

# Further Improvements of an existing IPv6 Network Mobility Test-bed

**JosepM<sup>a</sup>Tomàs Sanahuja**

Munich (Germany)  
September 2010 - March 2011

Tutor: Jaime M. Delgado Mercé  
Barcelona School of Informatics - FIB



Universitat Politècnica de Catalunya - UPC



TriaGnosys GmbH  
Supervisors: Àngels Via Estrem and Eriza Hafid Fazli





## Agraïments

A tots aquells que al llarg d'aquests últims anys han fet possible que seguís endavant en els moments més difícils i de necessitat, moltes gràcies a tots vosaltres.

A tu Joan que durant 3 anys has estat el meu cap en el departament del TSC, per tota la flexibilitat otorgada al llarg d'aquests anys i l'ajut prestat quan l'he hagut de menester. Han estat uns tres anys els quals en guardaré molt bons records, gràcies per tot.

Gràcies també Antonio per aquests tres anys al departament del TSC. Han estat tres anys dels que n'estic molt content de haver pogut viure i après. Gràcies.

Voldria agrair a la Núria Castell. Al llarg de l'últim període de la carrera t'he tingut com a professora de Intel·ligència Artificial i he pogut mantindre converses sobre el com encarar la meua carrera en un llavors, valorar possibilitats per a fer un màster, tràmits per a beques a Japó i finalment la possibilitat de realitzar el projecte de final de carrera (PFC) a l'extranger a Múnic, Alemanya. Ha estat una molt bona experiència de la qual sempre en mantindré un bon record i de la qual sempre me'n sentiré privilegiat de haver-la viscut.

Agrair a l'Àngels i a l'Eriza per el seu ajut i suport en la tasca dins el projecte. També agrair les recomanacions del meu tutor Jaime Delgado en el redactat final de la memòria.

Finalment agrair a tota la meua família per el suport donat al llarg d'aquests anys.

A la meua àvia Pepeta que no vaig arribar mai a conèixer. Al meu avi Rafel Tomàs que en pau descansi. Al meu avi Blai Sanahuja que ens va deixar farà un any, ell sempre va procurar per la meua educació i benestar, sempre, sempre en va tenir cura i fins els seus últims dies en mantingué. Estic segur que n'estaries orgullós i n'estaries cofoi. En els últims dies abans que ens deixéssis et vaig dir:

“Estigues tranquil per què tiraré endavant i me'n ensortiré”

A dia de avui et puc dir:

“He complert, i continuaré tirant endavant, pots estar-ne segur. Res és impossible si amb esforç i voluntat un hi persisteix, tard o d'hora sempre se'n acaba obtenint resultat”

I finalment a les dues persones que ho han donat tot per a mi en aquesta vida i que tant de sacrifici han fet, les dues persones per a les quals el meu present i el meu futur ho són i ho han estat sempre tot per a ells. A el meu pare i a la meua mare.

“Gràcies”



<b>List of Figures</b>	<b>v</b>
<b>Abbreviations</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Goals . . . . .	3
1.3 Report Structure . . . . .	4
<b>2 IPv6 Overview</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Basic characteristics of IPv6 . . . . .	5
2.3 Introduction to Mobility IPv6 protocol . . . . .	8
2.3.1 Mobile IPv6 protocol . . . . .	8
<b>3 Testbed Current Status</b>	<b>11</b>
3.1 Introduction to NEWSKY testbed architecture . . . . .	11
3.1.1 NEWSKY background . . . . .	11
3.1.2 NEWSKY testbed architecture . . . . .	12
3.2 Introduction to SANDRA scenario . . . . .	12
3.2.1 SANDRA goals . . . . .	12
3.2.2 SANDRA architecture and test-bed . . . . .	13
3.3 Aim of traversing IPv6 traffic over a satellite IPv4 network . . . . .	15
3.4 Relevant and non relevant traffic types in SANDRA . . . . .	15

<b>4</b>	<b>Mobility + IPv4 Traversal</b>	<b>17</b>
4.1	Considered Protocols as a solution for SANDRA . . . . .	17
4.1.1	M6T protocol . . . . .	17
4.1.2	DSMIPv6 . . . . .	19
4.1.3	NeXT . . . . .	20
4.2	Protocol comparison . . . . .	24
4.3	NeXT as option chosen . . . . .	26
<b>5</b>	<b>Testbed Automation</b>	<b>31</b>
5.1	IPv6 Network Mobility Testbed Architecture - Technical features . . . . .	31
5.2	How the IPv6 Network Mobility Test-bed was switched on . . . . .	33
5.3	Considered options for the automation . . . . .	34
5.3.1	Use of Cron + Bash scripts: . . . . .	34
5.3.2	Use of /etc/rc.local + Bash scripts: . . . . .	35
5.3.3	Use of /etc/rc.local + Java GUI + Bash scripts . . . . .	37
5.4	Final option: Use of /etc/rc.local + Java GUI + Bash scripts . . . . .	38
5.4.1	Different features of the Java GUI . . . . .	38
5.4.2	Where are the scripts located and how are they executed ? . . . . .	41
<b>6</b>	<b>NeXT software architecture/design</b>	<b>43</b>
6.1	NeXT first protocol version design . . . . .	43
6.2	New NeXT requirements in SANDRA scenario . . . . .	44
6.3	Two different NeXT design approaches . . . . .	46
6.3.1	NeXT multiprocess . . . . .	47
6.4	Threads Introduction . . . . .	53
6.4.1	What is a Thread . . . . .	53
6.4.2	Why using threads? . . . . .	55
6.4.3	What is pthread library? . . . . .	55
6.5	NeXT threads . . . . .	56
6.6	NeXT design chosen . . . . .	60

**7 Test Cases 61**  
7.1 Case 1 . . . . . 61  
7.2 Case 2 . . . . . 63

**8 Conclusions and future Work 67**  
8.1 Conclusions . . . . . 67  
8.2 Future work . . . . . 68

**9 References and Bibliography 69**





---

## List of Figures

---

2.1	IPv6 header format . . . . .	6
2.2	Mobility protocol (1) . . . . .	8
2.3	Mobility protocol (2) . . . . .	9
3.1	NEWSKY test-bed architecture . . . . .	12
3.2	SANDRA topology . . . . .	13
3.3	SANDRA architecture . . . . .	14
3.4	Air traffic network . . . . .	15
3.5	Inmarsat coverage Area . . . . .	16
3.6	SANDRA traffic . . . . .	16
4.1	M6T deployment . . . . .	18
4.2	Outer packet in MR and Outer packet in HA . . . . .	18
4.3	Packet in HA after UDP decapsulation is done by M6T entity, the outer IPv6 header is the NEMO header . . . . .	18
4.4	Packet in CN . . . . .	19
4.5	NeXT . . . . .	20
4.6	NeXT headers with IPv6 and mobility legend . . . . .	21
4.7	NeXT steps from MN to NeXTSlave . . . . .	22
4.8	NeXT steps from CN to MN . . . . .	23

4.9	SANDRA scenario with NeXT . . . . .	26
4.10	Flow packet . . . . .	27
5.1	Testbed Architecture . . . . .	32
5.2	Cron table entry format . . . . .	34
5.3	Cron predefined variables . . . . .	35
5.4	Rc.local file . . . . .	36
5.5	Java GUI . . . . .	38
5.6	Flow Checking Nodes Status . . . . .	39
5.7	Flow Start Node . . . . .	40
5.8	Flow Stop Node . . . . .	41
5.9	Stop VM command . . . . .	41
5.10	Flow Reboot Node . . . . .	42
6.1	NeXT real world architecture . . . . .	43
6.2	NeXT real world using satellite link . . . . .	44
6.3	NeXT64 Master flow . . . . .	45
6.4	NeXT46 Master flow . . . . .	46
6.5	Multiple instances . . . . .	48
6.6	Multiple instances with libnetfilter . . . . .	49
6.7	NeXT multipleprocess architecture . . . . .	49
6.8	NeXT launcher flow . . . . .	50
6.9	Interfaces hardcoded . . . . .	52
6.10	Interfaces hardcoded with dynamism . . . . .	52
6.11	PID Handshaking . . . . .	53
6.12	UNIX thread . . . . .	54
6.13	Thread Memory . . . . .	55
6.14	Thread Memory . . . . .	56
6.15	NeXT multiprocess table linkage . . . . .	57
6.16	NeXT threads table linkage . . . . .	58
6.17	NeXT threads SIGNALS . . . . .	59
7.1	extract of iptables man Linux page - Marking Section . . . . .	64
7.2	extract of iptables man Linux page - Matching Section . . . . .	64
7.3	Mip6d.conf interfaces with BID . . . . .	65

---

## Abbreviations

---

AOC	Aircraft Operational Communications
APC	Air Passenger Communications
API	Application Programming Interface
AR	Access Router
ARv6	Access Router version 6
ATM	Air Traffic Management
ATS	Air Traffic Service
BC	Binding Cache
BGAN	Broadband Global Area Network
BU Ack	Binding Update Acknowledgment
CoA	Care-of Address
CN	Correspondent Node
CPU	Central Processing Unit
DSMIPv6	Double Stack
ESA	European Space Agency
FTP	File Transfer Protocol
GUI	Graphical User Interface
HA	Home Agent
HoA	Home Address
IPsec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IRIS	Internet Routing in Space
L-DACS	L-band Digital Aeronautical Communication System
MIPv6	Mobile Internet Protocol v6
MN	Mobile Node
MNN	Mobile Node Network
MR	Mobile Router
NAPT	Network Address Protocol Translation

NAT	Network Address Translation
NEMO	Network Mobility
NEWSKY	Networking the Sky for Aeronautical Communications
NeXT	Network Crossing via Translation
OS	Operating System
PC	Personal Computer
PID	Process Identifier
Ra	Router advertisement
RAM	Random-access memory
RFC	Request for comments
ROHC	Robust Header Compressor
SANDRA	Seamless Aeronautical NETWORKING throughout the integration of Data links
	Radios and Antenas
ScGW	Security Gateway
SESAR	Single European Sky ATM Research
VDL2	Virtual Drum Line
VLAN	Virtual Lan Area Network
VM	Virtual Machine
WiMAX	Worldwide Interoperability for Microwave Access

# Chapter 1

---

## Introduction

---

This final project report is the result of the work done in the Full Internship made within TriaGnoSys GmbH company in Munich, Germany.

### 1.1 Motivation

The overall air transportation sector is currently under significant stress. With the demand in aircraft operations expected at least to double by the 2025 time frame, there are well-founded concerns that current air transportation systems will not be able to accommodate this growth. Existing systems are unable to process and provide flight information in real time, and current processes and procedures do not provide the flexibility needed to meet the growing demand. New security requirements are affecting the ability to efficiently move people and cargo. In addition, the growth in air transportation has provoked community concerns over aircraft noise, air quality and airspace congestion. In summary, with the tools and procedures in use today, the effective increase of air traffic will be fundamentally limited and it is already approaching its limits. Focusing on communications related aspects, the following high-level requirements can be identified, in order to allow future systems to be compatible with the expected air-traffic increase:

- Pilots situation awareness shall be improved
- Capacity at airports, today one of the main limiting structural factors, shall be increased
- ATS shall be primarily based on highly reliable data communication
- AOC data traffic shall strongly increase for efficient airline operations

## 1.1. Motivation

---

- Passengers and cabin communications systems shall be further developed
- Safety critical applications shall need diverse means to reach ground for global availability and higher reliability
- A simplification of on-board network architecture shall need convergence of protocols and interfaces

To cope with these issues, new communication concepts are being developed in SESAR, in the AOC/APC domains. They aim at the definition of an access to an open system. This results in a collection of communications technologies targeted at specific operational settings. This represents a considerable extra burden to be carried by the aircraft, should the new radio links be implemented in stand-alone equipments. Moreover, although it has been suggested that the new systems will eventually replace the legacy communications systems, the likelihood is that there will be a lengthy period where aircraft will be fitted with all of the systems for global interoperability. This is the forecast expressed by SESAR, and the additional airborne equipment required during this transition phase severely threatens the realization of the future communications vision. Hence, a different approach aiming at a broader level of integration is needed to achieve the required increase of capacity, safety, security and efficiency of air transportation operations while at the same time keeping complexity and cost of on-board networks and equipments within a sustainable level.

