary ISPs and for all if your main ISP is not the one allocating the IP block. Some actors even require a full AS number to operate multihoming, becoming basically their own ISP connected to multiple higher level providers.

2.1.3 Traffic Engineering

The last cause for deaggregation is Traffic Engineering. This is the practice of advertising a route that is not optimal in order to control the path of the traffic. This allows very important techniques for ISPs. The first being load balancing, allowing to spread the traffic over the whole network and ensure an efficient usage of resources. Secondly, it allows shifting traffic to path that are less costly, for example, choosing the cheapest provider. Finally, it allows enforcing certain policies, like avoiding sending some traffic through a certain country.

2.1.4 The underlying cause

Those three causes have some common characteristics. All of them are cases of externalization of cost. This means that each actor have a cost incentive to use them, but by doing so the cost is bared by someone else. In this case, all the cost is supported by the backbone of the internet. But, in the end there is a chain of clients all the way to the actors using the technique, so those cost can be transferred to their user. This is therefore not a problem as long as technologies exists to handle the added complexity. And that is the current problem, how long is the backbone going to be able to increase its capacity to deal with the increasing complexity to a reasonable cost.

All those causes also have a common cause of their own. Those techniques exist because of the double semantic nature of the IP address as a routing locator and an identificator. The IP address initial purpose was to define the location of a device in the network, but through the use of the internet it has also become a way to identify a system which may move in the network. This double role is the root problem, as the edge of the internet gives more importance to the second role and the backbone the first one. This creates a conflict of interest.

End users give more importance to the identificator role because they are interested in the communication between two systems and don't care where they are on the network. The opposite is true for the backbone, they give more importance to the locator role because their role is to route the packets. But the end users have the control over the assignation of IPs, the backbone, therefore, pay the price as the core of the internet is exposed to the dynamicity of the edge.

2.1.5 The solution

The Routing and Addressing Workshop finish by stressing the importance of a solution that would solve the problem without overbearing one specific type of actors in the internet ecosystem and be profitable to implement by all actors involved. It should also be backward compatible and involve a minimum of reconfiguration for the end users.

It's with those objectives in mind that the Locator/ID Separation Protocol (LISP) was designed in an IETF draft [6]. The main idea being to address the underlying issues and create two separate IP spaces, one where IPs have a locator role and another where they have an identificator role. A big part of the problem that arise then is how to map the two address spaces together. We will first ignore this problem and look at the data plane, tasked with routing the packets based on such a mapping, and then we will look at the control plane, tasked with creating and distributing the mappings.

2.1.6 Modern motivations

Even if the growth of the DFZ FIB size is the historical motivation behind LISP, after the proposition of the protocol, many actors came up with additional benefits to such an organization of the internet. Some were already in mind during the original Routing and Addressing Workshop of 2006, like Traffic Engineering, and were researched further afterward [19] [3]. Others, were quickly proposed, like the advantage of using LISP for mobility purposes [4].

2.2 Data Plane

2.2.1 Overview

The main idea behind LISP is to create two networks, with one overlayed over the other. The packets are then tunnelled from the overlay network through the underlay network. LISP is therefore a global dynamic tunnelling protocol. The two networks represent two different address spaces. The first address space is the Endpoint Identifier (EID) space, where IPs have an identifier role, and is the overlay network. The second is the Locator space, where IPs have a locator role, and is the underlay network.

The EID network is composed of many subnetworks connected through a core with border routers. From the perspective of the EID network, in the core, all border routers are connected to each other directly in a full mesh. In reality, the core is the Locator network. A schema of the EID network can be seen on Figure 2.2.

The Locator network doesn't see the EIDs, therefore the locator network look like a network connecting the border routers together. A schema can be found on Figure 2.3. Of course, those are examples and a real network would be way more complex.

The IP packets routed in the EID network are encapsulated to go through the Locator network, giving the illusion of a full mesh between the border routers. The EIDs can therefore be assigned however you want, and this will only affect the complexity of the subnetwork it is part of and the mapping system.

The Locator addresses, called Resource Locator (RLOC), are assigned according to the topology, allowing for efficient use of aggregation in the backbone.

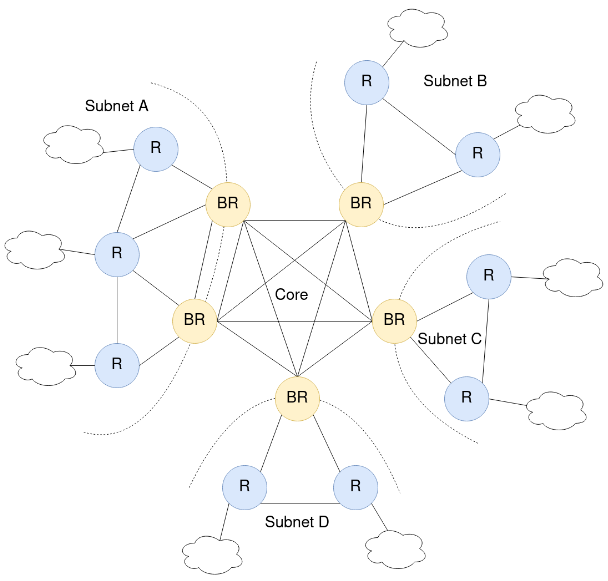


Figure 2.2: Network from the perspective of the EID space

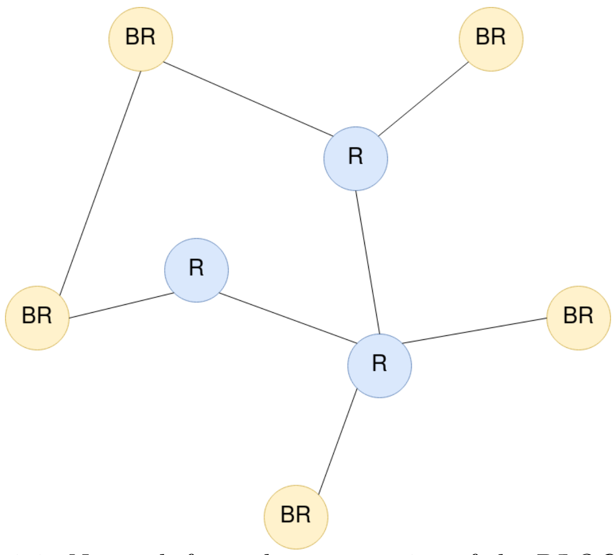


Figure 2.3: Network from the perspective of the RLOC space

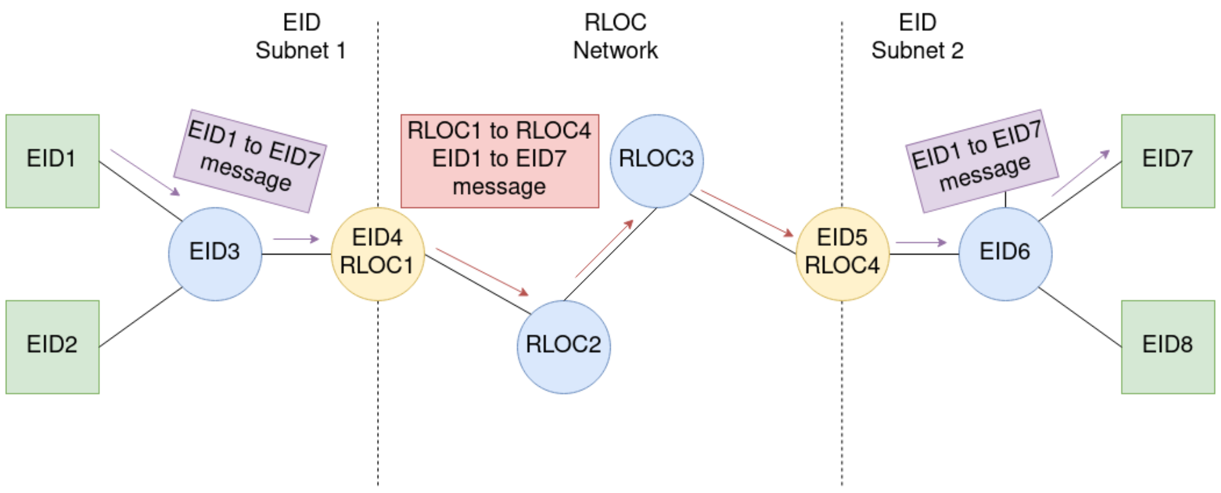


Figure 2.4: Encapsulation example

2.2.2 Encapsulation

In order to send packets through the RLOC network, they need to be encapsulated. The border router called the Ingress Tunnel Router (ITR) will wrap the packets into UDP packets with the destination address being the border router in charge of the destination EID. This router is called the Egress Tunnel Router (ETR). Upon reception of the encapsulated packets, the ETR will scrap the additional header and forward them toward their destination.

On Figure 2.4 an example of encapsulation can be found. In this example we can see that the packet is forwarded normally to the ITR, EID4/RLOC1 in the example, with EID1 as it's source address and EID7 as its destination. When it arrives at the ITR the packet will cross the EID/RLOC boundary. Therefore, it is encapsulated in a packet with source RLOC1, which is a different address IP than EID4, and destination RLOC4, which is in charge of the EID7. When it arrives at the ETR, RLOC4, it is decapsulated and forwarded to EID7.

To the end users, the RLOC network is completely opaque. Which means that the end user don't know the existence of routers RLOC2 and RLOC3, nor that it is forwarded through an underlay network.

The data packets are encapsulated in a UDP packet with an extra LISP header. This one can be seen on Figure 2.5. The UDP header is standard, the only point of interest is the LISP data destination port, which is 4341. The LISP header itself is composed of fields and flags at the beginning that indicate the content of the fields. The N bit indicates that the "Nonce/Map-Version" field contains a nonce. To the opposite, if the V bit is set, then that field contains a source and destination map-version. The L bit indicates that the second field contains Locator Status Bits (LSB). Once again, the I bit indicates the opposite, the field contains an Instance ID. However, the instance ID only use three fourth of the field, leaving 8 bits to serve as LSB, if needed. The E bit is used to request the ETR to include a nonce in packets it would send to the source if acting as an ITR as well. Finally, the R bit is unused and the K bits are used to indicate encryption of the payload packet.

```
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   / |       Source Port = xxxx      |       Dest Port = 4341        |
 UDP +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   \ |           UDP Length          |        UDP Checksum           |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 L   |N|L|E|V|I|R|K|K|            Nonce/Map-Version                  |
 I \ +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 S / |                 Instance ID/Locator-Status-Bits               |
 P   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
Figure 2.5: LISP encapsulation header

2.2.3 Map-Cache

In order to tunnel packets, an ITR needs to know which ETR is responsible for a given EID. Given the complexity of achieving such a goal in a distributed system, this is entirely delegated to the control plane, which will be discussed in Section 2.3. However, we will discuss the mechanism between the data plane and control plane here.

First, each ITR has a map-cache that allow it to avoid fetching the information for each packet. This cache store mapping of EID prefix to a set of RLOC entries with a specific time-to-live. When the ITR receives a packet that needs to be forwarded through the RLOC network, it will check its cache. If it finds a matching entry, the packet is encapsulated and sent. Otherwise, the packet is dropped or buffered, depending on the implementation, and a Map-Request is sent to the control plane. Upon receiving a response, it will store the entry in its cache for future packets.

Each locator in an entry is also assigned a priority and weight, allowing to control the traffic if an EID prefix can be reached through multiple ETR.

2.2.4 Advantages

This solution present many advantages. Firstly, it can be implemented with minimal disruption to the activities of end users and backbone operators. It is also retro compatible and don't require any flag-day. It also greatly facilitates traffic engineering, this being a great incentive to ISP to implement it.

It can also allow for segregation of networks, which means that you can have multiple separate EID networks using the same RLOC network as their core.

2.3 Control Plane

The control plane's objective is to resolve the mappings of EIDs to RLOC and allow the encapsulation to be dynamic by introducing mechanisms to modify them. However, many implementations of such a system are possible, with their own advantages and drawbacks. In order to keep the protocol flexible, LISP creators only defined the interface with the mapping system. This allows modularity between different database design. We will therefore look at the standard interface first and then the proposed implementations of the database.

2.3.1 The interface

The control plane's interface is composed of four entities, two of which have been introduced in Section 2.2, the ITR and ETR. The two others are the Map-Resolver and Map-Server, that will be described in more details here. Those entities will be communicating using control messages, which will have six different types.

All control messages have a similar form. They are all UDP datagram sent in an IP packet. The UDP source and destination ports depends on the type of messages. If the message is a request, the destination port will be 4342 and the source port random. If it is a reply, the source port will be 4342 and the destination either the source port of the request or 4342 dependent on the message type. All the LISP control messages then start with the type of message and some flags dependent on the type.

In the order of the traffic flow, we will first take a look at the Ingress Tunnel Router, and its interaction with the mapping system. The ITR need the mapping system to resolve the mapping between EID prefix and RLOC in order to perform the data plane encapsulation. If the ITR doesn't have any entry in its cache that match a needed EID, it will contact the mapping system. To do that, the ITR will send an Encapsulated Map-Request message to one of its configured Map-Resolver. The role of this Map-Resolver is to handle the querying of the mapping database and forward the Encapsulated Map-Request to the corresponding Map-Server. The ITR will then receive a Map-Reply message of one of two possible type, a positive or a negative reply. If the reply is positive, the ITR can cache the mapping that is contained in it. Otherwise, the negative reply will contain an action and the ITR will react accordingly. The possible actions are :

- Do nothing;
- Set the entry as "Natively-Forward" which result in the packets matching not being encapsulated, i.e. the address is an EID and an RLOC at the same time;
- Retry on the next packet matching;
- Drop all packets matching.

The Egress Tunnel Routers are responsible to update the mapping system with the correct mappings, as they are the ones knowing what they can reach. The ETRs must therefore periodically send Map-Register messages to their authoritative Map-Servers. The role of Map-Servers is to forward Map-Request messages to the authoritative ETRs based on the Map-Register they receive. The ETR then receives Encapsulated Map-Requests from the Map-Servers. It can retrieve the original Map-Request message inside and respond with its mappings directly to the source ITR.

The Map-Server is configured to be responsible for a set of EID prefixes and has authentication information about the ETR that owns the sub-prefixes under his own. It receives authenticated Map-Register messages from ETRs containing a list of RLOCs to forward the Encapsulated Map-Request to. ETRs can also request a proxy service, they provide all the information to the Map-Server, not just which RLOCs knows the information, and the Map-Server can then directly respond to Map-Requests. It is also possible to ask the Map-Server receipts, in this case, it will respond to Map-Registers with a Map-Notify message that is just a copy of the Map-Register's data. The ETR then confirm with a Map-Notify-Ack message that has the same content as the Map-Notify.

The Map-Server is also the last intermediate in the chain to forward Encapsulated Map-Requests. When it receives an Encapsulated Map-Request, the Map-Server will check its local database. If there is no entry for the requested EID, it will send a Negative Map-Reply. Otherwise, it will either forward the Encapsulated Map-Request to the corresponding ETR or send a Map-Reply with the mapping directly if the proxy service was requested for that entry.

The Map-Resolver is the first intermediate in the chain to forward Encapsulated Map-Requests. It may be able to respond directly to the Map-Request, if it is also a Map-Server, or if it knows the address is non LISP capable. Otherwise, it will forward it through the mapping system. This unclear definition of the inner workings of what we call the mapping system is what allow the modularity of the protocol. We don't need to know how it work and can choose depending on our needs. Of course for the goal of improving the internet, everyone needs to agree on an implementation, we will discuss the proposition of mapping system implement in Section 2.3.2.

2.3.2 Implementations

There has been a substantial amount of propositions for the mapping system implementation. They have been discussed in length in the literature [10]. Those are varied in their form and capabilities. Some are based on a full knowledge database, a single system, distributed or not, handle all the mappings, and other on partial knowledge, where the mapping system is composed of nodes which are responsible for a part of the EID space. However, we are only going to discuss two of those propositions, the most widely accepted ones, LISP-DDT and LISP-ALT.

Both LISP-ALT and LISP-DDT are partial knowledge systems because of the advantages of this design. First, the system can be more dynamic because when a change in mapping occur, only the nodes responsible for the prefix need to be updated. In a full knowledge system, the system is under a lot of stress when the network is very dynamic, as exampled by the routing scaling problem of the internet that LISP tries to solve. BGP was not designed to be a full knowledge system, thanks to aggregation, but has explained in Section 2.1 aggregation is not working as intended in today's internet. Any full knowledge system and BGP therefore share the same update frequency problem. The dynamical nature of a partial knowledge system also helps with mobility. The second big advantage is that it give the operation management to local operators that are in control of their EID-prefix and not to a central authority that need to be created and managed.

LISP-DDT or LISP Delegated Database Tree is currently the preferred implementation and is based on the DNS concepts [8]. Many implementation propositions for the LISP control plane share this idea, including some that directly use the DNS database as a mapping system [10]. LISP-DDT is a hierarchical mapping system and is composed of nodes, called DDT nodes, that are hierarchically arranged in a tree based on the EID-prefixes they are responsible for. The root DDT nodes are responsible for the whole EID space, but only store records for DDT nodes directly down the line. In turn, those secondary DDT nodes may only store redirection to DDT nodes down the line or act as a Map-Server and be able to forward the Map-Request to the corresponding ETR. As an example, if you query a root DDT node for 10.1.2.3/32, it may respond with (10.0.0.0/8, 1.0.0.1), meaning that the prefix 10.0.0.0/8 of the EID space is managed by the DDT node at RLOC 1.0.0.1. This new node when sent the same query may respond (10.1.0.0/16, 1.0.0.2). The map resolver can therefore iteratively contact DDT nodes until the leaf DDT node that has the requested information. Such an example can be seen on Figure 2.6.

LISP-ALT or LISP Alternative Logical Topology is another popular possible imple-

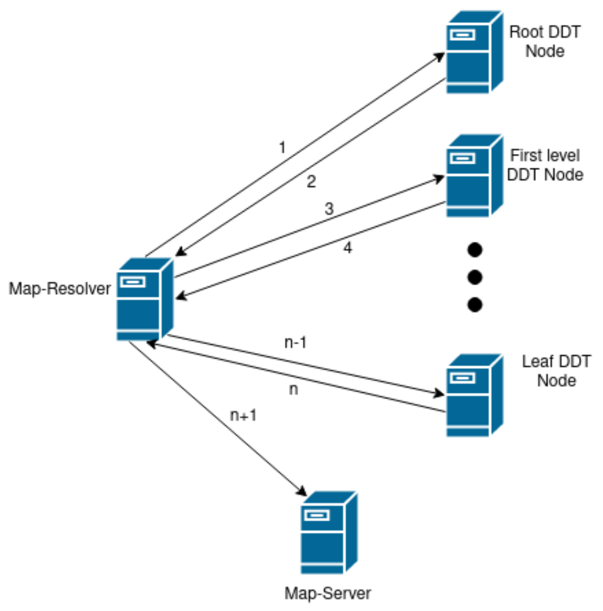


Figure 2.6: Example of Map-Server identification with LISP-DDT.

mentation. It is based on a semi-hierarchical overlay network of the RLOC network [7]. The LISP-ALT system is composed of ALT-Routers, those are RLOC nodes that are connected with each other to form an overlay network. Each ALT-Router has a BGP connection with its neighbours, and the routers advertise the EID prefix for which they know the Map-Server using BGP. This creates a network capable of routing packets to the Map-Server of the corresponding destination EID. Map-Resolvers are then configured with one of the ALT-Router as an entry point to the network and send its encapsulated Map-Request to it. The Map-Request is then forwarded through the overlay network, from ALT-Router to ALT-Router, until it reach the Map-Server of the corresponding EID that forwards it to the ETR for it to respond. The advantage of this proposition is that it allows forwarding data packets through the mapping system while waiting for a response, even if it is not recommended due to the scalability issues it causes. It also requires less configuration of the mapping system, as it uses only existing technologies.