To enable efficient exchange of information in aeronautical communications, the different available and future communication link technologies need to be interconnected, forming a global heterogeneous aeronautical communication network. Several European research projects are being undertaken with the goal to develop improved communication infrastructure for aeronautical communications. Among them is the project NEWSKY (Networking the SKY, <http://www.newsky-fp6.eu>) which aims at developing ATM networking concept based on IPv6 protocol stack. Within this project, TriaGnoSys was responsible for developing an IPv6-based network test-bed to simulate network handover between a satellite and a terrestrial communication link.

What has been mentioned is indeed the truly integrated modular approach for a global aeronautical network and communication architecture proposed by SANDRA (Seamless Aeronautical Networking through integration of Data links Radios and Antennas).

As NEWSKY project was nearly to end, the test-bed was going to suffer some internal modifications, for example the addition of IPsec capabilities introducing IPsec Security Gateways. SANDRA project was the chosen one to define the new test-bed configuration.

This project have had four main parts. The first one is the automation of an IPv6 Mobility test-bed. This test-bed was designed and used in order to simulate handovers between different networks and assure that network connectivity in the aircraft still remained. It will be explained in chapter 4. This test-bed was previously build under the scope of another project done before. Different options to automate the test-bed and make the whole process of launching or managing it quicker were considered, however, only one option has been chosen. Apart from that a Graphical User Interface to manage the test-bed main functions has been implemented. This GUI provides an usable and an easy interactive way to work with the test-bed.

During the project some the test-bed functionalities, configurations and services have been continue being modified. As an example given, some weeks before my project started the test-bed architecture changed, some IPsec functionalities were added and some new entity nodes were added (IPsec Security Gateways (IPsec ScGW), they are explained in later sections). This addition of new entities modified the behavior and purposes of some existing nodes within the test-bed, for instance, before the addition of ScGWs a node acted as a server, after this addition it no longer acted as server because other node take its responsibility of acting as a server. Therefore some new services had to be moved from node to node (apache server, ftp server and so on ...).

The second part of the project has been a whole process of analysis. During this process RFC's and papers were read, all these papers were about technologies that represented an alternative to NeXT protocol. NeXT protocol is the software I have worked with, modified, extended and improved. A comparison between the different protocols and NeXT have been done, the aim of this comparison has been to find the most appealing and suitable protocol to deploy in the SANDRA scenario which was not yet defined, an approach of SANDRA architecture was available during the project realization. NeXT is the protocol that has been chosen to be deployed in SANDRA, therefore NeXT has needed to be modified and extended according to what SANDRA topology and architecture defines.

The third part has been the process of design and implementation of the new version of NeXT protocol adapted to SANDRA topology. All new NeXT requirements have defined, different technologies have been considered as a resource to use (e.g thread libraries), then two designs have been thought. Design 1 represents a valid solution that with some modifications turns into Design 2 which uses a different technology from Design1, the reasons to this is explained in chapter 6. In this third process the implementation task it is also included, the Design1 and Design2 implementation has been performed, section 6 gives more details about that.

And finally the last part in this project, the fourth part. It has been the documentation part of the software within the company and also the final report.

It has to be said that, despite it is not mentioned in the report, RFC's from ROHC protocol has been read because at the beginning one task scheduled to be fulfilled was to made some coding modifications to an existent ROHC source code, its code was studied but no action was taken. It was considered as a non task for my project. Apart from that during the project XFRM technology was investigated but finally not used because XFRM was linked to DSMIP protocol and it will be seen that DSMIP is discarded as a valid option.

## 1.2 Goals

The principal aim of this project is to:

1. Automate and improve an existing IPv6 Mobility test-bed
2. Improve an existing protocol named NeXT which main function is to translate packets from IPv6 to IPv4 and viceversa.
3. Allow NeXT and Robust Header Compressor protocols to work together at the same time in the test-bed without any problems.

### 1.3 Report Structure

A brief explanation of how the report is structured will be given for an easy understanding to all person that in a close future want to base its work projects into this project report. The report first talks about the motivation of the project and mentions from an existent general scope that have been worked with. Tasks that have been done within this project are also mentioned in the Motivation section. Afterwards *chapter 2* and *chapter 3* introduces technology background that have been used. Chapter 2 and chapter 3 have no relation between them, the first one talks about IPv6 technology which is used in the whole project, a brief description of some of its features are explained, whereas chapter 3 talks about the existing and the upcoming architecture in which the NeXT protocol is going to be deployed. This last chapter is very important because it introduces the test-bed architecture and changes that this testbed suffered and was going to suffer, but also it is important because some important decisions in terms of the new NeXT protocol design were based taking the test-bed topology architecture into account.

Bear in mind one thing, *chapter 2* and *chapter 3* describes technologies that in this project have been used, however, other technologies will be introduced in other chapters in order to justify things and give a better comprehension of some aspects. This project is mainly about NeXT protocol and an IPv6 Mobility test-bed, however, NeXT protocol is deployed in the test-bed and both use other technologies for specific cases.

Chapter 4 introduces the analysis process of choosing the most appealing protocol option to be considered for the coming SANDRA test-bed scenario. Some aspects of protocols are explained, there are also some comparisons between them and some conclusions.

Chapters 1 to 3 are more theoretical, whereas chapter 4 introduces the analysis process in the project.

Chapter 5 is about the test-bed Automation, it is explained the problem that there was, options considered and actions taken, some theory about some technologies is also given. Chapter 6 is a chapter that joins the design and implementation process in NeXT new version. In chapter 4 an introduction to old NeXT first version was given, in chapter 6 a more technical explanation is given focused on the implementation aspects. An introduction to threads is also given.

In chapter 7 some test cases are explained. Chapter 8 with conclusions and future work and finally chapter 9 introduces references and bibliography.



# Chapter 2

---

## IPv6 Overview

---

The Internet Engineering Task Force (IETF) has been developing Internet Protocol version 6 since the 1990s, this protocol is expected to replace in a midterm the current IPv4 due to the scarce of new available addresses to assign, its address space is nearly to finish in a few years. Nowadays some companies and organizations are getting involved in the use of IPv6, an example given is the Olympic games of China where public net services used IPv6.

## 2.1 Introduction

IPv6 uses a 128-bit address, whereas IPv4 uses only 32 bits. It means the new IPv6 address space supports  $2^{128}$  (about  $3.4 \times 10^{38}$ ) addresses. This expansion provides considerable flexibility in allocating addresses and routing traffic. As address size in IPv6 is 128 bits it solves the problem of scarcity address space in IPv4. It also adds the feature that addresses can be auto configured, multicast routing has been improved and new address type has been introduced, called Anycast address explained in 2.1.

## 2.2 Basic characteristics of IPv6

### 2.2.0.1 Header

The IPv6 header has a fixed length of 40 bytes, 32 bytes are for addresses and 8 bytes for general header information. The header of an IPv6 packet is specified in RFC 2460<sup>1</sup>. Figure 2.1 shows the IPv6 format.

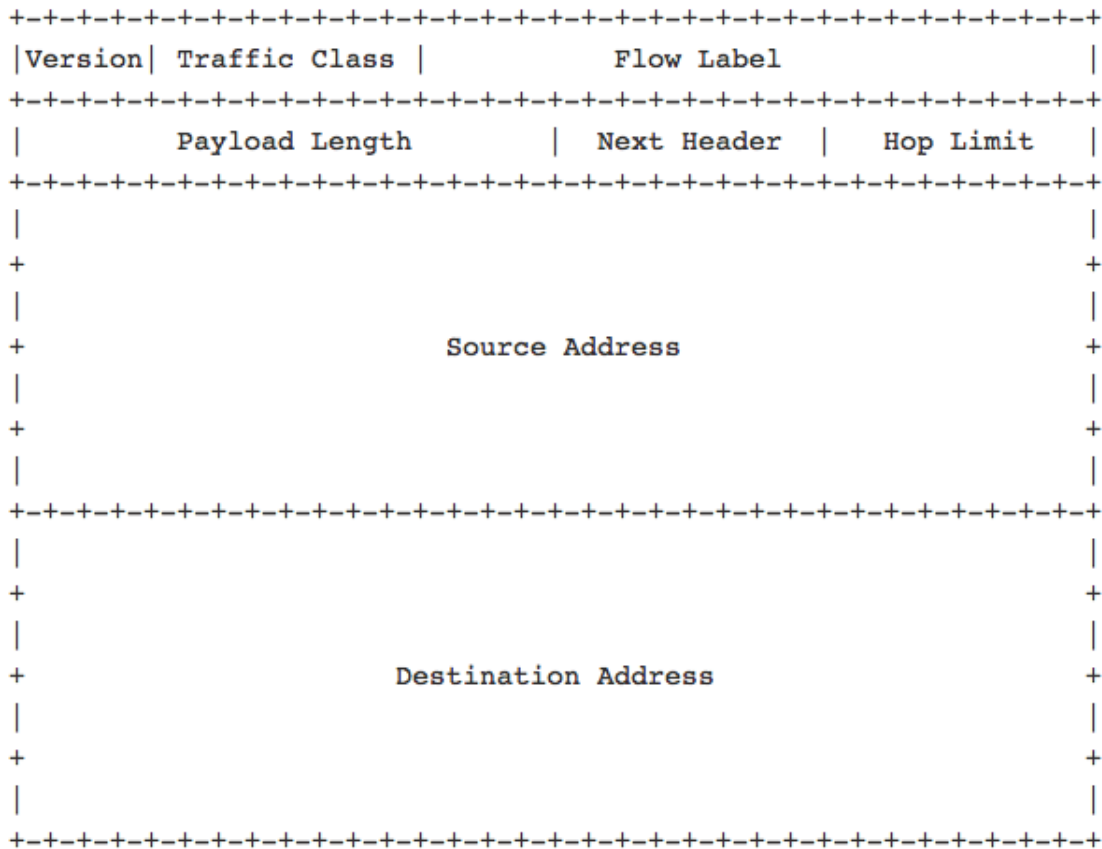


Figure 2.1: IPv6 header format

1. **Version** (*4 Bits*). Internet Protocol version number. In this case is 6.
2. **Traffic Class** (*1 Byte*). This field replaces the Type of Service field in IPv4. Facilitates the handling of real-time data and any other data that requires special handling.
3. **Flow Label** (*20 Bits*) This field distinguishes packets that require the same treatment, in order to facilitate the handling of real-time traffic: routers keep track of flows so they can process packets belonging to the same flow more efficiently because they do not have to reprocess each packet's header.
4. **Payload Length** (*2 Bytes*). Length of the IPv6 payload counting the extension headers.
5. **NeXT Header** (*1 Byte*). Identifies the type of header immediately following the IPv6 header.
6. **Hop Limit** (*1 Byte*). Expresses a number of hops permitted to reach the destination node. Every forwarding node decrements the number by one.
7. **Source Address** (*16 bytes*). It contains the address of the node that generates the packet.
8. **Destination Address** (*16 bytes*). It contains the address of the destination node.

Many fields that are considered “Options“ in IPv4 are considered as “Extension Headers“ in IPv6. Extension Headers can be from zero to more extension headers in a packet but they are always between the IPv6 header and the next layer header. For further details consult IPv6 RFC 2460a

### 2.2.0.2 Addressing

IPv6 has 3 types of addresses:

- **Unicast address:** unicast address uniquely identifies a single interface of an IPv6 node within the scope of the unicast address type. Depending of the prefix the address can be:
  - *Link-local:* Is for use on a single link. Its prefix is *fe80* and it can be used for auto configuration mechanisms, for neighbor discovery and others.
  - *Site-Local:* Its prefix is *fec0*. The address contains subnet information. It can only be routed within the site, therefore routers can not route packets outside the site. A site could be defined as a network area scope.
  - *Global:* It uniquely identify a node all over the world and are structured as Aggregatable Global Unicast *Addresses*<sup>2</sup>.
- **Multicast:** It comprises a group of addresses, this address can have different scopes. Packets that are sent to a multicast address, are sent to all addresses that takes part in the multicast group.
- **Anycast:** It is assigned to multiple interfaces (usually in multiple nodes). It can not be differentiated from a Global address, all nodes that have an Anycast address type configured are the only ones to identify it as Anycast type address. When a packet is sent to an Anycast address it is sent only to one of them, usually the nearest one.