2.3.3 Exploitation of the mapping system

It is interesting to note that, as discussed by M. Gabriel [9], the mapping system can be used as an amplification vector for DDoS attacks. The method of exploitation is reminiscent of DDoS attacks exploiting the DNS system. In both case, the user, for DNS, or the ITR, for LISP, send a small request that can result in a large reply. This allows the forging of requests with a victim address as source to generate a lot of traffic toward that address.

Chapter 3

Publish-Subscribe

In this thesis, we will discuss the implementation and evaluation of an extension to LISP called Publish-Subscribe [17]. The goal of this extension is to allow the mapping system to notify ITRs from changes, to enable faster mobility events and minimal amount of packet drop during such events and even the ability to conserve connections, such as TCP, through the events.

The extension therefore propose two procedures, a subscription and a publish procedure. The first one allows an ITR to express interest in updates on the changes of a specific EID prefix, and the second allows the Map-Server to notify the ITRs that requested it on a mapping change. It's important to note that this extension only work for ETR that request the proxy service of their Map-Server, by setting the P bit in the Map-Register message.

The subscription procedure can be seen on Figure 3.1, and work as follows. First, the ITR will send a Map-Request to its Map-Resolver to retrieve the mapping of the EID prefix to subscribe to. However, it will modify it slightly. The extension introduce two new flags to the Map-Request message, one in the header flags part and the other in the record flags part, as can be seen on Figure 3.3. The ITR will set the I bit in the header flags, this indicates that a xTR ID and Site ID are present at the very end of the message. Those two IDs are also introduced by the extension and are unique IDs assigned to xTRs and LISP subnetworks, LISP Sites, respectively. Those are used to identify a subscriber. The ITR also set the N bit for each record it wishes to subscribe to. This Map-Request message is then forwarded through the mapping system to the corresponding Map-Server. Then, the Map-Server create a subscription state to remember the request, it contains the nonce, the ITR RLOC address, site and xTR ID, and is associated to the prefix requested. The Map-Server then respond with a Map-Notify containing the nonce of the Map-Request to confirm the subscription and respond to the mapping request. Finally, the ITR respond to the Map-Notify with a Map-Notify-Ack. If the subscription fails, the Map-Server simply respond with a Map-Reply.

The publish procedure can be seen on Figure 3.2. This procedure begins by an update to the mappings in the Map-Server, that could be due to a modification with a Map-Register or a time-to-live that expire, removing the entry from the Map-Server database. The Map-Server will fetch the subscription state associated with the updated prefix, increase all nonces by one, and send a Map-Notify to all the subscribers. The ITR will check if the nonce of this Map-Notify is greater than the previous one he received and, if it is, update its cache and send a Map-Notify-Ack. If it isn't the Map-Notify is simply dropped as it is either a replay attack or an out-of-order message.

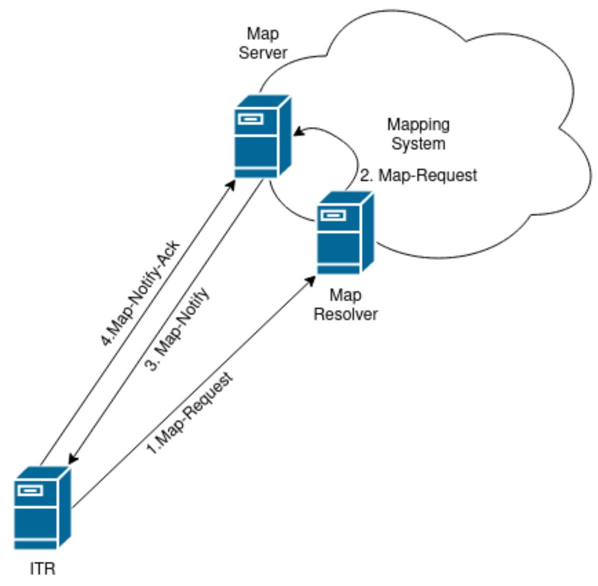


Figure 3.1: Schema of the subscription procedure.

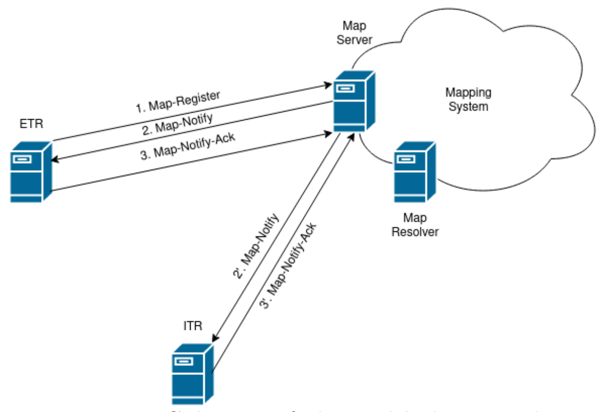


Figure 3.2: Schema of the publish procedure.

If the Map-Server doesn't receive the last Map-Notify-Ack in both procedure, it will resend the Map-Notify to a different ITR associated with the same Site ID. If an entry is removed, the Map-Server send a Map-Notify with a Time-to-Live of 0, essentially asking the ITR to erase the cache entry.

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |Type=1 |A|M|P|S|p|s|R|I|  Rsvd   |L|D|   IRC   | Record Count  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                         Nonce . . .                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                         . . . Nonce                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |         Source-EID-AFI        |   Source EID Address  ...     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |         ITR-RLOC-AFI 1        |    ITR-RLOC Address 1  ...    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                              ...                              |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |         ITR-RLOC-AFI n        |    ITR-RLOC Address n  ...    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  / |N|   Reserved    | EID mask-len  |        EID-Prefix-AFI         |
Rec +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  \ |                       EID-Prefix  ...                         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                   Map-Reply Record  ...                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +                            xTR-ID                             +
    |                                                               |
    +                                                               +
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                            Site-ID                            +
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.3: Format of the Map-Request with the Publish/Subscribe extension. The modifications are highlighted in red.

Chapter 4

The implementation

In this chapter, we will go over the LISP implementation in ns-3 and then will present our implementation of the Publish/Subscribe extension to LISP. The source code for this can be found at `https://github.com/tompiron/ns-3-lisp`.

4.1 ns-3

In order to perform our simulations of the Publish-Subscribe extension of LISP, we decided to use ns-3, as prior work had been done with LISP on this simulator and the simulator itself has many benefits [15].

Ns-3 is a discrete event network simulator. Which means that the network is simulated through events that are assigned an abstract time to be executed. The simulator then always runs the event in the queue with the smallest time assigned in a single thread. The event can schedule other events in the future. This means that the simulation time is independent of the computation time. The simulator may simulate multiple thousands of packets per simulation's second, but actually take many hours of computation. In this way, the results of the simulation are independent of the hardware that runs it, as the execution time is not taken into account. If the simulation needs to process a lot of packets at the same time, it will take more time to execute, but the simulation results will be identical.

The simulator is programmed in C++ and make heavy use of callbacks to handle its event based architecture. It is organized in modules, the most important are :

- *Core* : Classes related to the simulator and base abstractions like Object and Ptr.
- *Network* : Classes related to nodes, packets and sockets.
- *Internet* : Classes related to the IP, TCP and UDP protocols.

Each module is separated in two parts, a low level part called *model* and a high level called *helper*. The models contain the low level API of each module and is constituted, as its name imply, mostly of classes that correspond to objects that will be kept alive during simulation and which methods are going to correspond to simulator events. An example would be a network device class that has a *receive_packet* method, that will be called by another instance of the same class to communicate between each other. The helpers contain classes used to create instances of the models in a more abstract way and not requiring the developers to know and allocate everything by hand.

Ns-3 also provide a strong system with its *core* module to help programmers and computer scientists write code more easily and assure an easy and complete control over the simulation parameters. This come through four main concepts : Smart pointers, an Object abstraction, attributes and finally random variables.

Smart pointers are a very common pattern used in languages like C or C++ in order to remove some burden of the memory allocation scheme of those languages. In ns-3, this is done with the Ptr class that is the smart pointer itself and can be used in a very similar way to traditional pointers with the extra features of allocating memory on creation, having a reference counter and freeing the associated memory space when this last one drops to zero. They are created with the Create function, that takes a class as template argument and the constructor's arguments of this class as arguments. The class given to the Create function must be a child of the SimpleRefCount class. This abstraction allow programmers to mostly use smart pointer without paying attention to memory leaks. However, they are still required to think about circular references that will not allow the reference counter to drop to zero and therefore create a memory leak.

The object abstraction of ns-3 is based on aggregation. That means that member variables of an object that are themselves objects are aggregated together in a single contiguous memory space, with a table referencing the different parts of the global object for easy access. An example is the Ipv4 class that represent the IPv4 protocol and define how IP packets are handled by a node. The Ipv4 and Node classes are both child of the Object class of ns-3 and are aggregated together when we want to create an IP capable node. This is done in order to provide for maximum flexibility and configuration of the simulation, as it is easy to create a child class of Ipv4 and aggregate this one to one node in place of the standard Ipv4 in order to modify its behaviour without having to modify the class in the *internet* module directly and affect all node at the same time. This also allow some part of an object to have different behaviours depending on the presence of another part without forcing the node class to be aware of all possible parts it may have. For example, an application object could detect the presence of either an Ipv4 or Ipv6 object in the node it is associated with and use the correct address format. The Object class is a child class of the SimpleRefCount allowing it to benefit from the smart pointer mechanism. However, it is not created with the Create function, but with the CreateObject function, as the object class needs to be in control of its memory for aggregation purposes.

The attributes abstraction is strongly linked with objects. The Object class also come with an attribute system, each class has attributes that modify the behaviour of its instances. This can for example be the type of congestion management of the TCP protocol. It's an interesting feature because of the way they are set. Attributes have a default value and can be set either globally, by group or individually, allowing for a broad configuration or a precise effect.

Finally, the random variables are classes defined by ns-3 that generate random numbers based on a global seed and guaranties repeatable results. Many exists already, and new ones can be created if need be. They have a great symbiotic relationship with the attribute system and allow controlling precisely any random event. Random variables are child classes of the SimpleRefCount once again.

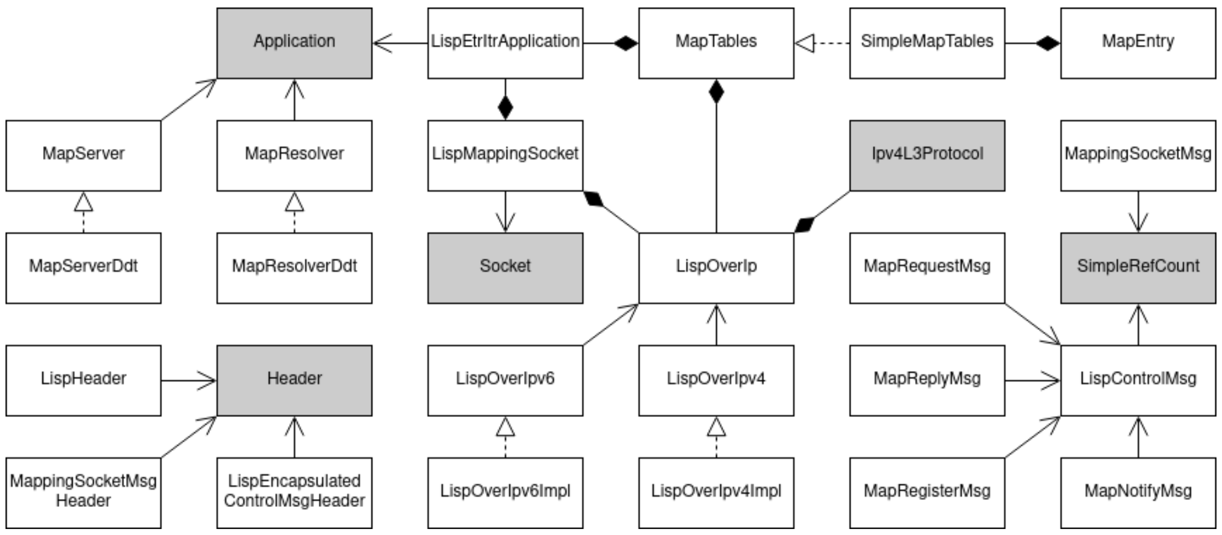


Figure 4.1: UML diagram of the LISP implementation in ns-3.

4.2 Lisp implementation

We have chosen to use the existing LISP implementation proposed by L. Agbodjian [1], Y. Li [12] and E. Marechal [13]. L. Agbodjan first introduced this LISP implementation in ns-3 with basic data plane and control plane capabilities in ns-3.24. Y. Li then based its own version of LISP in ns-3 on the work of Agbodjan and improved it significantly, mainly in the control plane. Li also added mobility capabilities to the implementation as defined in LISP-MN [4]. Finally, E. Marechal improved it further by adding LISP+NAT [2].

The implementation is organized as shown on the UML diagram in Figure 4.1. In the diagram, grayed out objects are ns-3 original objects and white ones are from the LISP implementation. Simple arrows represent parent-child relations, from child to parent. Dotted arrows represent the implementation of an abstract class, and the full diamond arrows a composition relation.

The implementation starts with the Ipv4L3Protocol class, this is a ns-3 class part of the internet module and which responsibility is to implement the processing of IP packets. As a design choice, our predecessor decided to implement LISP within the internet module. This is because of the bilateral dependency between the IP protocol and the LISP protocol. This could have been avoided thanks to the aggregation concept of ns-3, allowing to create a separate module with a LispIpv4L3Protocol that can replace the original Ipv4L3Protocol in the Node object. However, due to the choice having already been done, we kept it as is. But, there is a big drawback, it means we can't use module that depend on the internet module as it would create a circular dependency. The Ipv4L3Protocol class has been modified to encapsulate packets between LISP capable devices. To do it the class is helped by a LispOverIp object that determines if the packets need to be encapsulated, manages the map-cache and contacts the control plane in case of a cache miss. The LispOverIp class is further separated in an Ipv4 and Ipv6 version and each of those have their own implementation, this allows to swap implementations if need be. It's important to remark, however, that the Ipv6 implementation of LISP in ns-3 is largely incomplete.

The map-cache of the LispOverIp classes are themselves an instance of a MapTables class that is implemented in the SimpleMapTables class. This last class maps EndpointId to MapEntry object that contain all the necessary information to perform encapsulation.

The LispOverIp class represent the data plane processing of LISP. It communicates with the control plane represented by the LispEtrItrApplication through a LispMapping-Socket that allows them to send messages to each others. The messages exchanged are MappingSocketMsg with a MappingSocketMsgHeader and allow the data plane to notify the control plane in the event of a cache miss and the control plane to give the mapping information back to the data plane when received.

In this implementation, the control plane is composed of ITR/ETR, Map-resolvers and Map-servers, the operations of which are respectively handled by the LispEtrItrApplication, MapResolver and MapServer classes. The last two are further implemented by the MapServerDdt and MapResolverDdt, which implement a minimal version of LISP DDT [8]. This implementation is minimal, as it can only be composed on one Map-Resolver and one Map-Server, without any DDT nodes in between. The LispEtrItrApplication class, as well as the MapServerDdt use the MapTables class in order to manage their respective mappings. Those three type of control plane entities communicate over IP using LispControlMsg classes : MapRequestMsg, MapReplyMsg, MapRegisterMsg and MapNotifyMsg.

4.2.1 Improvements and bug fixes

Over the course of our implementation of the Publish/Subscribe extension for LISP, we stumbled upon some bugs and needed improvements to the existing implementation. We will go over those.

4.2.1.1 Cyclic dependencies

Firstly, as stated in Section 4.2, the design choice of inserting the LISP implementation in the existing internet module restrict the use of module that require the internet module. This is unfortunately the case for the LispMNHelper that require the DhcpHelper from the internet-apps module. In order to circumvent that, our predecessors decided to not declare the dependency, as it is illegal for the compiler, while still using it. This cause linking errors during compilation, if you attempt to compile a script with the internet module and without the internet-apps module, essentially fusing them together. We decided to disable the LispMNHelper as a result, by commenting its declaration in the wscript of the internet module.

4.2.1.2 Improper message sending in Etr/Itr Application

In the LispEtrItrApplication, a major issue was the way LISP control messages were sent. The sending procedure worked as follows :

- Set the destination address on the socket.
- Schedule a message to be sent on the socket after a defined time.

The problem being that this operation is not atomic, and only scheduling the Send method creates a race condition. If another message is received during the time delay and needs a response, then the destination is reset and both messages will be sent to the second destination address. Essentially, this drop control messages with a higher chance when the frequency of messages is high and therefore when the number of nodes is important.

We solved this simply by bundling those two steps to send a message in a single method, SendTo, and schedule it instead of the Send method. This ensures that the destination address is always set right before sending the message and as ns-3 has a single event thread this resolves any race condition.

4.2.1.3 Many memory leaks

As our plan was to run a single script handling multiple runs of an experiment with many nodes, it was critical to limit the number of memory leaks. The implementation was riddled with small and big ones. We fixed all of them.

The main offenders were circular references between objects. It isn't obvious, but all the aggregated parts of an object share the same reference counter, which mean that storing the reference of a Node in its LispOverIp object will cause a cyclic reference as the LispOverIp object reference itself. The whole Node memory block would be leaked in this case. For those situations, a DoDispose method exists on all Object and is called when the object owner wants to destroy it. For example, when you call `Simulator::Destroy()`. Simply implementing those where necessary fixed the problem.

More anecdotally, the use of the keyword `new` to create buffers was also a major point of memory leaks, even if those were easier to track down with valgrind.

4.2.1.4 Access to uninitialized memory space

Another source of confusion and weird bugs is the read of uninitialized memory space. There are two major sources of those errors. The first being that every time a message must be sent over a socket connection, the message must be serialized into a buffer. In most of the code, a fixed size buffer is created and then populated. There existed some cases where the size calculation of the packet was wrong and ns-3 would read too much data from the buffer on the receiving side. This caused mostly a lot of clutter of the valgrind output, masking more dangerous problems.

The second problem is uninitialized member variables in the message classes. This once again caused a lot of clutter in the logs when those uninitialized memory area are copied to the socket's buffer and then deserialized. However, a much more annoying problem with it is when programmers expect a default value, like the address mask of an EID. This caused the mask to be random, then stored in the database and propagated through the network, creating very confusing outcomes. We believe we managed to track down all of those cases.

We couldn't unfortunately track and resolve all the instances of the read of uninitialized memory space due to the sheer amount of them, and the difficulty to track them with valgrind as some are copied and send through multiple callbacks. Finding the origin is therefore close to impossible.

4.2.1.5 Addition of Time-To-Live in cache

In order to compare the performance of diverse mapping update notification mechanism, we also implemented the time-to-live mechanism of the map-cache of ITRs. This is done very simply by scheduling an event every second for each map-cache in the simulation. The event will then iterate over the cache and reduce the TTLs by one, if it drops to zero the entry is deleted.

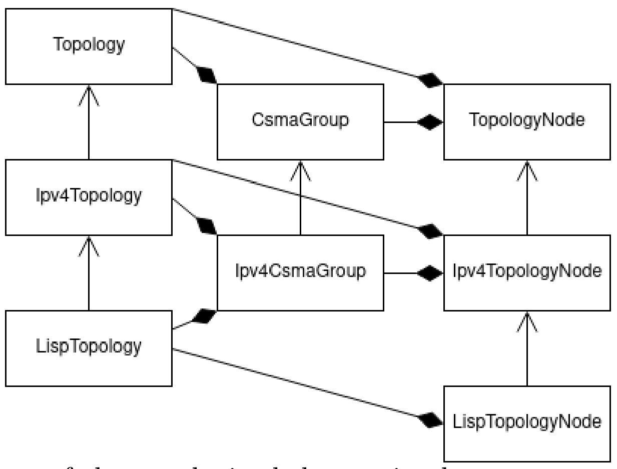


Figure 4.2: UML diagram of the topologies helpers, simple arrows represent parent-child relations, from child to parent and the full diamond arrows a composition relation.

This solution is not the most optimized, but has the benefit of being easy and avoid the creation of additional subtle bugs that could appear with a more complex solution trying to schedule an event only on cache expiration.

4.2.1.6 Creation of Topologies helper classes

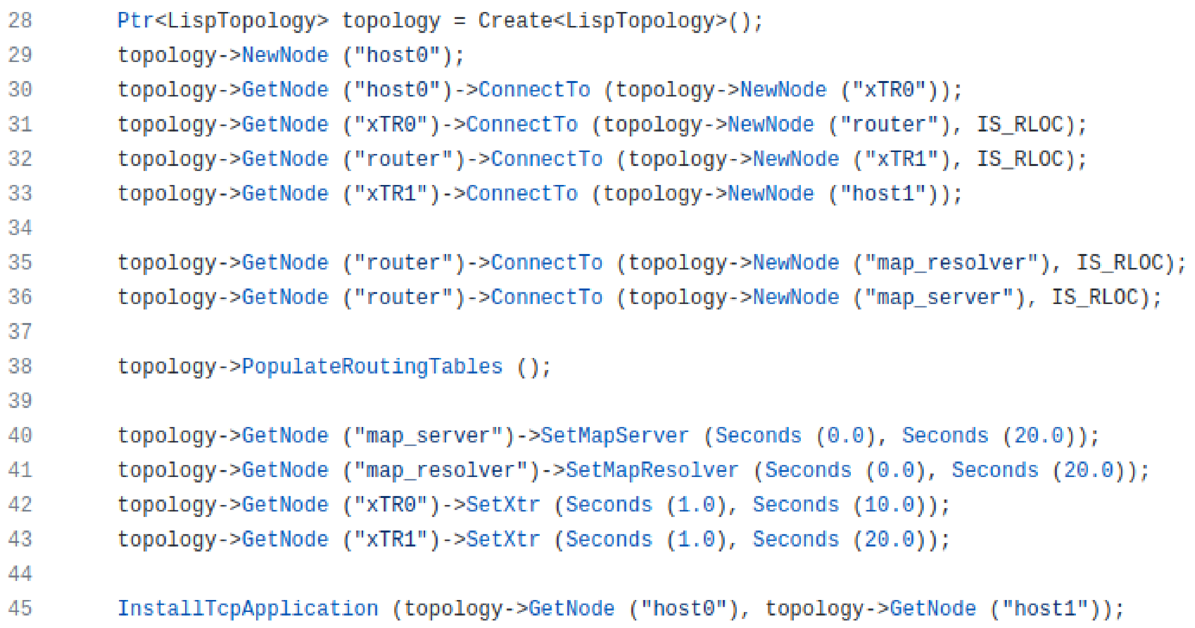
The last improvement to the implementation was the creation of a set of helper classes to create topologies for simulation scenarios. The UML diagram of those classes can be found on Figure 4.2.

The Topology class represent a topology composed of nodes linked together through point-to-point link or a common connection medium. Its responsibilities are to create TopologyNode objects, associate them to a name for easy access using the Topology object directly and create the links between them. The CsmaGroup class allows the creation of a group of Node that are all connected together through a common medium, this can be seen as a bus.

The Ipv4Topology class build upon the Topology class in order to add IP capabilities to the topology. It creates all the IP interfaces necessary, overloading the methods of the Topology class to make it as easy to use as possible. The Ipv4Topology object can also populate the routing tables of the nodes with static routes to make the topology an IP network. Ipv4CsmaGroup adds the capabilities to connect or disconnect nodes dynamically as well as modifying the default IP route of the group, aka the gateway, on the fly. Which is useful to simulate mobility events. It is also possible to assign the IPs of new nodes manually or automatically, allowing for flexibility and ease of use.

Finally, the LispTopology sit on top of the Ipv4Topology. Its role is to handle the configuration of xTR, Map-Server and Map-Resolver. It also manages the map-tables configurations, avoiding the lengthy processes of creating text configuration files that was used before with the LISP implementation, and allows adding entry in the tables of ETRs.

Those classes allow the creation of simulation scripts with a maximum of abstraction and yet a complete control over the configuration of each node. This results in shorter and clearer experiment scripts. An example of such a simple script can be found in the `scratch/simplelisp/simplelisp.cc` file of the code, the important part being copied to Figure 4.3. This script create a simple LISP topology connecting a host to another host with two xTR in between and a router between those last two. It also creates a