An IPv6 address consists of three parts: a global routing prefix, a subnet ID, and an interface ID. The addresses are written as follows: *IPv6 address/prefix length*

Addresses are divided into eight 16-bit hexadecimal blocks, separated by colons, an example is given:

*2001:5c0:1505:6101:0000:0000:0000:3*

IPv6 addresses can be simplified as follows:

- Leading zeros can be simplified by skipping them.
- Consecutive zeros can be replaced by a semicolon but it can only appear only once in an IPv6 address.

After the transformation the IPv6 address looks like as follows:

*2001:5c0:1505:6101::3*

## 2.3 Introduction to Mobility IPv6 protocol

Every node has an IPv6 globally unique address and a link local address. These addresses are related to the router in the network they are connected, thus when the node changes its point of attachment, IPv6 global unique addresses and link local addresses change.

When the address changes the existent connections terminate because they can not be maintained. This happens because connections such as TCP are characterized by the address and ports of each one. It means that the information related with the interface ( the one that changes its point of attachment) changes and the other edge can not reach it because it has the old configuration address. A new connection with the new configuration must be opened.

Mobile IPv6 overcomes this issue and allows maintaining the existing connections the mobile node that moves to a new network is communicating with. Section 2.3.1 explains briefly how Mobile IPv6 protocol works.

### 2.3.1 Mobile IPv6 protocol

When a node moves to a new network it acquires a new address, called Care-of Address (CoA) from the new local router. A CoA is a temporary address that the mobile device acquires when it joins to a foreign network. It identifies the current point of attachment to the internet and makes it possible to connect from a different location without changing its permanent address, the Home Address (HoA). When the mobile device moves to a new network it also maintains HoA. When a node enters to a new network the IPv6 Mobility protocol works as follows:

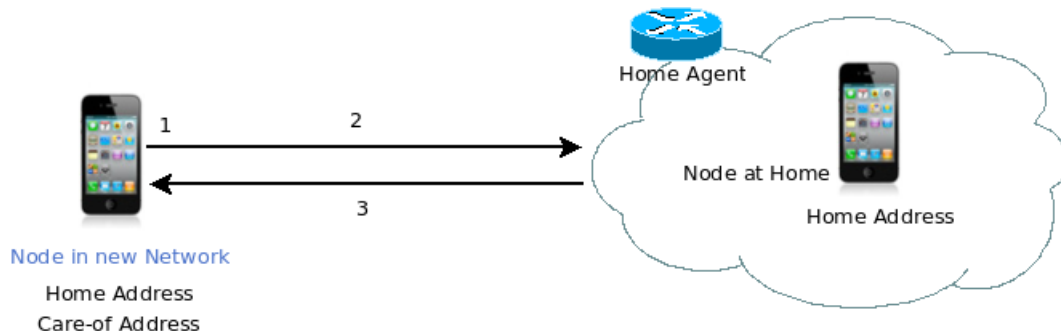


Figure 2.2: Mobility protocol (1)

Before explaining how the IPv6 Mobility protocol works some terminology will be introduced for a better comprehension.

- Care-of Address (CoA): It has been explained in the initial section paragraph.
- Home Address (HoA): Is the permanent address that identifies the point of attachment to the mobile node's home network.

- Home Network: Network to whom a mobile node belongs, the one that configures its permanent address (the HoA).
- Foreign Network: Network in which a mobile node is operating when away from its home network.
- Home Agent: Node that stores information about mobile nodes whose permanent home address is in the home agent's network.
- Binding Update: Packet that creates or renew an association of the HoA with a CoA.
- Binding Cache: Table maintained by a node. It contains the current bindings (associations) for mobile nodes.
- Binding Update Acknowledgment: Confirmation of the BU packet.
- Correspondent Node: Node that communicates with a mobile node that is out of its own network.

The following steps correspond to figure 2.2:

1. Address auto configuration, called Care-of Addresses.
2. The node sends a Binding Update (BU) to the HA, it binds the CoA to the HoA in the HA which is in the Home network. If the node does not know the HA it uses HA discovery process.
3. When the HA receives the BU, it stores in the Binding Cache (BC) and sends back a Binding Update Acknowledgment.

After receiving the BU Ack, the mobile node (mobile device) must use the Return Routability Process in order to send packets to the Correspondent Node. The following steps are related to this process. Figure 2.3 illustrates them.

1. The node sends a Home test Init (HoTI) message indirectly to the correspondent node, tunneling the message through the Home Agent.

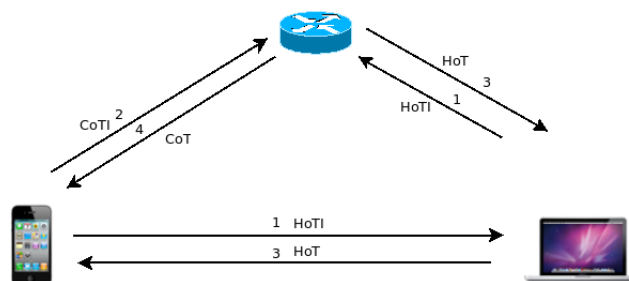


Figure 2.3: Mobility protocol (2)

### 2.3. Introduction to Mobility IPv6 protocol

---

2. The mobile node sends a Care-of Test Init (CoTI) message directly to the correspondent node.
3. The correspondent node sends a Home Test (HoT) message in response to the HoTI message.
4. The CN sends a Care-of Test (CoT) message in response to the CoTI message.

After doing this process the nodes can communicate directly using Routing Optimization (RO) instead of using HA. When a node is in a foreign network it sends packets from its care-of address and includes the mobile node's Home Address in a Home Address option.

One advantage of using RO is that it frees the HA from traffic, HA tunnel interfaces tends to become a bottleneck.

### 3.1 Introduction to NEWSKY testbed architecture

#### 3.1.1 NEWSKY background

The development of efficient aeronautical communication systems is currently a predominant topic in view of the expected saturation of ATM communications by 2020-2025 due to air traffic increase. In addition, the envisaged paradigm shift in ATM as developed in SESAR and the existing high market demand for passenger communications are the driving factors for the modernization of aeronautical communications.

It is foreseen that different services, data links and networking solutions will be deployed.

Different services with highly diverse requirements shall coexist and partly or totally share the aeronautical network infrastructure. No single service on its own justifies the cost implication of a new communication system. Trends for the different aeronautical services include:

- Air Traffic Services (ATS): It will be primarily based on highly safety-related data communication whereas voice communication will be mostly used as fallback solution
- Airline Operational Communications (AOC) data traffic: It will strongly increase for efficient airline operations
- Air Passenger Communications (APC) systems: They foreseen to be further developed to meet passengers expectations of on-board broadband communication services.

According to both SESAR and the Future Communications Study jointly performed by Eurocontrol and FAA under Action Plan 17, these services will use ground-based, satellite-based, aircraft-to-aircraft and airport communication systems to fulfill the requirements: a satellite link (new standard developed within ESA IRIS program), an airport link (WiMAX), a high data rate air-ground link (L-DACS-1/2) and support of legacy data links (e.g. VDL2). In addition, further data links for APC are expected to be deployed.

### 3.1.2 NEWSKY testbed architecture

Figure 3.1 shows the old NEWSKY test-bed architecture. It has not got any IPsec Security Gateways nor IPsec functionalities.

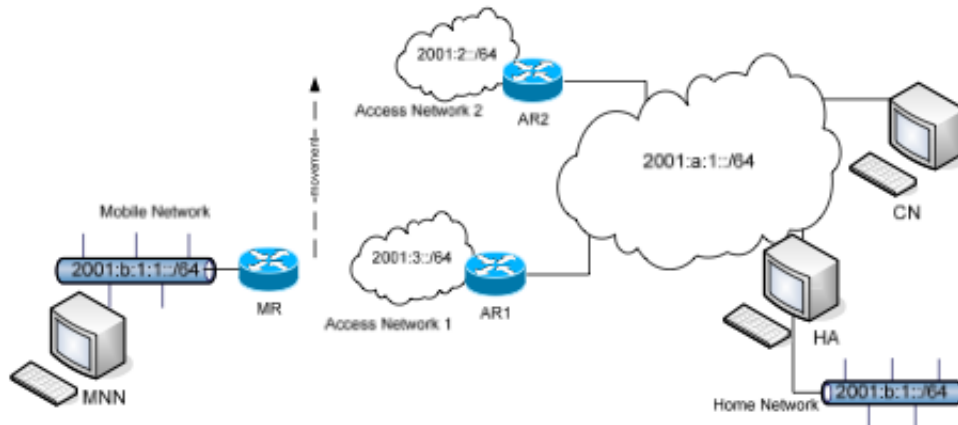


Figure 3.1: NEWSKY test-bed architecture

A further explanation of this type of topology is done in section 3.2.2.

## 3.2 Introduction to SANDRA scenario

### 3.2.1 SANDRA goals

The SANDRA concept consists of the integration of complex and disparate communication media into a lean and coherent architecture that:

- Provides and manages seamless service coverage across all airspace domains and all aircraft classes
- Sustains growth in the service market and enables easy plug-in of future radio technologies through modularity and configurability
- Is upgradeable, easy reconfigurable and radio technology independent
- Is distributed and instantiated into consistent ground-based and airborne sub-networks ensuring full interoperability. SANDRA covers from RF and avionics components up to the middleware layer of the on-board network, assembled and integrated under the most stringent safety and security requirements. Ultimately, SANDRA pursues the architectural integration of aeronautical communication systems using:
  - Well-proven industry standards like IP, IEEE 802.16 (WiMAX), DVB-S2, Inmarsat SwiftBroadBand



- A set of common interfaces
- Standard network protocols having IPv6 as final unification point to enable a cost-efficient global and reliable provision of distributed services across all airspace domains and to all aircraft classes.

### 3.2.2 SANDRA architecture and test-bed

Figure 3.2 shows how SANDRA topology is in a laboratory environment. The upper box on the left is a lab environment that emulates a real scenario. The most right box of the picture can be considered as a black box, what happens inside it is transparent to us. It is the one that provide network access to the technology services such as satellite links (INMARSAT). The most left box is the one that corresponds to the airplane and that is also emulated in a laboratory environment. Keep in mind that the current testbed architecture is not like the one of the picture, but in a close future it should be something alike.

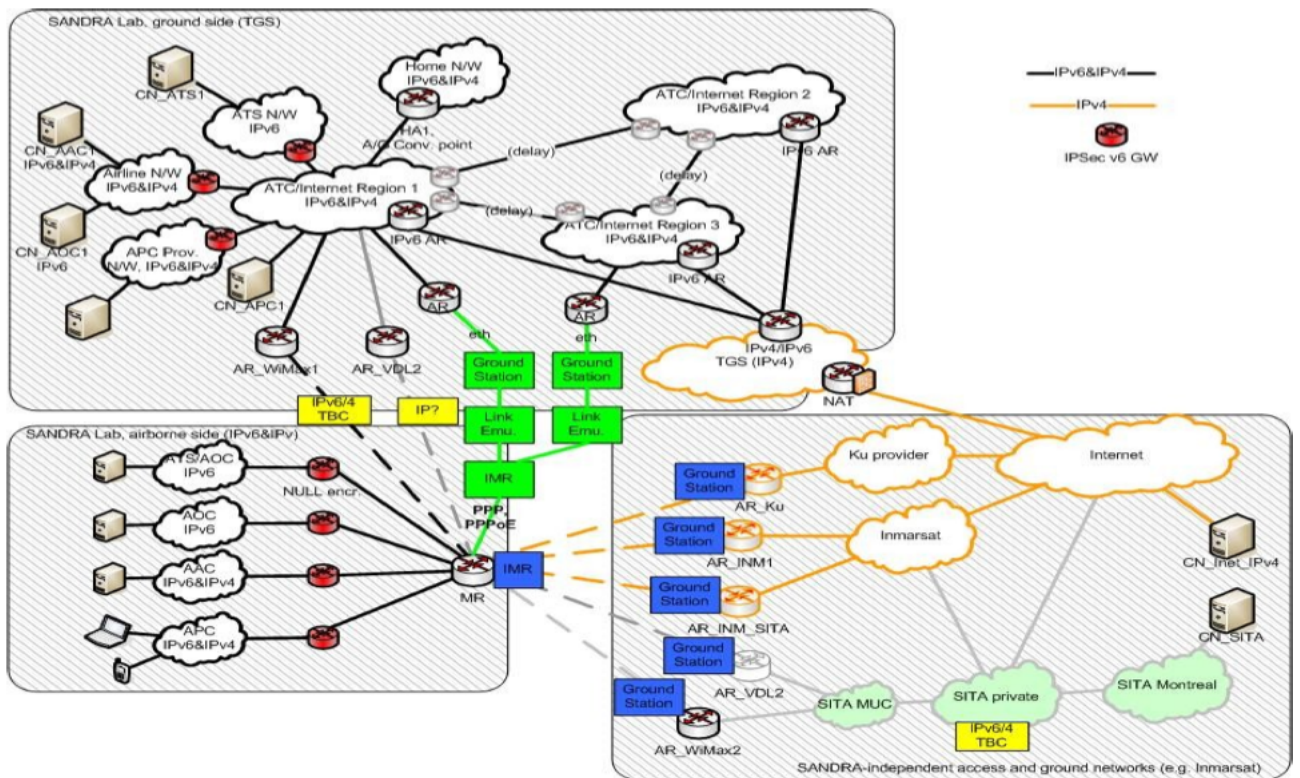


Figure 3.2: SANDRA topology

In figure 3.2 a box labeled *IMR* can be identified. *IMR* is an entity attached to *MR* that manages interface and traffic information transparent to *MR*. Its mainly task is to take control of interfaces configuration. For instance, *IMR* determines that interface *eth1* from *MR* is configured to be used with *AR\_Ku*, it also handles situations where interfaces in *MR* go down and have to be reassigned to another *AR\_* link.