```
    Ptr<LispTopology> topology = Create<LispTopology>();
    topology->NewNode ("host0");
    topology->GetNode ("host0")->ConnectTo (topology->NewNode ("xTR0"));
    topology->GetNode ("xTR0")->ConnectTo (topology->NewNode ("router"), IS_RLOC);
    topology->GetNode ("router")->ConnectTo (topology->NewNode ("xTR1"), IS_RLOC);
    topology->GetNode ("xTR1")->ConnectTo (topology->NewNode ("host1"));

    topology->GetNode ("router")->ConnectTo (topology->NewNode ("map_resolver"), IS_RLOC);
    topology->GetNode ("router")->ConnectTo (topology->NewNode ("map_server"), IS_RLOC);

    topology->PopulateRoutingTables ();

    topology->GetNode ("map_server")->SetMapServer (Seconds (0.0), Seconds (20.0));
    topology->GetNode ("map_resolver")->SetMapResolver (Seconds (0.0), Seconds (20.0));
    topology->GetNode ("xTR0")->SetXtr (Seconds (1.0), Seconds (10.0));
    topology->GetNode ("xTR1")->SetXtr (Seconds (1.0), Seconds (20.0));

    InstallTcpApplication (topology->GetNode ("host0"), topology->GetNode ("host1"));
```

Figure 4.3: C++ code using the topology helper classes.

Map-Resolver and Map-Server. As can be seen in the code, the nature, EID or RLOC, of IP addresses used is also indicated in the `ConnectTo` method. This is important for xTR configuration. The four lines creating LISP application layer indicate start time and stop time of the applications. Finally, the user needs to define its application layer, which is only shown in the figure by calling the `InstallTcpApplication` function.

4.3 Publish-Subscribe implementation

Implementing Publish-Subscribe on top of the existing structure required modifications to the operations of the Map-Server as well as the ETR/ITR. Those were done in a spirit of modularity, adding attributes in order to enable fine-grained configuration and with the default behaviour of disabling the Publish/Subscribe implementation.

4.3.1 Modification of the Map-Server

The Map-Server has been modified in two of its processes: the processing of Map-Register messages and of Map-Request. A new class, the SubscriberList, was also created to represent the subscribers' state in the Map-Server. This is a simple class that is a map between EID-prefixes and maps of SiteId and xTRId to RLOC. It allows having a list of subscriber for each EID-prefix and being able to efficiently modify this list.

As can be seen on Figure 4.4 representing the changes to the Map-Register processing. The processing of the P bit was added. If the P bit is set, then a new field in the database entry is set to mark it as a proxy requested one. If no entry exists in the SubscriberList for this prefix, then an empty list of subscriber is added as an entry for that prefix. Otherwise, that means that it is a prefix for which the mapping is updated, a Map-Notify is sent to all subscribers. On the figure, gray nodes represent legacy processing, yellow nodes represent the implementation of the proxy service from the LISP Control-Plane RFC [5] and the green nodes represent the Publish/Subscribe implementation.

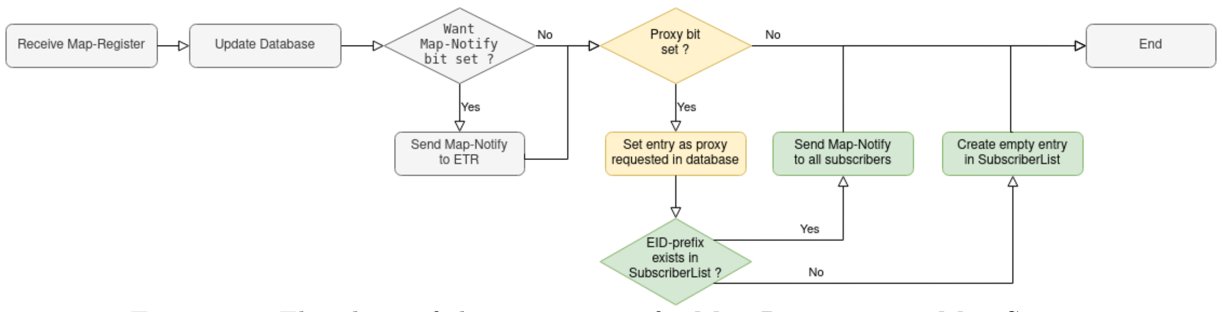

Receive Map-Register
Update Database
Want Map-Notify bit set ?
No
Yes
Send Map-Notify to ETR
Proxy bit set ?
No
Yes
End
Set entry as proxy requested in database
Send Map-Notify to all subscribers
Create empty entry in SubscriberList
EID-prefix exists in SubscriberList ?
Yes
No

Figure 4.4: Flowchart of the processing of a Map-Register on a Map-Server.

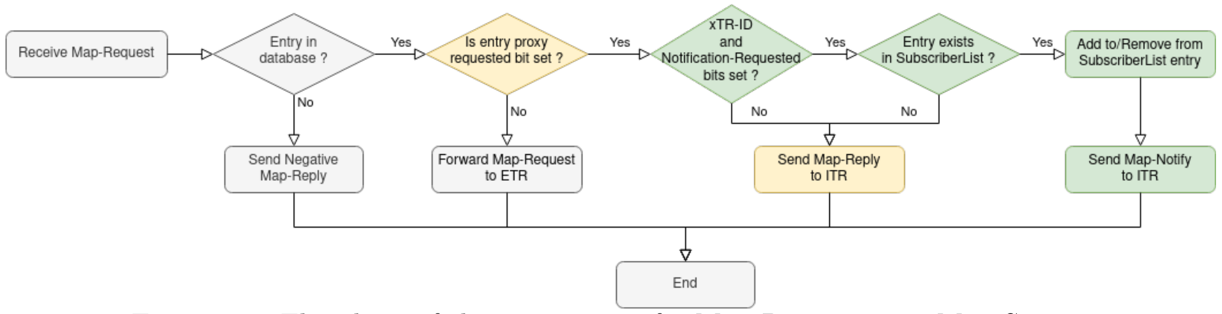

Receive Map-Request
Entry in database ?
Yes
No
Is entry proxy requested bit set ?
Yes
No
xTR-ID and Notification-Requested bits set ?
Yes
No
Entry exists in SubscriberList ?
Yes
No
Add to/Remove from SubscriberList entry
Send Negative Map-Reply
Forward Map-Request to ETR
Send Map-Reply to ITR
Send Map-Notify to ITR
End

Figure 4.5: Flowchart of the processing of a Map-Request on a Map-Server.

On Figure 4.5, we can see the processing of an encapsulated Map-Request on a Map-Server. First, if the Map-Server is not responsible for the requested EID-prefix, a Negative Map-Reply is sent back to the ITR to signify a hole in the LISP mapping system. Otherwise, the Map-Server check if the proxy service has been requested by the ETR for this EID-prefix. If it is not the case, the Map-Server forward the encapsulated Map-Request to the authoritative ETR. If the proxy service has been requested, then the Map-Server will check if the ITR wish to subscribe to the entry. This is done by checking if the xTR-ID and Notification-Requested bits are set. The Map-Server also check if a list of subscriber exists for the EID-prefix, this is just some defensive programming it should never happen unless the implementation is modified. If one of those two conditions aren't met, then the Map-Server reply in the standard proxy way by sending a Map-Reply to the ITR. If both condition are met, the origin ITR is added to the list of subscriber for the requested EID-prefix and a Map-Notify is sent to the ITR to acknowledge the subscription. If the origin ITR is an empty address, the existing subscription for that xTR-ID is removed.

We can see that no modification was done to the processing of the Map-Notify-Ack message, this is because the procedure to repeat sending Map-Notify until receiving a Map-Notify-Ack is ignored. There is also a point in saying it makes little sense in the simulation, as the LISP implementation only support one record per message. The RFC [17] calls to send one Map-Notify to each ITR locator received during subscription until it receives a Map-Notify-Ack or reach the end of the list. As in the simulation, the list is always composed of only one locator, the process would only stop without sending anything. It is therefore not implemented.

4.3.2 Modification of the xTR

The xTR has also been modified in three of its processes: the processing of Map-Notify, the registration procedure and mapping request procedure. Two attributes were also added to serve as configuration. The first is the `EnableProxyMode`, that cause the ETR to request proxy service to its Map-Server. And the second is the `EnableSubscribe`,

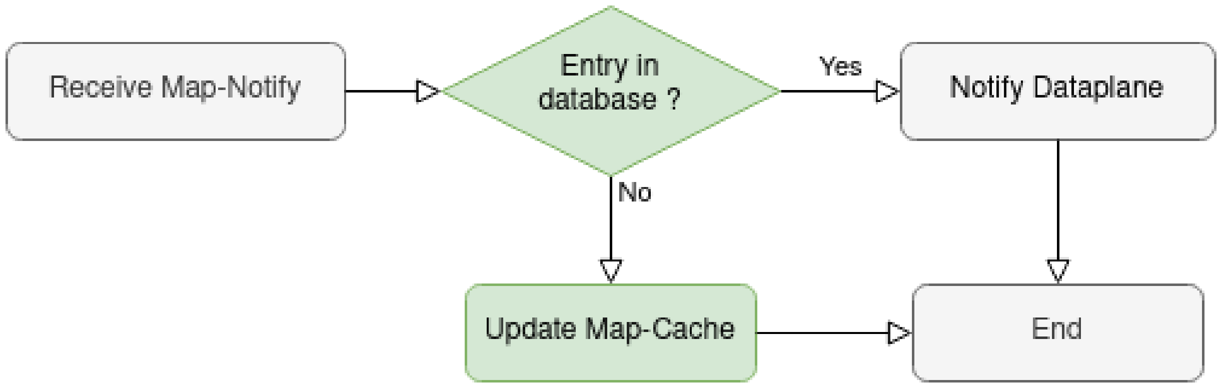


Figure 4.6: Flowchart of the processing of a Map-Notify on an xTR.

which cause the ITR to request notifications from the mapping system as defined by the Publish/Subscribe procedure.

The modifications of the registration and mapping request procedures are minimal. During registration, the Map-Register is modified so the Proxy Map-Reply bit reflect the `EnableProxyMode` configuration. During the mapping request procedure, if the `EnableSubscribe` is true, the xTR-ID and Notification-Requested bits are set and a SiteID and xTR-ID are added at the end of the Map-Request sent to the Map-Resolver. The SiteID is set to the node ID of the xTR and the xTR-ID is set to 0. The node ID is unique for each node in the simulation, this allows to make sure that the requirement on SiteID and xTR-ID allocation are met. Which are that the pair SiteID and xTR-ID should be unique for each xTR.

The processing of Map-Notify messages has been slightly modified. This can be seen on Figure 4.6. Upon reception of a Map-Notify, the xTR will check if the EID-prefix is part of its database, aka its configuration not its cache. If it is, then that means it's the confirmation of a registration. This information is forwarded to the data plane to inform it that it can start to encapsulate packets, as its addresses are now resolvable through the mapping system. If the EID-prefix is not local, then the Map-Notify is a confirmation of subscription and should be processed like a Map-Reply, the Map-Cache is therefore updated.

There are two point of interest in this last procedure. First, the implementation should send a Map-Notify-Ack to the Map-Server, but this is not done. This is because this message has no use as it will be ignored anyway on the Map-Server side, and sending it or not doesn't change the time needed to perform a handover. This would therefore only burden the simulation for nothing. This however change the way the exchanges look slightly, as a message is missing at the end.

The second part is that no check is done on the EID-prefix if it isn't local. The RFC for LISP Publish/Subscribe calls for maintaining a list of nonce associated with EID-prefix that were subscribed to. This allows to avoid replay attack and simple malicious Map-Notify for unrequested EID-prefix. However, we didn't implement it as it was not of interest for our work.

Chapter 5

Evaluation

In this chapter, we will evaluate the efficiency of the proposed Publish/Subscribe extension to the LISP protocol [17].

5.1 Methodology

Our aim in this work is to compare the effectiveness of the different procedures to notify an ITR of a change of mapping at the ETR it is currently communicating with. We identified four main methods.

- Doing nothing, this relies on the fact that entries in the map-cache of ITR have a time-to-live and will eventually be dropped.
- Using the SMR procedure, this relies on sending Solicited Map-Request messages to the ITR that are in communication with an ETR when the mappings for that ETR change.
- Using the SMR procedure and the proxy service of Map-Servers.
- Using the Publish/Subscribe extension of LISP.

In order to conduct experimentation with those methods, we created a simple class structure for our scripts, allowing to avoid code repetition and assure that all the experiments are done in the same context. This structure can be found on Figure 5.1. This structure allows the main experiment to be situated in the LispExperiment class and to run multiple experiments in a single script by destroying the LispExperiment object. All Experiment classes also take as their constructor arguments the number of receivers and senders that are in the simulation to see the effect of an increase in the traffic on the result of the protocol. The role of the runner class is to execute multiple Experiments. And each Experiment classes correspond to a scenario.

The LispExperiment class perform those actions in order to build the simulation.

- Configure timing models.
- Build topology.
- Install load applications.
- Schedule the mobility event.

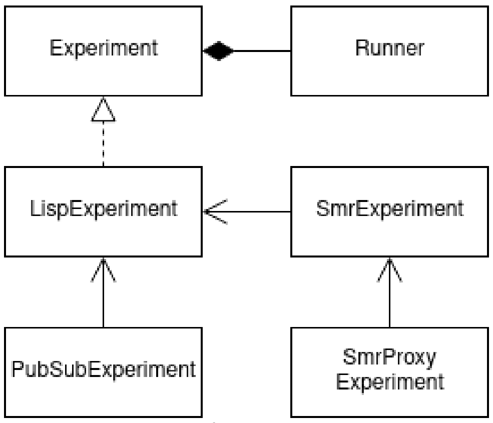


Figure 5.1: UML diagram of the simulation script's structure.

5.1.1 Timing models

As ns-3 is an event based simulator, only the protocols are simulated without consideration for operating systems or other limitations of the actual hardware or software, unless specifically implemented in ns-3. This means that all limitations or point of interests to be studied need to be simulated through models. This is also the case for timing, as the simulation run on an internal discrete clock, that only move when an event is scheduled. This is great because simulation results are independent of the hardware that run the simulation. In our case, we will only need to simulate timings with some models.

In order to determine what timing we need, we performed a theoretical analysis for each scenario. On all the following figures, the full arrows represent messages that are important to the global handover delay. The dotted arrows represent message that are part of the protocol but don't influence the handover delay, their legend is also between parentheses. Finally, the numbers indicate the order of messages, with apostrophes indicating the spilt of the message exchange in two different streams.

The analysis of the scenario without update mechanism can be found on Figure 5.2. In this scenario, as in every other, the mobility event is triggered at the ETR, which kickstart the process by sending a Map-Register to its Map-Server (1) to update the corresponding entry. This is always followed by the Map-Server processing the Map-Register (2) and the confirmation process being started, however this last one doesn't influence the handover delay. The confirmation process is composed of sending a Map-Notify back to the ETR (3), the ETR processing the Map-Notify (4) and finally the ETR acknowledge with a Map-Notify-Ack (5). As soon as the Map-Register has been processed (2), we need to wait for the time-to-live of the Map-Cache entry of the ITR to expire (3'). When this is the case, the ITR will send a Map-Request to its Map-Resolver (4'), that will forward it via the mapping system to the corresponding Map-Server (5'). The Map-Server then forwards the request to the ETR (6'), that process it (7') and, finally, it responds to the ITR with a Map-Reply (8').

Overall, the sequence of events that define the global handover delay for the scenario without any update mechanism is :

1 . Map-Register from ETR to Map-Server.

2 . Processing of the Map-Register by the Map-Server.

3'. Waiting for the TTL to expire at the ITR.

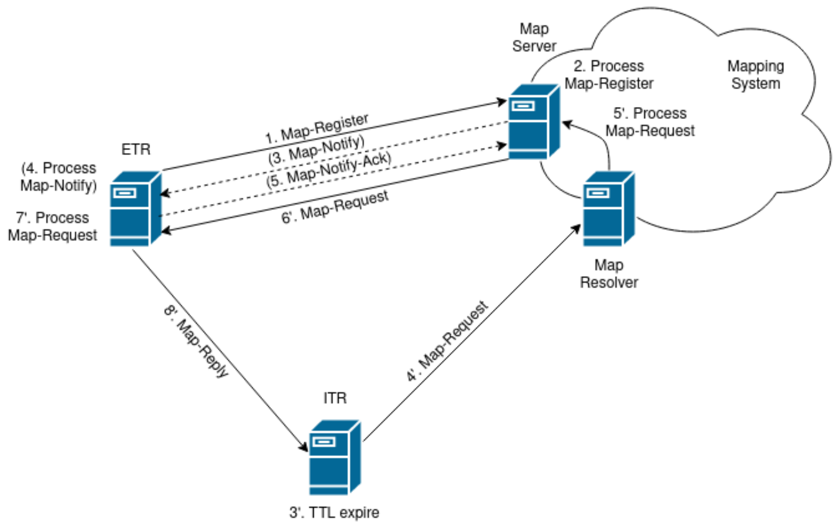


Figure 5.2: Theoretical timing analysis of the scenario without any update mechanism.

4'. Map-Request from ITR to Map-Resolver.

5'. Processing of the Map-Request by the mapping system.

6'. Forwarding of the Map-Request from the Map-Server to the ETR.

7'. Processing of the Map-Request by the ETR.

8'. Map-Reply from the ETR to the ITR.

Then, the scenario where the ETR, acting as an xTR, send a Solicited Map-Request message to the ITR, also acting as xTRs, can be seen on Figure 5.3. This scenario begins in the same way as the last with the registering procedure (1-5). However, as part of this process, after the Map-Notify was processed (4), a Solicited Map-Request is sent to the ITR (5'). The message is not sent directly after the Map-Register to avoid reordering of packets that would lead the ITR to receive the old mapping, which actually often happened with our simulations. The ITR then process it (6') and query the mapping system as usual (7'-11'), leading to the ITR receiving a Map-Reply with the updated mapping.

The sequence of events influencing the handover delay is therefore :

1 . Map-Register from ETR to Map-Server.

2 . Processing of the Map-Register by the Map-Server.

3 . Map-Notify from the Map-Server to the ETR.

4 . Processing of the Map-Notify by the ETR.

5'. SMR from the ETR to the ITR.

6'. Processing of the SMR by the ITR.

7'. Map-Request from ITR to Map-Resolver.

8'. Processing of the Map-Request by the mapping system.

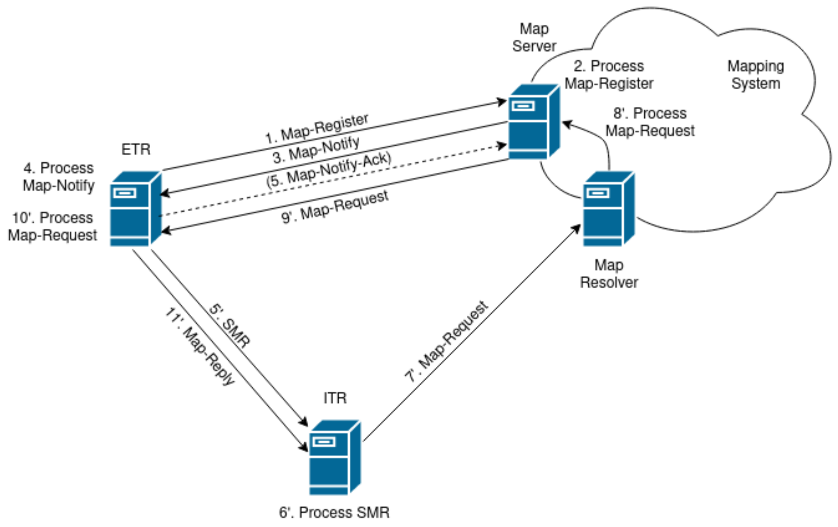


Figure 5.3: Theoretical timing analysis of the scenario with SMR messages.

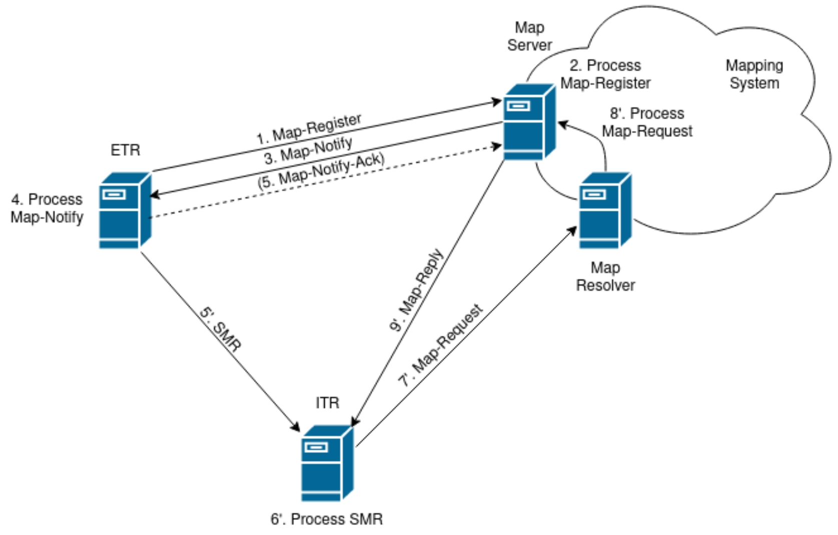


Figure 5.4: Theoretical timing analysis of the scenario with SMR messages and proxy service.

9'. Forwarding of the Map-Request from the Map-Server to the ETR.

10'. Processing of the Map-Request by the ETR.

11'. Map-Reply from the ETR to the ITR.

The analysis for the scenario in which we send SMR and use the proxy service of the Map-Server is very similar to the second one and is represented on Figure 5.4. This scenario is identical to the precedent up to the point of querying the mapping system. It starts in the same way, sending a Map-Request to the Map-Resolver (7') and then the Map-Request is processed by the mapping system (8'). But, instead of forwarding the Map-Request to the ETR, the Map-Server sends the Map-Reply directly to the ITR (9').

The sequence of events is therefore identical to the last one up to (8') and then is followed by only one event instead of three :

9'. Map-Reply from the Map-Server to the ITR.

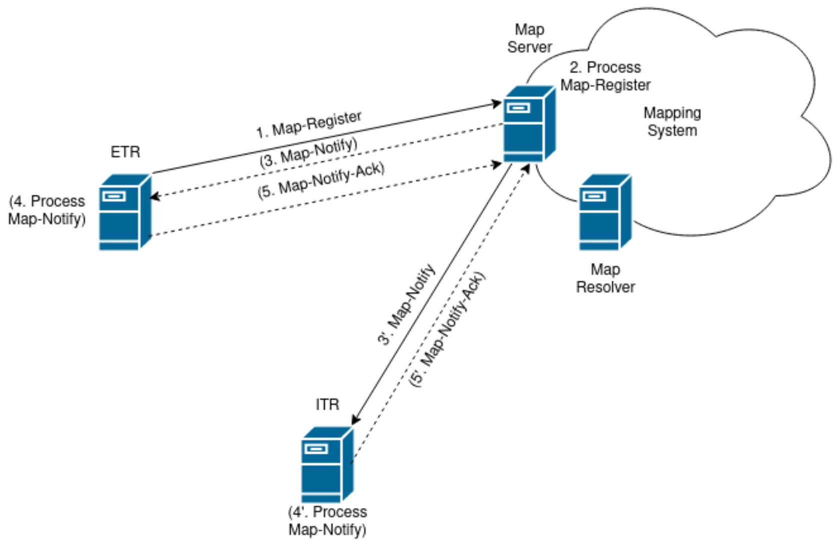


Figure 5.5: Theoretical timing analysis of the scenario with Publish/Subscribe.

Finally, the scenario using Publish/Subscribe can be found on Figure 5.5. This scenario is quite different from the others. As we can see, it starts in the same way as the others, with the registering procedure (1-5). As part of this procedure, however, the Map-Server sends a Map-Notify to the ITR (3') at the same time as the ETR. This Map-Notify is then processed (4') and the ITR respond with a Map-Notify-Ack (5'). Those last two steps don't have an impact on the handover delay. They exist for reliability purposes, as we won't model any packet drop, they will not have an impact.

The sequence of events influencing the handover delay for the Publish/Subscribe scenario is :

1 . Map-Register from ETR to Map-Server.

2 . Processing of the Map-Register by the Map-Server.

3'. Map-Notify from the Map-Server to the ITR.

There are a lot of diverse events that we need to find a time model for. We therefore decided to simplify the picture by concentrating on the most important ones. Firstly, we decided to ignore all processing times, except the processing of Map-Request by the mapping system. This is because those times are way smaller than the transmission delays of messages and therefore account for an insignificant part of the overall handover delay. We also decided to consider ETR and ITR as the same, as they are designed to play both role or be very similar from a network and delay perspective. We will also merge the Map-Resolver's delays with the mapping system processing, as it is difficult to find data for only part of the mapping system. We therefore need 4 models :

- The delay before expiration of TTL at the ITR.
- The delay between an xTR and a Map-Server.
- The delay between two xTR.
- The processing time of the mapping system.

The first one is already taken care of as we implemented the cache expiration process in the ITR application. We decided to use one minute as our default time-to-live. But still, we performed an experiment with three minutes as default TTL to confirm the impact of this variable on our simulations.

For the second one, we unfortunately were not able to find any data to create a model of the delay between an xTR and a Map-Server. As creating such a dataset was way out of the scope of this work, we decided to consider that this delay was equivalent to the delay between two xTR. This is not the best choice and is definitely a weak point in our results, but this still allows for meaningful results, as both delays should be on the same order for a connection between an ITR and the Map-Server. For a connection between an ETR and its Map-Server the delay is probably smaller, as network administrators can optimize the position of a Map-Server relative to the ETRs it serves. This however still allows us to compare the result to each others, as this introduce the same delay to the results in all scenarios.

We therefore need a model for the timings of a packet going from a xTR to another xTR and from an xTR to the mapping system added to the time the mapping system takes to process a request.

In order to create the first model, xTR to xTR, we used data from D. Saucez [18]. We used his dataset containing data from the lisp4.net network, that is unfortunately no longer available. Precisely, we used his `pings` dataset that was constructed by measuring the round-trip time to locators of the network from a vantage point located in Louvain-la-Neuve University campus in Belgium. This allows us to construct a model of global reachability of a RLOC, that can be used for the delay between two xTR as well as xTR and Map-Server. We then divided by two the values in order to approximate the one-way delay. With the help of the EmpiricalRandomVariable class of ns-3, that allows the creation of a random variable that will draw random numbers from a distribution defined through its cumulative distribution function, we were able to create a model that approximate the data of D. Saucez. However, we are not able to input the full CDF, we therefore approximated it. The final model we used can be seen on Figure 5.6 with the original data to compare the difference.