### 3.2. Introduction to SANDRA scenario

The current architecture of the testbed which I have dealt with is the one shown in figure 3.3.

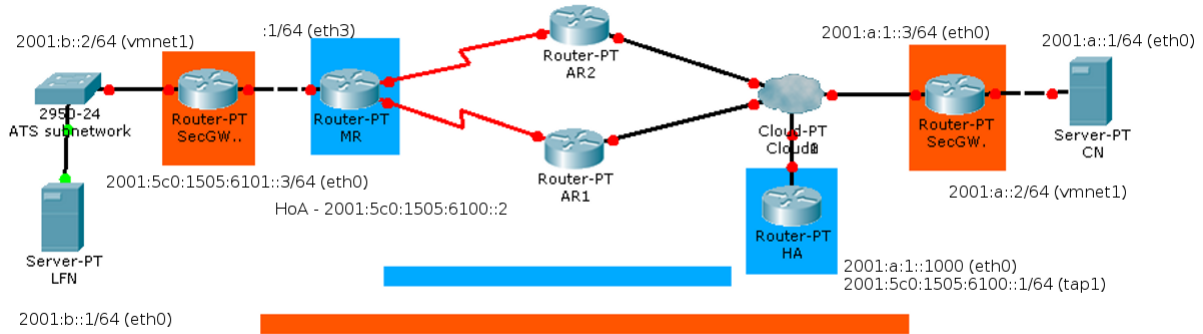


Figure 3.3: SANDRA architecture

In orange there is respective Security Gateways establishing a secured tunnel between them, and over that tunnel MR-HA establishes its own tunnel for mobility purposes, in blue color.

This current architecture is an approach of what SANDRA test-bed would be in a close future. *IPsec*<sup>3</sup> capabilities were being added meanwhile I started working with the testbed. In section 5.1 this topology is explained more deeply, however, the basics has been previously explained in chapter 2, so please refer to this chapter to be familiar with Mobility concept.

Internet Protocol Security (IPsec) is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPsec also includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session.

IPsec capabilities have been introduced due to the will of adding security in the traffic flow and Mobility signaling (both in MR for Binding updates and Binding acknowledgments). All traffic behind IPsec Security Gateways is encrypted and decrypted in the end edge IPsec Security Gateway.

In section 7.1 from chapter 8, a matter related with IPsec will be explained. However, bear in mind that IPsec is a topic out of the scope of this project. All knowledge about IPsec have been acquired by listening some conversations between the workmate in charge of the IPsec project and its supervisor, and also by some other technical questions to him because of his modifications within the testbed. Apart from that some of my tasks related with test-bed automation and NeXT protocol have made me deal with some specific IPsec aspects that will be explained in section 7.2.

### 3.3 Aim of traversing IPv6 traffic over a satellite IPv4 network

IPv6 has not arrived in satellite communications yet. Thus IPv6 packets have to be sent through the IPv4 satellite link. Looking at figure 3.3 an aircraft network can be identified, it is the one behind MR and which is not attached to any AR.

In real world it would look like in figure 3.4.

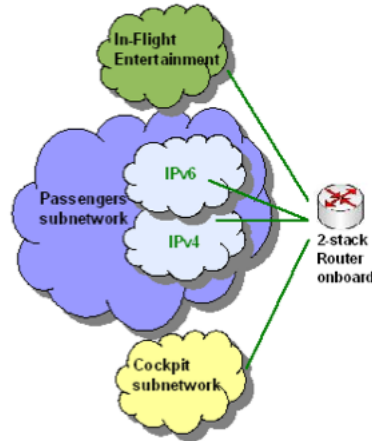


Figure 3.4: Air traffic network

The satellite IPv4 network used and considered for the project is the *INMARSATBGAN*<sup>4</sup> (*BroadBand Global Area Network*) which has 3 Inmarsat-4 satellites. Satellite communication have an advantage; its wide coverage area of each satellite which covers all over the world. However, they have some disadvantages: the atmosphere and the ionosphere add errors that provoke packet losses. In figure 3.5 the coverage area of Inmarsat can be appreciated.

### 3.4 Relevant and non relevant traffic types in SANDRA

One thing that SANDRA project stands for is to define a model that allows both IPv4 and IPv6 coexist during the transition process of the IPv6 instauration all around the world. It is important that fact because there is still a huge amount of IPv4 devices and traffic applications, thus they do not support IPv4 and an infrastructure has to be given to those devices. See figure 3.6

IPv4 traffic (either public or private) can be tunneled by IPsec version 4 tunneling, it was proven by the person working in the IPsec project. IPsec version 4 tunneling means that IPv4 traffic is tunneled by both IPsec SecGW. When a packet reaches one SecGW an outer IPv6 header is added to the packet (apart from the IPsec header), those headers are only removed when the packet reach the other SecGW, it is when IPv4 is used again. An issue shows up if this is done, but only if private IPv4 addresses are used due to uniqueness address aspect. That is solved by creating a pool of IPv4 private addresses in both networks sides behind each SecGW. This pool allocates a specific rang of addresses and its purpose is to avoid addresses conflicts.

### 3.4. Relevant and non relevant traffic types in SANDRA

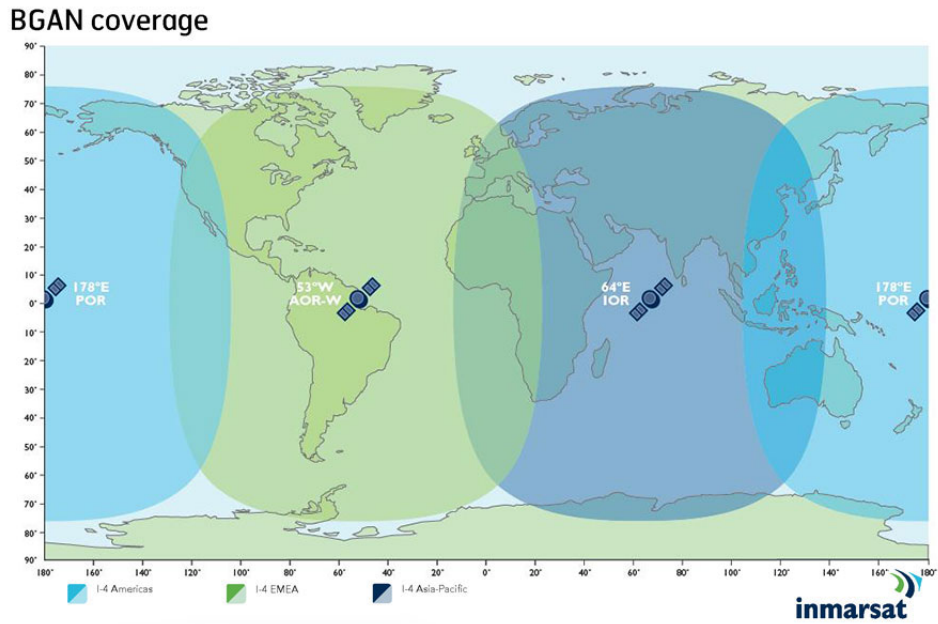


Figure 3.5: Inmarsat coverage Area

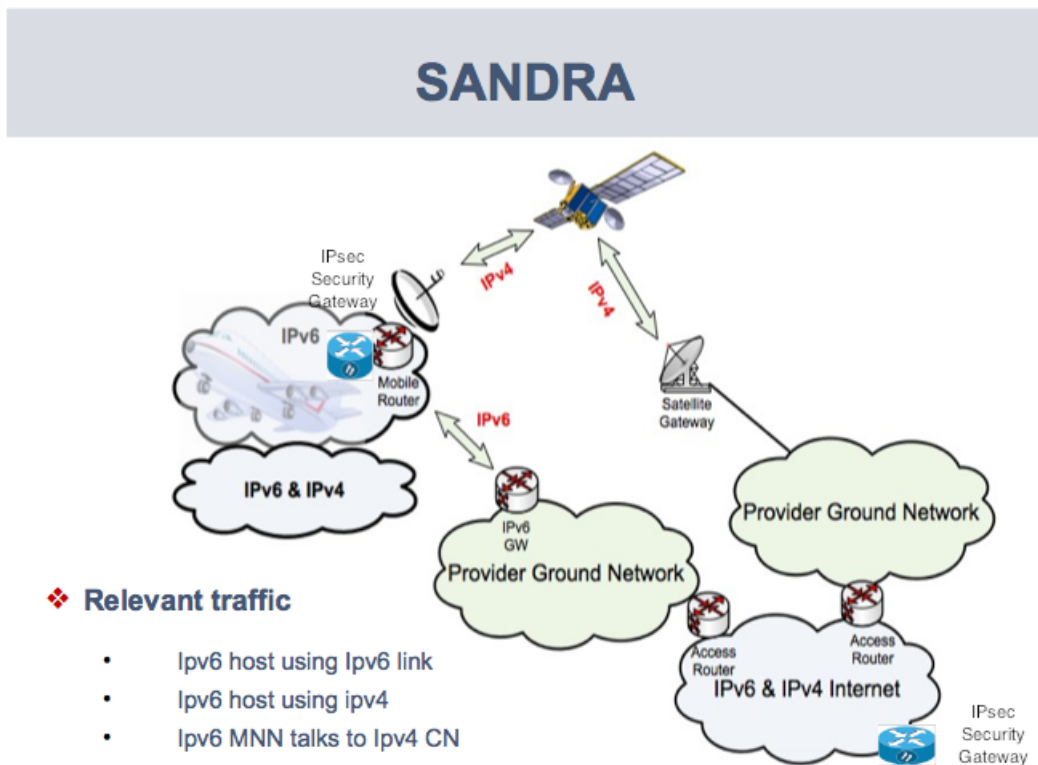


Figure 3.6: SANDRA traffic

# Chapter 4

---

Mobility + IPv4 Traversal

---

## 4.1 Considered Protocols as a solution for SANDRA

In this section it will be discussed the most suitable protocol option to deploy and adapt to the needs of the SANDRA project (see 3.2). A list of nowadays existent protocols has been considered to its deployment in SANDRA. A deep study of them has been performed, its documentation (RFC's and other public available documentation resources) has been read. It has also been done determining its applicability in SANDRA scenario, its time-effort costs and finally drawbacks and advantages of them.