Finally, for the second model, we decided to base our model on the DNS system. This is done because data are more readily available and the LISP mapping system in its LISP-DDT [8] proposal is very similar to DNS. We used the `alexa1m` dataset from OpenINTEL [1], a joint project of the University of Twente, SIDN, SURF and NLnet Labs [16]. We used exclusively the round-trip time values to construct our model. We constructed the model in ns-3 in the same way as the precedent, and it can be seen on Figure 5.7 with the original data CDF as well.

It is important to note that our model doesn't correspond perfectly to what we are trying to model. We want to model the time between the Map-Request being sent and the Map-Server response being **sent**. This model is closer to the time between the Map-Request being sent and the Map-Reply being **received** in the SMR and proxy scenario. But our goal is to be able to compare our scenarios. This only add an equivalent delay to all the scenarios except for Publish/Subscribe. So we decided to remove the xTR to xTR delay for the step (6'), (9') and (9') for the scenarios without update, with SMR and with SMR and proxy respectively.

With those models, we can calculate the theoretical results of our simulations. We calculated the results considering our models as returning constants depicted by xTR-

[1]https://www.openintel.nl/

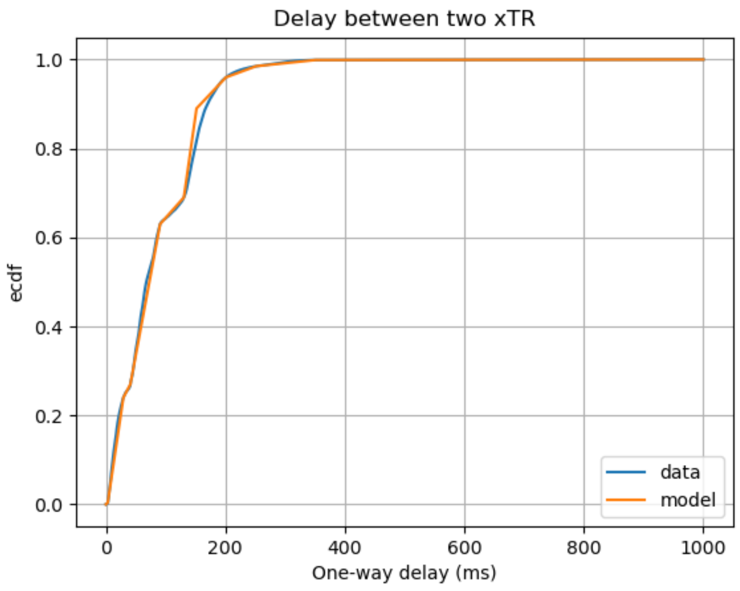


Figure 5.6: Dataset used and model of the xTR one-way delay.

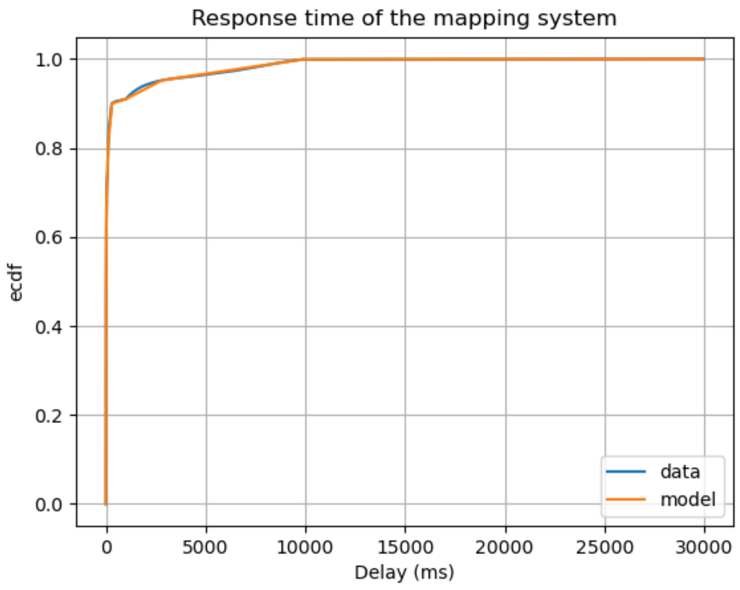


Figure 5.7: Dataset used and model of the mapping system delay.

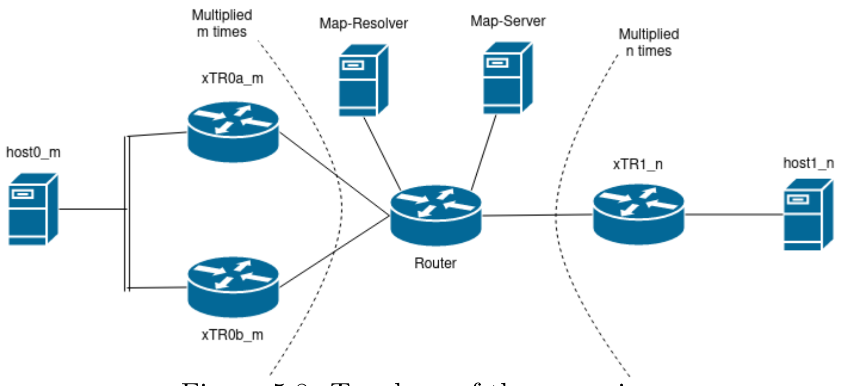


Figure 5.8: Topology of the scenarios.

toxTR for the first model and MappingSystem for the second. By inputting the average of the distributions in those formulas, we should get the average of the results. The results should be :

- **Without update** : Two steps have an xTR to xTR delay (1 and 8'), three steps represent one mapping system delay (4'-6') and step (3') is modelled by the simulation. The total expected delay is

$$2 * \text{xTRtoxTR} + \text{MappingSystem} + \text{uniform}(\text{defaultTTL})$$

  , where "uniform" represents a uniform distribution between zero and the argument. The uniform distribution represents the fact that the mobility event can happen at anytime and therefore the TLL expiration take a random amount of time following a uniform law.

- **With SMR** : Four steps have an xTR to xTR delay (1, 3, 5' and 11') and three steps represent one mapping system delay (7'-9'). The total expected delay is

$$4 * \text{xTRtoxTR} + \text{MappingSystem}$$

- **With SMR and proxy** : Four steps have an xTR to xTR delay (1, 3 and 5') and three steps represent one mapping system delay (7'-9'). The total expected delay is

$$3 * \text{xTRtoxTR} + \text{MappingSystem}$$

- **With Publish/Subscribe** : Two steps have an xTR to xTR delay (1 and 3'). The total expected delay is

$$2 * \text{xTRtoxTR}$$

5.1.2 Topology

The topology is composed of a static part and 2 modules that are duplicated as needed in order to create the `m` receivers and `n` senders requested. The topology can be seen on Figure 5.8.

On the left, we can see the receiver module that will receive the data connection and the module in which the mobility event will occur. It is composed of 3 nodes, `host0_m`, `xTR0a_m` and `xTR0b_m`. Those 3 nodes are connected through a CSMA connection, allowing more flexibility of connection at runtime than a point-to-point link. This means that we can designate a node as the gateway to the global internet for that network, as well as connect or disconnect nodes dynamically. On the right, the sender module can be seen, it is simpler and only composed of two nodes, `host1_n` and `xTR1_n`. The two hosts are the nodes that will be communicating over LISP. The last part of the topology is in the middle and is composed of a router that connect all the module together, as well as to a Map-Resolver and a Map-Server.

5.1.3 Load application

In order to see the handover of a LISP tunnel happen, we need a data connection. We decided to use a UDP connection in order to have more control over the connection and avoid creating too much unnecessary traffic. Each scenario has a main forward connection from senders to receivers. This connection send 2 datagrams with 10 bytes of data each per second.

In the case of the two SMR scenarios, we also need a backward connection. As in the Solicited Map-Request process, the xTR will sends SMR to all the ETR for which it has an entry in its ITR Map-Cache. These streams go from the receivers, `host0_m`, to the senders, `host1_n`, and have the same load as the primary one.

5.1.4 Mobility event

In order to simulate a mobility event, the `xTR0b_m` nodes are connected and set as default route of their receiver network. The `xTR0a_m` are disconnected. Then, the new xTRs send Map-Register to the Map-Servers.

In the two SMR scenarios, we also need to start the SMR sending process. When the Map-Notify is received by the new xTR the old xTR send an SMR to each ETR in its Map-Cache.

The last point of attention is that the time for a mobility event is always 25 seconds before the TTL expires, except for the first scenario, without any notification, in which case the mobility event is randomized in such a way that the TTL may expire at anytime after the mobility event following a uniform distribution between zero and the default time-to-live value. This is done in order to observe the part of the protocol that is of interest in the three last case. Whereas the first case is there as a reference.

5.1.5 Data collection

In order to compare the efficiency of the different scenarios, we decided to compare the time it takes for an ITR to be notified of the modification of a mapping at the ETR. In order to do that, we traced the mobility event as well as mapping update of all senders xTR in custom output files and used the discrete simulator time as a reference.

Those are then processed with python scripts in order to extract the handover time from those raw data.

Scenario		Expected result	Empirical result
No notification	Max	60560 ms	60337 ms
	Mean	30560 ms	31832 ms
SMR		720 ms	724 ms
SMR with proxy		640 ms	643 ms
Pub/Sub		160 ms	162 ms

Figure 5.9: Empirical and expected results for the scenarios with constant timing models.

5.2 Results

5.2.1 Comparison with theoretical calculation

First, in order to assert the correctness of our implementation, we decided to run one hundred simulations with constant models for the timings in order to verify our results. The results can be found on Figure 5.9. First, we can note that only the first scenario without notifications is not constant and is therefore denoted by its maximum delay and mean delay. This is due to the experiment setup that randomize the mobility event for this scenario. The three others have the same result for each and every of the hundred simulations, they are therefore denoted by this single value. We can see that the empirical values are all exactly as expected, safe for a few milliseconds. The cause of this small delay is internal workings of ns-3, when managing the data rate of a link for example, but was considered small enough to be ignored. Furthermore, it seems to be quite stable and therefore doesn't hinder our capacity to compare the results. The results of the first scenario are a bit more off, but this is due to the fact that there are only a hundred simulations.

5.2.2 Scenario comparison

On Figure 5.10, we can see the results of a thousand simulations with one sender and one receiver for each scenario. The first thing to jump at you is of course the gap between all scenarios with notifications and the one without. The scenario without any notification mechanism has a uniform distribution of delay, as expected. We can also see on Figure 5.11, that it is a uniform distribution from zero to the default TTL. One last thing on this scenario, we can see a small tail at the end of the graph, indicating that the distribution is not exactly uniform but is influenced by the mapping resolution and xTR delay.

On Figure 5.12, we focused on the beginning of the graph. We can clearly see the remaining 3 scenarios. As expected, the best performing one is the one with Publish/Subscribe, followed by the SMR with proxy and then the SMR alone. This graph is however quite surprising. The Publish/Subscribe scenario is performing as expected with a mean of 174 ms, exactly as expected as the mean of the xTR to xTR delay model is 86 ms, and the result should be two times this delay. However, the SMR scenarios are being way more efficient than expected. The version with proxy and without have a mean delay of, respectively, 259 ms and 345 ms. Significantly lower than the expected 694 ms and 780 ms of average, as the mean of the mapping system delay model is 436 ms. Those are

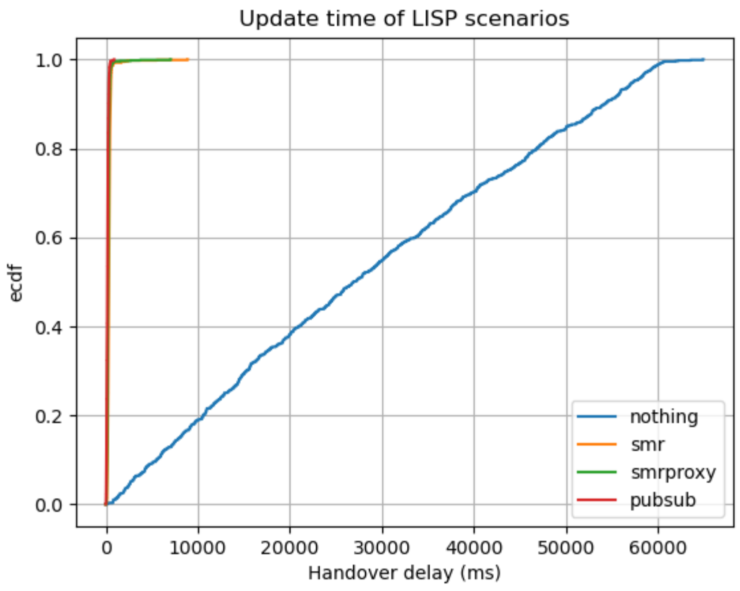


Figure 5.10: Results of 1000 simulations with 1 sender and 1 receiver.

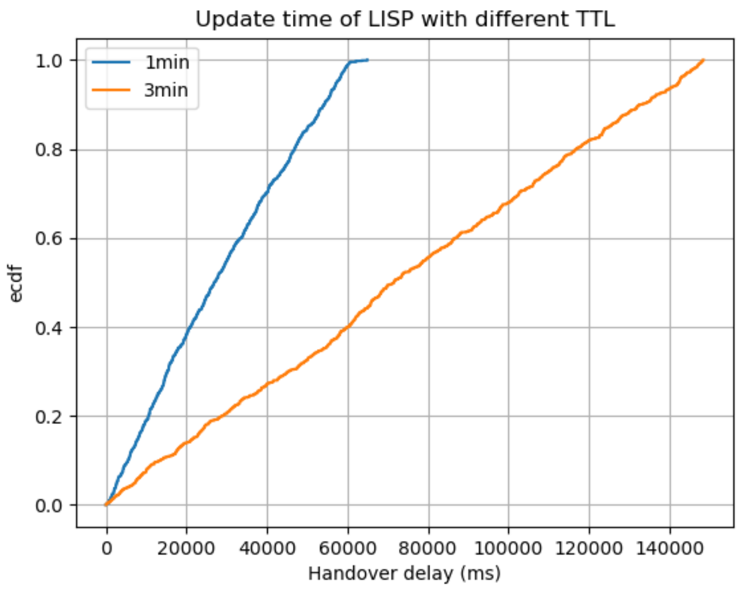


Figure 5.11: Comparison of the handover delay without notification.

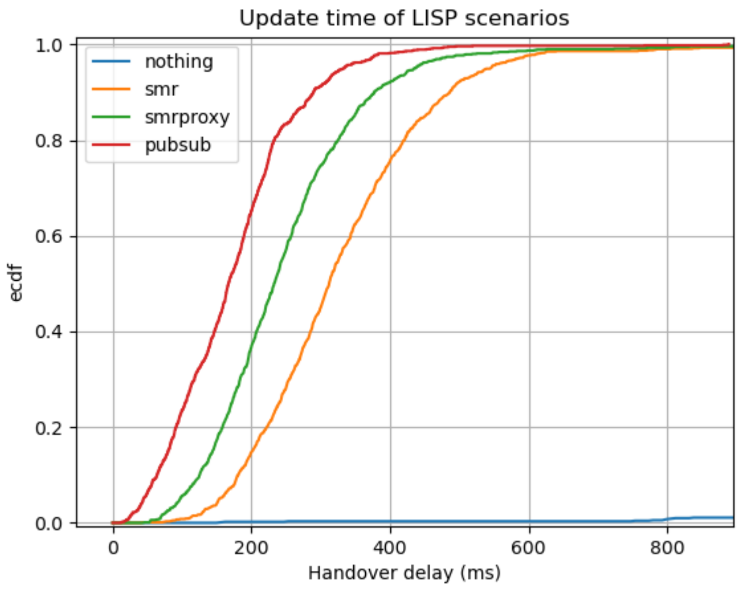


Figure 5.12: Zoom on the results of 1000 simulations with 1 sender and 1 receiver.

both 435 ms under the expected value.

Fortunately, there is a logical explanation to this, and it holds in two part. First, we must remember that when receiving an SMR request, the ITR will remove its corresponding entry in the cache. Meaning that it will not only directly send a Map-Request, but also resend one every time it receives a packet for that destination. This means that even if we were unlucky and got a very long response time for the first Map-Request, on average after 250 ms another will be sent, and then a third 500 ms later. The second element can be seen on Figure 5.13. We can see that the mapping system model starts sharply with more than 80% of delays under 100 ms, even beating the xTR delay model at the beginning. This means that 90% of the times the first Map-Request will have a response in under 250ms, and if not there is a 50% chance that another Map-Request has already been sent. This mechanism heavily pull down the average and explain the unexpectedly good result of the SMR scenarios.