### 4.1.1 M6T protocol

M6T is a protocol that allows transporting IPv6 data over an IPv4 network, in our case of study a satellite link. See figure 4.1 for a better understanding. M6T does the following steps:

1. It provides an IPv6 interface to the node in order to let MIPv6 do Mobility (e.g. do NEMO encapsulation).
2. M6T entity in MR adds an UDP header and an extra outer IPv4 header to allow UDP tunneling.
3. M6T entity in HA decapsulates the packet by erasing the outer IPv4 and UDP headers
4. The decapsulated packet passes to the MIPv6 from Home Agent and it decapsulates the NEMO IPv6 outer headers

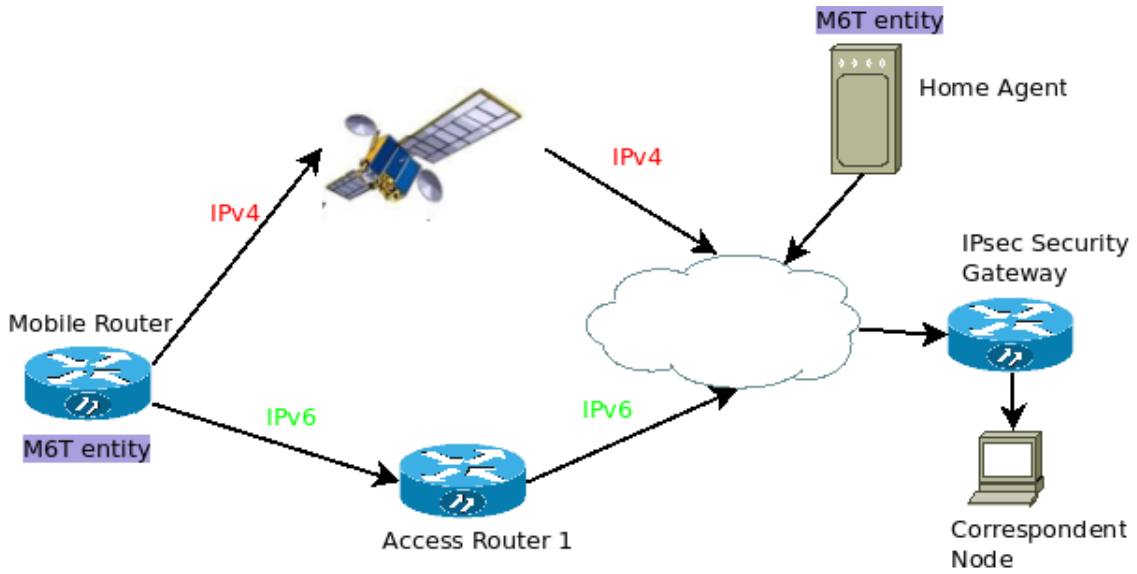


Figure 4.1: M6T deployment

5. Once NEMO headers have been erased, HA routes the packet to the appropriate IPv6 CN node.

The aim of M6T is to provide an easy, adaptive and simple solution to let IPv6 packets traverse IPv4 networks. This solution works in Linux only, for further details see <sup>5</sup>.

The following figures show which is the packet's structure in each point of the path to its destination, taking the MR as origin and the CN as destination.

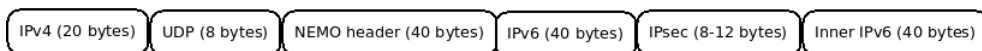


Figure 4.2: Outer packet in MR and Outer packet in HA

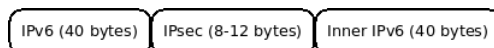


Figure 4.3: Packet in HA after UDP decapsulation is done by M6T entity, the outer IPv6 header is the NEMO header

MR knows where the HA is because it is defined in the `mip6d.conf` file from the Linux `MIPv6`<sup>17</sup> daemon.

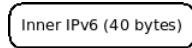


Figure 4.4: Packet in CN

### 4.1.2 DSMIPv6

DSMIPv6<sup>6</sup> is a protocol that stands for Dual Stack Mobile IPv6 and aims at extending the Mobile IPv6 and NEMO Basic Support protocols to support movements in IPv4 networks. DSMIPv6 is an extension of the Mobile IPv6 protocol thus it follows standards which is always a good asset.

What DSMIPv6 seeks is to save the use of two Mobility protocols, IPv4 and IPv6. Using both at the same time will probably lead into a non stable situation of conflicts. As an extension it provides two main modifications in Mobile IPv6 headers structure. A brief introduction to DSMIPv6 will be given, for further details see<sup>6</sup>.

DSMIPv6 also consider cases where a mobile node moves into a private IPv4 network and gets configured with a private IPv4 care-of address. In these scenarios, the mobile node needs to be able to traverse the IPv4 NAT in order to communicate with the HA. IPv4 NAT traversal for Mobile IPv6 is presented in<sup>6</sup>.

In DSMIPv6 it is assumed that the HA can be reached through a global and unique IPv4 address. It can happen that is configured with a private address, therefore a NAT will be needed.

The typical scenarios where DSMIPv6 acts will be introduced:

- Scenario 1: IPv4-only foreign network
  - In this scenario the mobile node is located in a network where it can only provides an IPv4 CoA.
- Scenario 2: Mobile node behind a NAT
  - In this scenario, the mobile node is in a private IPv4 foreign network that has a NAT device connecting it to the Internet. If the home agent is located outside the NAT device, the mobile node will need a NAT traversal mechanism to communicate with the home agent.
- Scenario 3: Home agent behind a NAT
  - In this scenario the thing is even more complicated because the HA is located in an IPv4 private network, thus the communication between the MN and the HA is more complex. However there is the assumption that the HA has a global IPv4 address that make it reachable around all over the Internet. This global IPv4 address is not physically configured in an interface in HA, it is configured in a device that supports NAPT and stores HA routing information.
- Scenario 4: Use of IPv4-only applications

#### 4.1. Considered Protocols as a solution for SANDRA

---

- In this scenario the mobile node is located in an IPv4/IPv6 network capable, however, it uses some application that requires IPv4 traffic.
- Scenario 5: IPv6 and IPv4-enabled networks
  - In this scenario, the mobile node should prefer the use of an IPv6 care-of address for either its IPv6 or IPv4 home address.

##### 4.1.3 NeXT

NeXT is a protocol created and owned by TriaGnoSys GmbH company in Munich, Germany. The main aim of NeXT protocol is allowing the transmission of IPv6 packets through IPv4 links (i.e. Satellite). This is done by translating IPv6 headers to IPv4 and again to IPv6 after traversing these links.

It can also work the other way round, to traverse IPv6 networks with IPv4 packets.

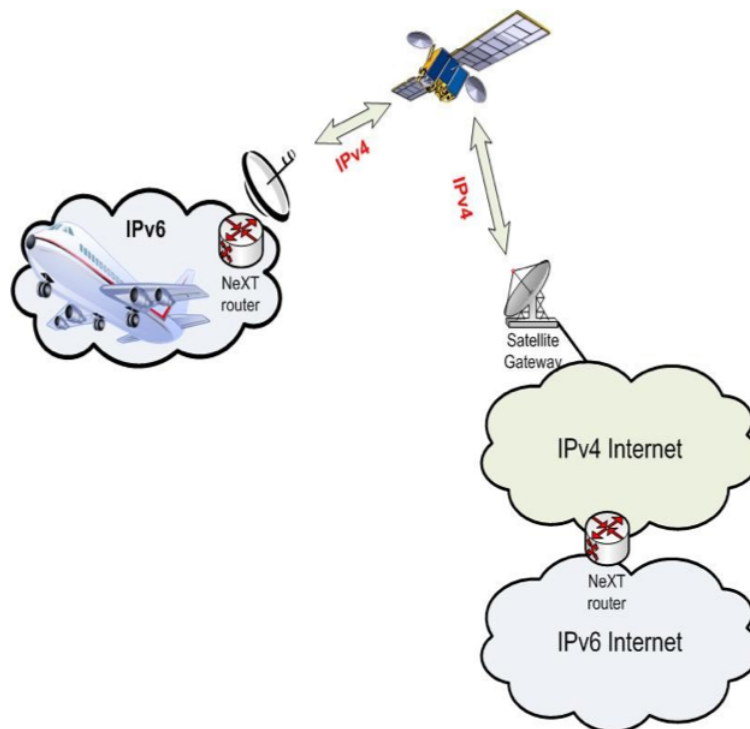


Figure 4.5: NeXT

NeXT consists of two main entities,

- NeXTMaster
- NeXTSlave



Both NeXTMaster and NeXTSlave consist of two sub-entities

- NeXTMaster = NeXT64Master and NeXT46Master
- NeXTSlave = NeXT64Slave and NeXT46Slave

NeXT64 entity translates IPv6 traffic into IPv4 traffic and NeXT46 entity translates IPv4 traffic into IPv6.

In figure 4.6 the legend for the different types of packets used in figures 4.7 and 4.8 are shown.

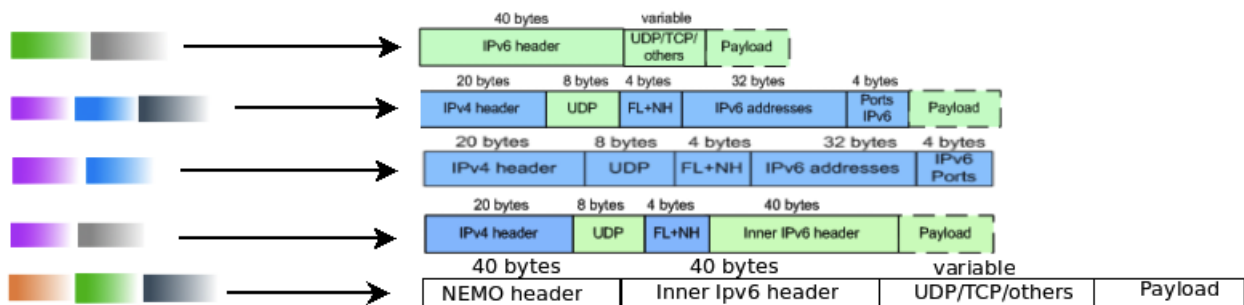


Figure 4.6: NeXT headers with IPv6 and mobility legend

Figure 4.7 shows in a real world scenario which are the steps that NeXT does when MN starts talking to CN and MN is in a foreign network:

1. A node (MN) generates the the first packet of a session, the destination is CN outside of the current MN network, thus the packet is sent to MR
2. When MR receives the packet it applies NEMO protocol to the packet (adding an extra IPv6 outer header). Afterwards it looks for an entry in the translation table(structure that holds the information required to translate the packet from one IP version into other), as it is the first packet of the session it does not find any entry matching the packet. Then a new entry is created and addresses and ports are assigned.
3. When an entry for the new packet is created the packet is sent with the IPv6 addresses and original ports as shown in figure 4.6.
4. The packet reaches NeXTSlave, it looks for an entry in the table, as it does not find any entry it reads the IPv6 addresses from the packet and also the ports, then it creates a new entry for the packet.
5. The packet is translated and sent to the HA (remember that there is NEMO header in the packet).

#### 4.1. Considered Protocols as a solution for SANDRA

- The packet reaches the HA, due to in step 2 an outer IPv6 Header was included by NEMO using as source address the MR's HoA and as a destination address the HA's HoA. When the packet arrives to HA it decapsulates the packet (removes the NEMO outer header which includes the Home Addresses previously mentioned) it reads the CoA form the packet and searches it within its Binding cache (BC), if it is found in the BC then is routed to the CN.

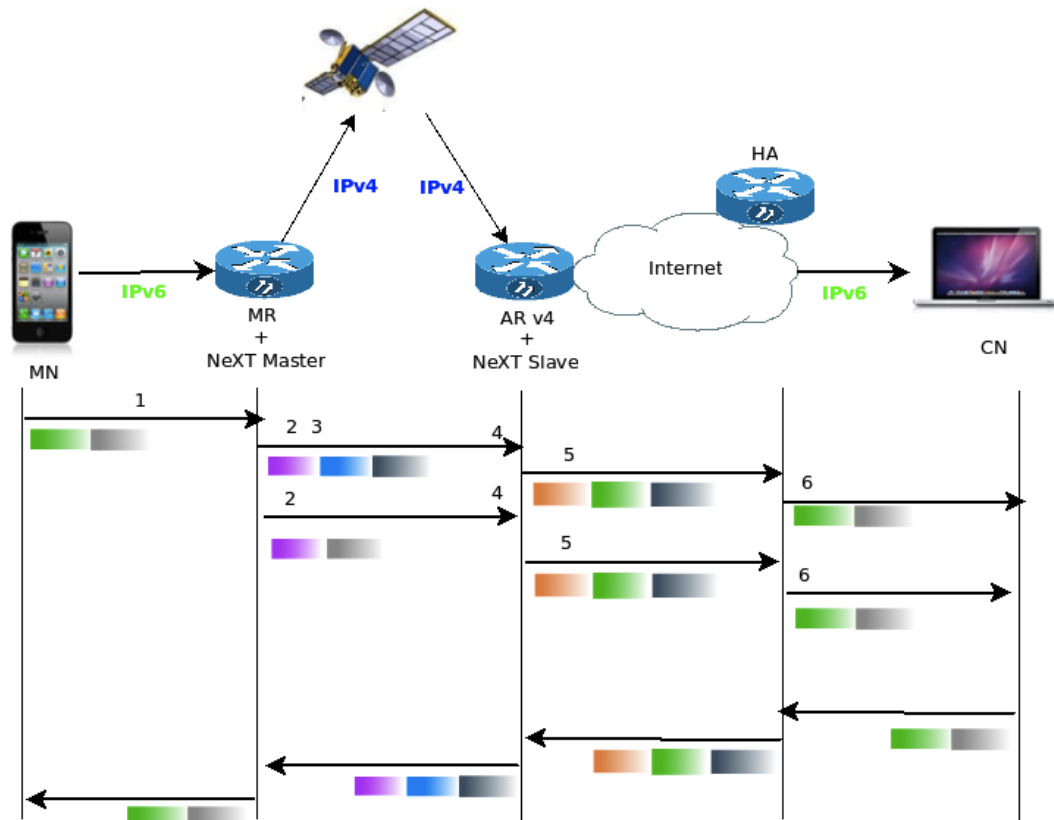


Figure 4.7: NeXT steps from MN to NeXTSlave

Figure 4.8 show in a real world scenario which are the steps that NeXT does when CN starts talking to MN and MN is in a foreign network:

- CN sends IPv6 packet to MN
- Packet goes to HA
- HA consults its BC and if the destination is found adds NEMO outer header (it includes source and destination HoAs)
- HA forwards the packet to the destination HoA(MR HoA)
- NeXT64Slave in AR receives the packet

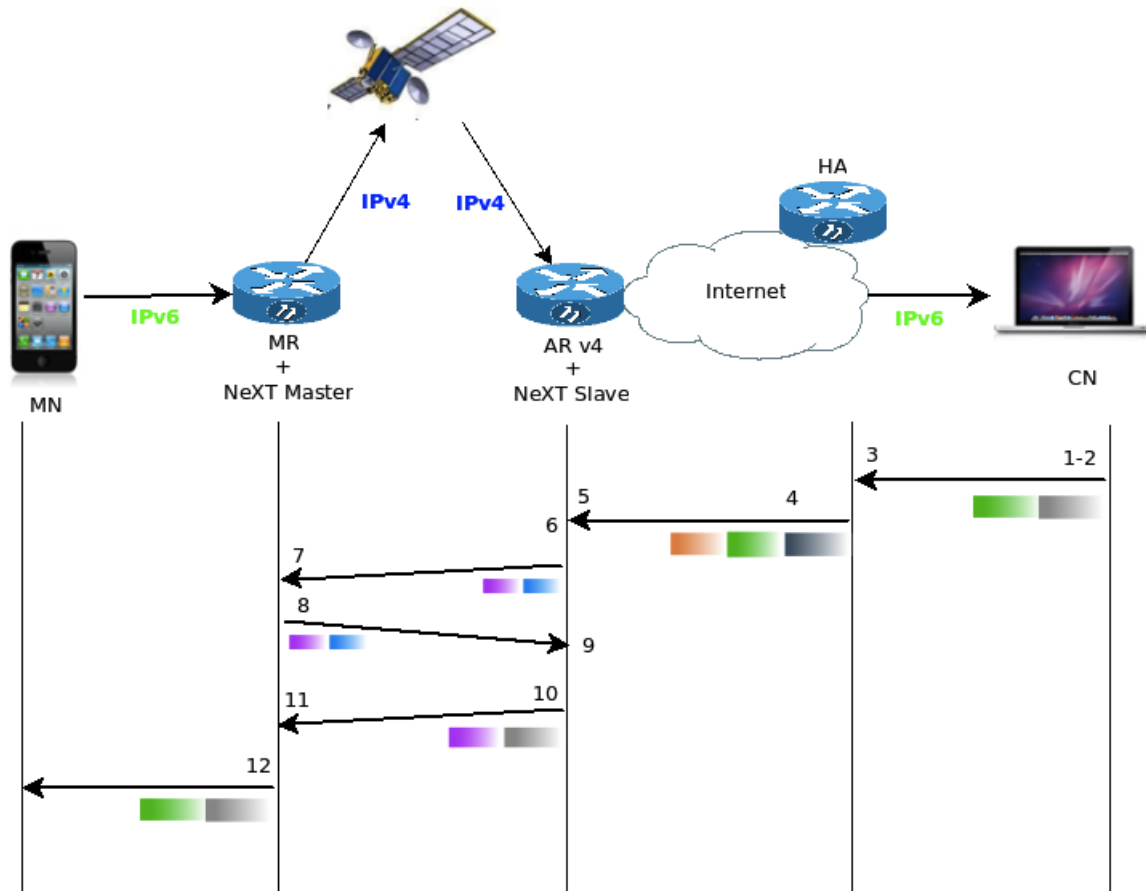


Figure 4.8: NeXT steps from CN to MN

7. NeXT64Slave looks in the table if there is any entry that match the packet, as it does not find any NeXT64Slave sends a signaling packet to the NeXTMaster with the addresses and ports in IPv6 and asking to NeXTMaster an allocation of a IPv4 address. NeXTSlave knows the master address due to it is specified in a configuration file, however, the address is also provided by the first packet that NeXTMaster sends to NeXTSlave to start an exchange of traffic data. For further details about this configuration please see figure 4.10.
8. NeXTMaster assigns a IPv4 address and ports and creates a new entry in its table structure.
9. NeXTMaster sends a request ack to NeXTSlave with the new information.
10. When NeXTSlave receives the information, it creates a new entry in the translation table and translates the header. Bear in mind that this packet that is going to be sent does not contain any IPv6 address since IPv6 address have been previously stored in NeXTMaster when the signaling request packet was sent.
11. NeXTSlave sends the packet
12. NeXTMaster receive the packet and translates it

13. NEMO protocol removes the outer IPv6 NEMO header, and forward to its recipient.

NeXT protocol uses `libnetfilter_queue` library. `Libnetfilter_queue` is an userspace library providing an API to packets that have been queued by the kernel packet filter. It is part of a system that deprecates the old `ip_queue /libipq` mechanism. `Libnetfilter_queue` has been previously known as `libnfnetlink_queue`. It allows receiving queued packets from the kernel `nfnetlink_queue` subsystem and also issuing verdicts and/or reinjecting altered packets to the kernel `nfnetlink_queue` subsystem.

NeXT stands for an efficient way of using IPv4 links bandwidth. One kind of IPv4 link that nowadays is used is satellite links. It provides communication in vast areas where other technologies and mechanisms can not, for example in transatlantic airplanes flights. However, despite satellite links are currently thought as an appealing solution to intercommunicate (specially in the skyline) it has one feature, using a satellite link is expensive in terms of time/traffic usage. NeXT introduces traffic overhead saving by how it translates IPv6 packets into IPv4 format.

NeXT saves space bandwidth on using the satellite link due to how it translates packets from IPv6 to IPv4. NeXT does save space by sending at first all the IPv4 and IPv6 information required to NeXTSlave to do a mapping between the two IP versions (IPv6 address with ports 85000 and 86000 corresponds to the IPv4 address 184.67.9.5). When the NeXT Slave has a mapping entry in itself data structure the following packets from the flow only carry IPv4 information (IPv6 mobility headers also) due to IPv6 information had been previously stored in the NeXTMaster and NeXTSlave. It is not just UDP encapsulation, it is something more efficient than this.

## 4.2 Protocol comparison

A list of drawbacks and advantages will be given for two of the three protocols introduced. M6T is not considered for the following reasons.

In the earlier process of analyzing its features I thought that it was not worth to consider it anymore because it was something alike to NeXT protocol and furthermore it added a constant overhead into the satellite link. What is considered as overhead is the outer IPv4 and the UDP headers added to the original packet.

A satellite link is something very expensive in terms of traffic usage, normally whoever uses a satellite link will be normally paying for an amount of traffic consume through the link. Therefore adding always 28 bytes of extra headers was not an appealing matter, NeXT solution managed even better and in an efficient way this issue.

This is why for that reason M6T will not be considered in this comparison and pros and cons. DSMIPv6 and NeXT will be the only ones.

Pros and Cons for DSMIPv6 protocol are shown below.

Advantages :

- Allow v4 CoA

- Allow v4 HoA
- Is a Standard

Disadvantages :

- DSMIPv6 implementation is buggy
- Can't run MultipleCoA-patched implementation + NEMO
- NAT traversal is not implemented
- Overhead with UDP addition

The DSMIPv6 implementation that exists and has been considered in this analysis is the one from *Nautilus*<sup>7</sup>. One of the drawbacks says that the MultipleCoA-patched implementation and NEMO patch cant not work together. There is not so much information about this drawback, it is only mentioned in its official website. Therefore it has been deduced that MultipleCoA-patched implementation and NEMO patch cant not work together due to how they are installed. Both of them are patches to the Linux kernel that add and replace lines from it, It can happen that there can be and overlapping of some kernel lines, therefore some functionalities of those patches can not work properly.

In the test-bed, one very important feature was the use of multiple addresses, it was a must. Multiple-CoA had to be present in the test-bed. Moreover NEMO was also a feature of the testbed that was a must, on the top of that, they were both already installed in the testbed. For that reason it was an important drawback that subtracted points into consideration to DSMIPv6.

Apart from that DSMIPv6 implementation performance was not quite stable. It showed so many troubles with handovers performance making the whole system unstable because after a serie of repetitive handovers the DSMIPv6 software crashed. There were other bugs such as some configuration policies defined by DSMIPv6 that were not removed after exiting from it. Having a stable behavior was something sought and DSMIPv6 didn't seem to provide it.

Nowadays DSMIPv6 implementation hasn't got some functionalities, for example DSMIPv6 lacks of NAT traversal implementation. However, not all about DSMIPv6 is negative, it has positive features like the possibility of using a CoA v4 address and the dynamism it introduces because it makes possible to have only one Mobile IP protocol deployed. DSMIP is being also considered by companies such as Qualcomm whose Smart Mobility team is developing a solution based on the Dual Stack Mobile IP (DSMIP).

Pros and Cons for NeXT protocol are shown below.

Advantages :

- Reliability of its functionality
- Satellite bandwidth performance

Disadvantages :

- It is not a Standard

### 4.3 NeXT as option chosen

NeXT protocol has been finally the chosen option for its deployment in the SANDRA scenario. It allows having Multiple-CoA and NEMO running at the same time. Furthermore it guarantees an efficient use of the satellite link bandwidth as it has been explained in section 4.1.3.

Adopting NeXT for the SANDRA project has its implications and a deep process on analysis has to be performed. NeXT has two main entities, NextMaster and NextSlave. It was first implemented as a one-to-one channel, it means NeXTMaster entity only communicates with NeXTSlave entities, a one-to-one speech. However within SANDRA this change due to its architecture.

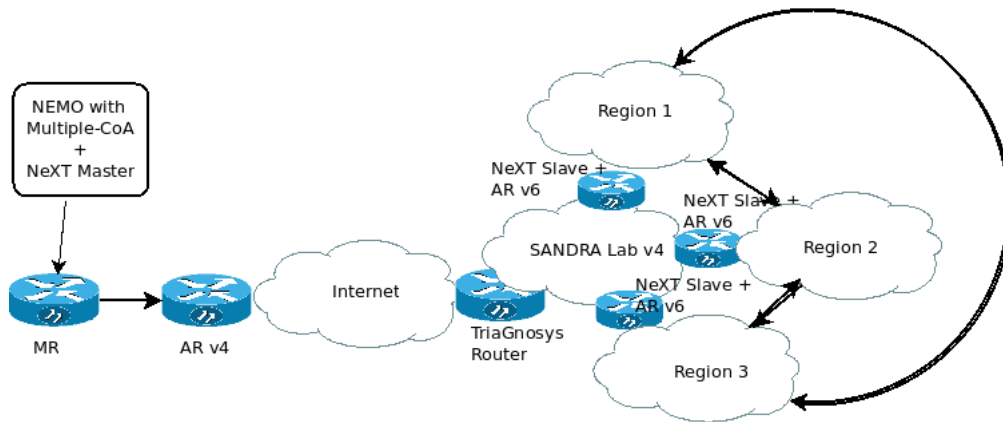


Figure 4.9: SANDRA scenario with NeXT

What comes now is the whole analysis that was made to determine how NeXT would look like if it would have been used in SANDRA and which new requirements would appear as a result of this process.