We can however be reassured by the fact that the difference between the scenario with and without a proxy is, as expected, 86 ms. All in all, those graphs shows a weakness of our implementation in the fact that our models are completely random for each packet. An alternative would be to create a model that takes into account the EID address of the request in order to avoid having later requests process faster than earlier requests if their processing is not finished. Effectively, it doesn't make sense as later requests should take less time, specifically because some information has been cached during the previous request. If this previous request is not finished, the information has not been cached.

5.2.3 Influence of the number of nodes

We performed a lot of simulations with different number of nodes. In the table on Figure 5.14, there are the number of simulations we conducted for each setup, with a variating amount of senders and receivers. However, it seems that the number of nodes don't influence the performance of the protocol. All the graphs are similar to the one in Figure 5.10. With averages of the results that stay very close to the value given in Section 5.2.2.

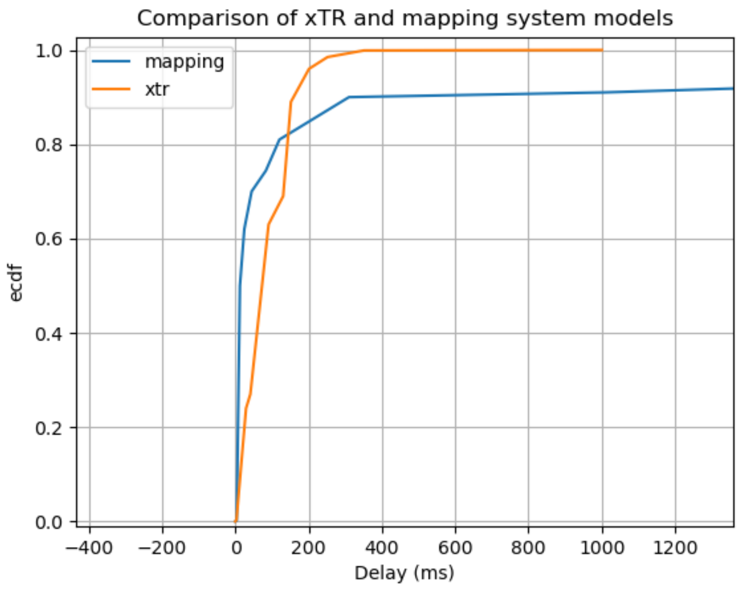


Figure 5.13: Comparison of the xTR and mapping system delay models.

# Senders \ # Receivers	1	10	100	1000
1	1000	300	100	10
10	300	100	/	/
100	100	/	/	/
1000	10	/	/	/

Figure 5.14: Table of the number of simulation performed by setup.

Nonetheless, it is obvious that the resources necessary to maintain the same performance are not the same. This can be very easily seen in the time taken to run a simulation, from less than a second for one sender and one receiver to multiple hours with a thousand connections. We also had scalability issues with the simulator, primarily with the queue size in routers and the data rate of links. The defaults of 1000 packets per queue and 5Mbps per link was not enough for one thousand connections and packets were dropping, in a simulation this is easy to increase, but it is clear that the resource usage of those notification mechanisms can be important if not limits are in place.

5.2.4 Global discussion

By looking at all the results above, we can conclude that Publish/Subscribe is indeed faster, as expected, but there is a strong influence of the Mapping System speed on the results of the SMR procedure. Using any notification process, is of course way faster than none, especially if the TTL are long.

It is however important to note that those results correspond to the best case scenario of every method. A major point is that in the absence of the backward connexion for the SMR procedure, this method regress to the standard LISP behaviour. And such a situation is not uncommon with the use of UDP or traffic engineering. Publish/Subscribe doesn't build upon any external assumption, outside the mapping system.

Chapter 6

Conclusion

In this thesis, we provided 3 main contributions:

- The implementation of the Publish/Subscribe extension in ns-3.
- The improvement of the existing LISP implementation in ns-3.
- A comparison of Publish/Subscribe with the SMR procedure and LISP standard behaviour.

Our implementation of Publish/Subscribe, allowed us to run multiple simulations and provide data about the Publish/Subscribe extension. The data about this extension being inexistent, we believe that it will enable further research on the subject. The improvement to the existing implementation also was a great help in running bigger simulations. We made sure that those improvements are usable even without the Publish/Subscribe modifications, and hope that it will benefit further research on the LISP protocol as a whole.

Our results clearly indicated that Publish/Subscribe is indeed an improvement over the existing notification method of LISP. We observed that an efficient mapping system caching scheme would benefit the SMR procedure mechanism. However, such a caching system would still need to be designed with care in order to provide sufficient consistency, especially in the case of the SMR procedure. We also noted that over the higher effectiveness of LISP Publish/Subscribe, the extension also move the handling of the notifications entirely to the mapping system, which ensure that the data plane do not interfere with the notification mechanism. This is not the case for the SMR procedure, that needs a backward connection and therefore only work for LISP Tunnel Routers that are both an ITR and ETR. The existence of techniques like traffic engineering, that can result in an asymmetric routing, where the path in one direction is different from the other, is also a problem to the SMR procedure.

Our results indicate that the Publish/Subscribe extension of LISP is promising, however further research is needed. This thesis has some shortcoming that could be fixed. First, the lack of data available for the LISP protocol made the elaboration of models hard, and our models are far from perfect. Particularly, a better model for the mapping system would be a huge improvement, as we have been using the DNS system as a source of data in order to create our model. This make for a correct approximation, but forbid the usage of our results as an exact indicator of the actual delay that we can expect from the Publish/Subscribe extension. Another problem related to our model is that an unexpected interaction between our models and the implementation, that runs the

LISP protocol as defined in standards, creates results that are too beneficial to the SMR procedure. This could be fixed by using a more complex model for the mapping system, that would take into account the EID that is requested to generate a delay, avoiding the aberrations in the delays of identical successive requests to the mapping system in our simulations.

Furthermore, simulations will never be enough to grasp the full complexities of the LISP protocol and the Publish/Subscribe extensions. In order to move toward a potential implementation in the global internet network, many real world experimentations will be necessary.

Bibliography

[1] Lionel Agbodjan. *Towards A LISP Simulator*. http://hdl.handle.net/2268.2/1654.

[2] Vina Ermagan et al. *NAT traversal for LISP*. Internet-Draft draft-ermagan-lisp-nat-traversal-19. Work in Progress. Internet Engineering Task Force, May 2021. 29 pp. URL: https://datatracker.ietf.org/doc/draft-ermagan-lisp-nat-traversal/19/.

[3] Dino Farinacci, Michael Kowal, and Parantap Lahiri. *LISP Traffic Engineering Use-Cases*. Internet-Draft draft-ietf-lisp-te-10. Work in Progress. Internet Engineering Task Force, Mar. 2022. 18 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lisp-te/10/.

[4] Dino Farinacci et al. *LISP Mobile Node*. Internet-Draft draft-ietf-lisp-mn-12. Work in Progress. Internet Engineering Task Force, July 2022. 25 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lisp-mn/12/.

[5] Dino Farinacci et al. *Locator/ID Separation Protocol (LISP) Control-Plane*. Internet-Draft draft-ietf-lisp-rfc6833bis-31. Work in Progress. Internet Engineering Task Force, May 2022. 60 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lisp-rfc6833bis/31/.

[6] Dino Farinacci et al. *The Locator/ID Separation Protocol (LISP)*. Internet-Draft draft-ietf-lisp-rfc6830bis-38. Work in Progress. Internet Engineering Task Force, May 2022. 48 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lisp-rfc6830bis/38/.

[7] Vince Fuller et al. *Locator/ID Separation Protocol Alternative Logical Topology (LISP+ALT)*. RFC 6836. Jan. 2013. DOI: 10.17487/RFC6836. URL: https://www.rfc-editor.org/info/rfc6836.

[8] Vince Fuller et al. *Locator/ID Separation Protocol Delegated Database Tree (LISP-DDT)*. RFC 8111. May 2017. DOI: 10.17487/RFC8111. URL: https://www.rfc-editor.org/info/rfc8111.

[9] Mattias Gabriel, Luigi Iannone, and Benoit Donnet. "Lisp Mapping System as DoS Amplification Vector". In: *IEEE Networking Letters* 3.1 (2021), pp. 36–39. DOI: 10.1109/LNET.2021.3050814.

[10] Michael Hoefling, Michael Menth, and Matthias Hartmann. "A Survey of Mapping Systems for Locator/Identifier Split Internet Routing". In: *IEEE Communications Surveys Tutorials* 15.4 (2013), pp. 1842–1858. DOI: 10.1109/SURV.2013.011413.00039.

[11] Geoff Huston. *BGP in 2021 - The BGP Table*. https://blog.apnic.net/2022/01/06/bgp-in-2021-the-bgp-table/.

[12] Yue Li. *Futur internet services based on LISP technology*. Networking and Internet Architecture, Télécom ParisTech. 2018.

[13] Emeline Marechal. *Simulating LISP with NS3*. Master thesis, University of Liège, Faculty of Applied Sciences. 2019.

[14] D. Meyer, L. Zhang, and K. Fall. *Report from the IAB Workshop on Routing and Addressing*. RFC 4984. RFC Editor, Sept. 2007. URL: https://www.rfc-editor.org/rfc/rfc4984.txt.

[15] nsnam. *ns-3 | a discrete-event network simulator for internet systems*. https://www.nsnam.org/.

[16] Roland van Rijswijk-Deij et al. "A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements". In: *IEEE Journal on Selected Areas in Communications* 34.6 (2016), pp. 1877–1888. DOI: 10.1109/JSAC.2016.2558918.

[17] Alberto Rodriguez-Natal et al. *Publish/Subscribe Functionality for LISP*. Internet-Draft draft-ietf-lisp-pubsub-09. Work in Progress. Internet Engineering Task Force, June 2021. 14 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-lisp-pubsub/09/.

[18] Damien Saucez and Benoit Donnet. "On the Dynamics of Locators in LISP". In: (2012). Ed. by Robert Bestak et al., pp. 385–396.

[19] Damien Saucez et al. "Interdomain traffic engineering in a locator/identifier separation context". In: (2008).

[20] *The BGP Instability Report*. https://bgpupdates.potaroo.net/instability/bgpupd.html.

[21] *The BGP IPv6 Instability Report*. https://bgpupdates.potaroo.net/instability/v6-bgpupd.html.