An explanation of figure 4.9 will be given next. Figure 4.9 shows the SANDRA topology scenario using NeXT. In this case what is on the right side of TriaGnoSys router is an emulation of SANDRA scenario but emulated inside TriaGnoSys network. TriaGnoSys router acts as an ARv4 that leads to an IPv4 network. Within this IPv4 network there are some ARv6 that correspond each one to a specific Regional Area IPv6 Network. A Regional Area Network is an IPv6 Network that can be related whether to a geographic area or a type of service. Lets suppose that there are two mobile nodes (two mobile devices) that have paid for some satellite data bandwidth and also for a certain time duration. Lets identify the two mentioned mobile nodes as device A (DA) and device B (DB). DB has paid for a better and quicker access to the network than DA. DB then is supposed to have better speed than DA. DA has paid for service A and DB has paid for service B. When packets from DA reaches MR, they are sent via the link that is associated to the type of service, service A in this case. Then packets reach TriaGnoSys ARv4 and some polices are consulted to determine to which ARv6 the packet has to be sent.

Looking to figure 4.9 there is a Mobile Network attached to a MR. In the Mobile Network there are different nodes that use different contexts, by different contexts we mean there are

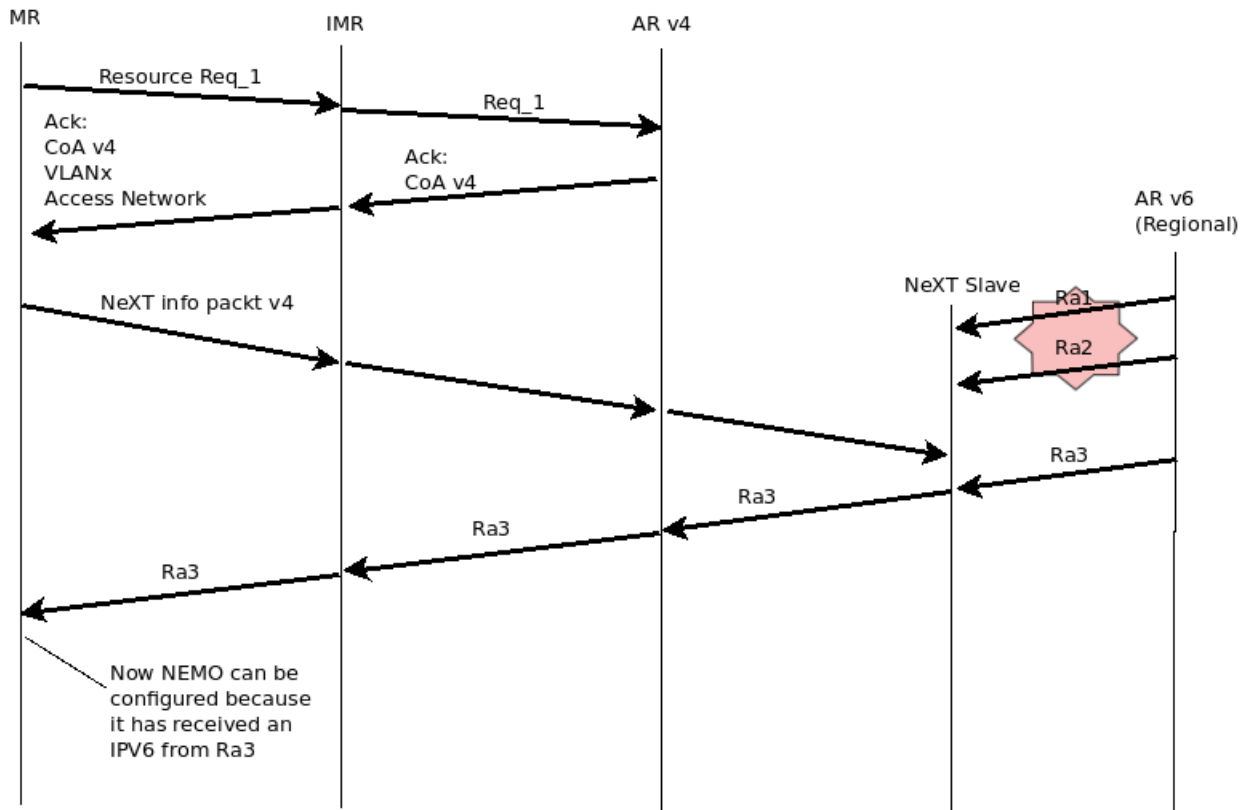


Figure 4.10: Flow packet

different needs of using one specific resource or another (L-band, Ku-band, WiMax2,...) depending on the requirements in flow traffic type. So in some how we depend of what IMR tells us, on which information it provides allowing to configure Policy Rules, NEMO, and NeXT Master in MR.

- NeXT Master needs to know the interface information to establish a binding socket channel with NeXT Slave, i.e. NeXT Master needs (1) a local interface on the MR with an IPv4 address (IPv4 CoA) of the respective IPv4 access network (can be virtual I/F), and (2) the IPv4 address of the respective NeXT slave. This channel is used to exchange signaling and data.
- Assuming this, it is supposed that IMR (an entity attached to MR that manages interface and traffic information transparent to MR) provides the information required. The following flow chart illustrate what is exchanged since MR request information resource to the IMR, and to when all is properly configured.

Figure 4.10 shows a flowchart that describes the steps taken to allow the MR receiving some router advertisements from a regional ARv6 which it has not any bound yet with. The steps are the following:

### 4.3. NeXT as option chosen

---

1. MR asks for information (makes a resource request “Resource Req 1”) to IMR
2. IMR issues according resource request to the respective access network (chosen by either MR or the IMR)
3. ARv4 (ground station) replies giving a CoAv4
  - CoAv4 can be public
  - CoAv4 can be private (NAT required in IMR or in access network)
4. With allocation of an CoAv4, the IMR needs to decide which virtual interface is set-up between IMR and MR (VLAN, ppp,...). CoAv4, VLANx, and AccessNetwork information is sent within the Ack to the MR.
5. MR receives the Ack from IMR
6. MR configures a CoAv4 and it configures a new interface. Now NeXTMaster (in MR) can establish a channel communication with NeXTSlave (in Regional ARv6).
7. NeXTMaster sends a packet with information to NeXTSlave using a src IPv4 address (the CoAv4) and a destination IPv4 address (the NeXTSlave IPv4 addr.).
8. NeXTSlave doesn’t know about NeXTMaster, but Master does (it knows AccessNetwork/Ground Station, thus public address v4 of NeXTSlave is deduced (it is likely that there is a NAT before the NeXTSlave that must be configured accordingly to pass packets from the NeXTMaster to the proper NeXTSlave)
9. MR link hasn’t got a CoAv6 yet, it is provided by Router Advertisements (Ra). However router advertisement 1 (Ra1) and router advertisement 2 (Ra2) can not reach MR till NeXTSlave knows about NeXTMaster.
10. Ra1 and Ra2 are discarded by the Arv6 because it doesn’t know what to do with them. After NeXTSlave knows about Master router advertisement 3 (Ra3) is sent.
11. MR receives the Ra3 and configures an CoAv6 (NEMO does this).
12. Afterwards MR configures IPv6 policing rules to route the respective traffic over the newly set-up (virtual) interface. The MR also need to transmit the applicable reverse policy routing rules to the HA.

Which NeXTSlave shall be used by the NeXTMaster depends on a-priori assumptions w.r.t. where the ARv6 is located in the ground network for a given access network, i.e. it should be configured and defined by the user. Example: If the assumption is that the ARv6 of the Ku-band access network lies in Region 1, then it must be configured in the NeXTMaster that for all CoAv4 belonging to the Ku-band access network the NeXTSlave belonging to the ARv6 in Region 1 is used. This ensures:

- (a) The CoAv6 address allocated to the respective MR interfaces are belonging to the respective ARv6, which ensures that also the return packets (CN -> MNN) are routed over the correct ARv6



(b) NeXT packets are sent from the NeXTMaster to the right NeXTSlave.

Further, there are several options that could be considered for the case when an access network and accordingly the resources already allocated on it become unavailable, however, it is out of the scope of my project considering them.

But why NeXTSlave is said to be placed in ARv6s? The reason is the following one. Apart from ARv6s, the only feasible candidate to think for hosting NeXTSlave is the HA. However, placing NeXTSlave in HA means that there is no possible way at all to route the packets through a specific region network area due to the lack of routing information. Thus turning into an impossible task to guarantee that a packet traverses a specific Region network. That is why by using ARv6 it can be provided within router advertisements prefixes addresses to configure an IPv6 CoA which solves the routing problem.



# Chapter 5

---

## Testbed Automation

---

In the previous chapter 3 the testbed architecture has been introduced taking NEWSKY and SANDRA scenarios into account. However, this testbed description in chapter 3 is more theoretical and conceptual rather than technical, thus I will make a description of the different test-bed's features. For instance, which software is running the testbed or which OS is hosting. This will be helpful to situate oneself when testbed automation features and decisions are mentioned in later paragraphs of this chapter.

### 5.1 IPv6 Network Mobility Testbed Architecture - Technical features

The IPv6 Network Mobility Test-bed consists of:

- 1 CN (Correspondent Node)
- 1 MNN (Mobile Node Network)
- 1 HA (Home Agent)
- 2 AR (Access Router)
- 1 MR (Mobile Router)
- 2 IPsec Security Gateways (ScGW)

## 5.1. IPv6 Network Mobility Testbed Architecture - Technical features

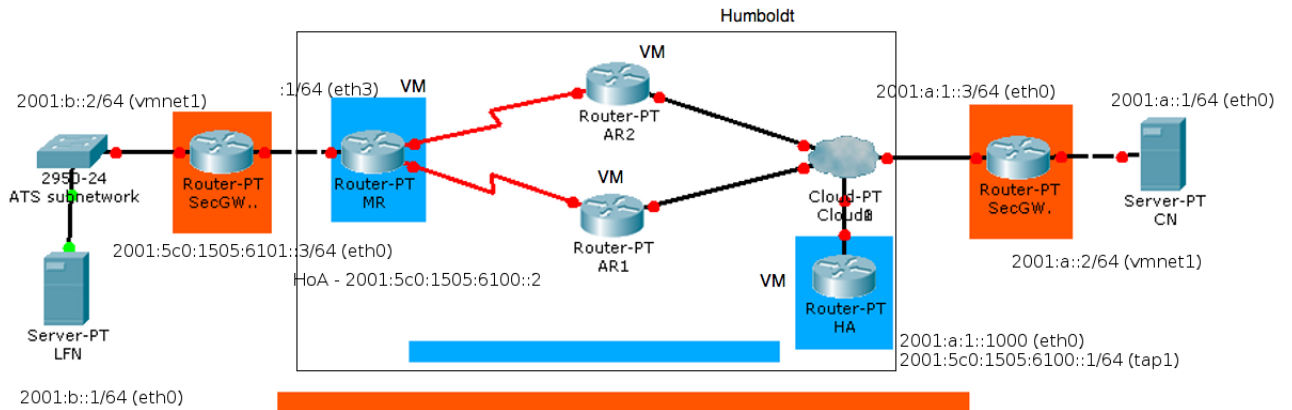


Figure 5.1: Testbed Architecture

The current architecture of IPv6 Network Mobility Test-bed is shown in Fig 5.1

Both HA, MR and the two AR's are Virtual Machines (VM) created with XEN. These four VM's are hosted in the same physical node inside the testbed. The name of its node is Humboldt. Virtual machines let test scenarios easily and they are also easy to move between physical machines. In a final instance they appear as independent machines to the software it is being run.

The MNN is a VMWare VM located in a node named ubuntu2. It represents either a whole network, or a device attached to a network which aims to send data through MR. The ubuntu2 node acts as a MNN Ipsec ScGW encrypting all the incoming traffic from MNN and decrypting the traffic coming from MR whose final destination is MNN. The IPsec functionality is out of the scope of this project, therefore it will mainly be mentioned when some aspects related with IPsec need some justification.

The CN is also a VMWare VM. The CN has a FTP server and an Apache web server installed, therefore it mainly acts as a service provider. CN is hosted in a node named Zuse which acts as a CN IPsec ScGW, encrypting all the incoming traffic from the CN and decrypting the traffic coming from HA whose final destination is CN.

All the nodes in the testbed use either Linux Red Hat Distribution, or Linux Ubuntu 8.04 hardy release. Its power in terms of CPU frequency and RAM are the typical of a Pentium 3 / Pentium 4 node, that means if a software shows good performance in the testbed (it consumes a few RAM and CPU resource) its deployment in more modern physical nodes will show and even more good performance. Thus efficiency and scalability of the software will be demonstrated.

The Xen hypervisor is a layer of software running directly on computer hardware replacing the operating system thereby allowing the computer hardware to run multiple guest operating systems concurrently. It provides support for x86, x86-64, Itanium, Power PC, and ARM processors. This allow Xen to run on a wide variety of computing devices. It currently supports

Linux, NetBSD, FreeBSD, Solaris, Windows, and other common operating systems as guests running on the hypervisor. Xen hypervisor is a free solution licensed under the GNU General Public License.

## 5.2 How the IPv6 Network Mobility Test-bed was switched on

As I have mentioned in section 5.1 the IPv6 Network Mobility Test-bed consists of three physical nodes, Zuse, Ubuntu2 and Humboldt. This last one is the one that hosts MR, HA, AR1 and AR2 VM's. Zuse and Ubuntu2 acts as a ScGW and each one hosts its VM's, CN and MNN.

The IPv6 Network Mobility Test-bed runs mainly four protocols, Mobile IPv6, Network Mobility (NEMO), a TriaGnoSys protocol called NeXT and IPsec.

Due to the deployment of these protocols and other functionalities to reach a certain final simulation performance behavior there were some configuration commands and some scripts that required manually execution. There were also some configuration commands and scripts that were used in such a way that either in restart, or boot mode they were launched automatically. The way of making them launch automatically was specifying them in the `/etc/rc.local` Linux script.

Rc.local is a standard script in Linux distributions, which means it is common in almost all the Linux distribution releases that nowadays exist. This script is executed after all the other init scripts, so all the initialization lines can be put in there, this means whatever daemon that is initialized in the boot step will be started before the rc.local. Once rc.local is called it is put in the execution background.

Humboldt node use `/etc/rc.local` to configure some internal things. VM have been decided to be run manually via a script and also from a Graphical User Interface.

A node can go down for many reasons:

- An outage
- A manual reboot of the node
- A non desirable halt ( Eg. When an execution by accident of a countdown switch off)
- The node gets broken and it has to be replaced

For all those reasons, this way of proceeding for the restart of whether a node or all the nodes<sup>1</sup> lead into a scenario where the person who had to restart some nodes had to spent some time on it. He had to switch on the node, start typing commands and launching scripts.

Avoid spending time on things that can be skipped is something valuable in an investigation environment, thus the way of how the testbed was launched was not convenient.

---

<sup>1</sup>From now on when I say "all the nodes" I will be referring to both physical nodes and VM's, in case of talking about only VM's or physical nodes these last two terms will be employed for that purpose

## 5.3 Considered options for the automation

A new way for automation had to be developed. It came out that one of the new requirements had to minimize the human interaction in the configuration steps and by just pressing or rebooting a node, this one configured itself. Therefore the user interaction with the testbed reduced to what mattered, investigation and research.

For that reason I thought which would be the feasible and optimal way of doing a better startup automation, three options were considered. Those options will be explained in details in the following subsections 5.3.1 5.3.2 and 5.3.3

### 5.3.1 Use of Cron + Bash scripts:

Cron is a time-based job scheduler in Unix-like computer operating systems. The name cron comes from the word "chronos", Greek for "time". Cron enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance or administration, though its general-purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading files, for instance using wget or curl commands.

Cron makes use of what it is called *crontab table*, a system file that contains the command or scripts to be periodically launched given a scheduling configuration. Each user can have its own table, making not possible to other users to modify it, the root user has its own table too.

The crontab table can have various crontab entries, each entry has five fields for specifying day, date and time followed by the command to be run at that interval. However those fields can be replaced by an equivalent namespace or typo, for instance to deploy a task every month it can be specified by typing '@monthly' in the scheduling task field.

The following figure from below 5.2 describes the syntax of a crontab entry, showing the 5 scheduling fields and also the command field.

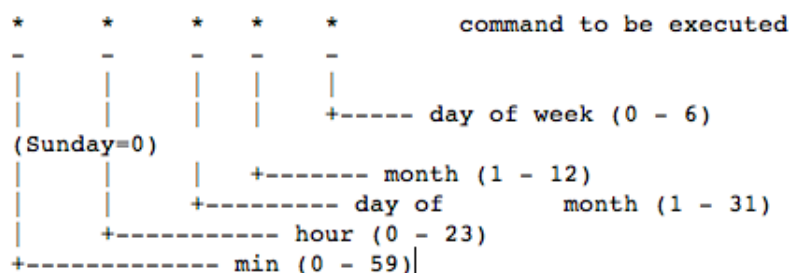


Figure 5.2: Cron table entry format

However as it has been mentioned there are some special predefined values which can be used to replace the Cron scheduling expression.

The asterisk in fields indicates to cron that any value is acceptable.

Entry	Description	Equivalent To
@yearly (or @annually)	Run once a year	0 0 1 1 *
@monthly	Run once a month	0 0 1 * *
@weekly	Run once a week	0 0 * * 0
@daily (or @midnight)	Run once a day	0 0 * * *
@hourly	Run once an hour	0 * * * *
@reboot	Run at startup	

Figure 5.3: Cron predefined variables

The intention of the automation is to make hardly all the existent configuration processes start automatically in the startup, therefore by looking in the figure 5.3 it can be deduced that the appealing feature of Cron for the test-bed automation is the predefined variable *@reboot*. @reboot can be useful if there is a need to start up a server or daemon under a particular user, and the user does not have access to configure init to start the program. Tasks launched by Cron inherits the permissions and ownership of the user who have configured the crontable and is assigned to it.

In all testbed nodes there are some configuration scripts files, those scripts determine the node behavior. All those configuration scripts are backed up in a *Subversion repository* enabling a quickly reestablishment of the them in a new node in case of a node failure.

By considering Cron as a valid solution it implies that every configuration script in every node has to be manually added into the *crontab file* using an editor. This means that if for some reason the table's content empties then entries have to be added again. Besides An extension of cron is the ability to specify a time as "@reboot", this means there are some versions of cron that don't support it. It can be considered as a drawback, and it has to be said that the intention is to work with standards.

### 5.3.2 Use of /etc/rc.local + Bash scripts:

In section 5.2 a brief description of /etc/rc.local has been done. As it has been explained rc.local is an standard way of launching commands and processes when a node starts up. As /etc/rc.local script is the last one in being launched it can be guaranteed that all system environment is properly started, for instance the system network environment.

In rc.local a list of either commands or processes can be specified in a decreasing cascade order, that means the first lines are the first ones in being launched. For each command line in rc.local file a shell is created in the background for its execution (normally a bash shell, it can depend of configurations), however, unless you do not specify the command to be launched in the background (specifying with *&* symbol), rc.local will deploy the command execution and wait for a response to continue interpreting the following lines.

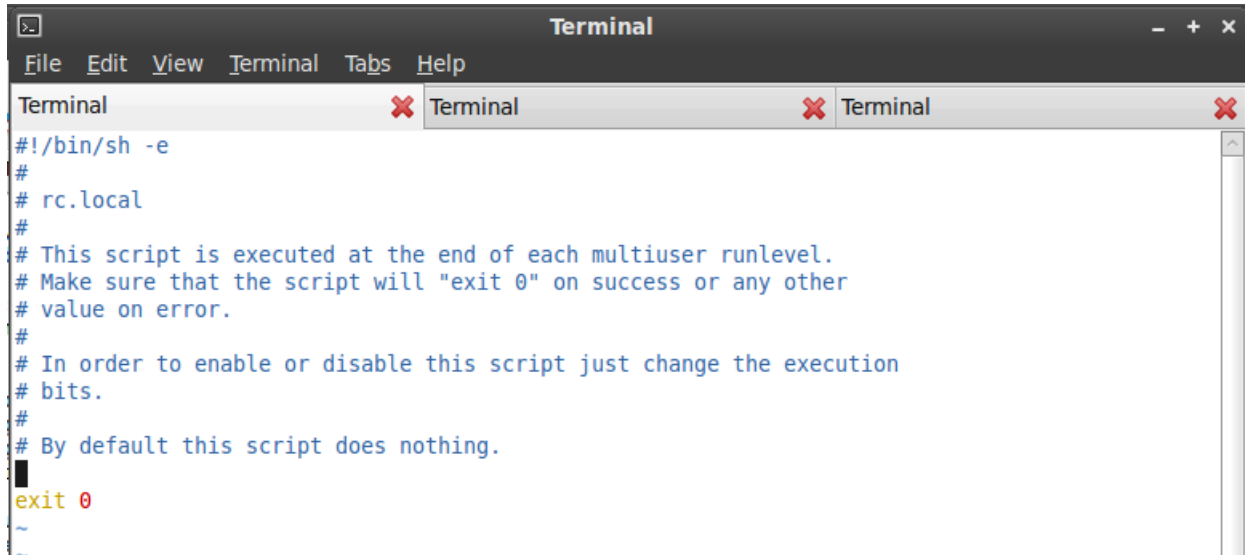
Choosing whether launching a script or command in the background or in a established sequence depends of the behavior to be expected, like Command in Line 2 needs that Command

### 5.3. Considered options for the automation

---

in Line 1 had previously configured some behavior. It can be said a matter of prerequisites and requirements.

A typical `/etc/rc.local` structure is the following one from below:



```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
exit 0
```

Figure 5.4: Rc.local file

The file starts with a shebang line, the one that says “`#!/bin/sh -e` “. In computing, a shebang line is the character sequence consisting of the characters number sign and exclamation point (`#!`), when it occurs as the first two characters in the first line of a text file. In this case, the program loader in Unix-like operating systems parses the rest of the first line as an interpreter directive and invokes the program specified after the character sequence with any command line options specified as parameters.

After the command lines the script terminates with the `exit` command with a 0, saying it successfully exits from the script.

As it has been mentioned, testbed nodes have various configuration scripts, some of them can be specified in the `/etc/rc.local` file, but there are others that are better not to because depending on which scenario it is intended to perform a simulation there will be some configuration scripts that will be load and others that won't. Besides, there are scripts that needs input values that define for example packet delay, interfaces and other parameters.

That means there is the need of human interaction to in a final instance configure the testbed for a scenario that requires an specific behavior.

It can be said that there are mainly two scenarios, each one with its correspondent scripts and configuration files. These two scenarios are:

- Simulating a satellite link using `tc` command, adding delay to the packets.
- Using BGAN satellite link.



The steps to be followed in order to perform the automation process are:

- Some configuration files are added to the `/etc/rc.local`
- Once the node is either rebooted or started up the configuration files in `/etc/rc.local` are sequentially launched.
- The user executes a bash script from the command line

This last point of the list, "*executing a bash script from the command line*" is the step in which there is human interaction with the system in terms of configuration actions. An example of a scenario simulation base case will be now introduced.

Base case:

Given an scenario using Satellite emulation link, scripts A, B, C and D are defined in the `/etc/rc.local`. All the testbed nodes are rebooted. Then afterwards launching a script, it doesn't matter if it is whether from local computer or Humboldt. This script will make a series of question asking for specific behavior matters, such as, "¿Do you want to apply packet filtering?"; if the answer is affirmative then some parameters to the packet filtering are suggested for the system. Afterwards the user is able to choose whether accept the suggestion or define its owns parameters. This series of questions establish a specific configuration for the whole testbed behavior.

Once the user has answered all the questions, a list of all the options previously chosen is shown and the user is asked to confirm if all the parameters and options are okay. In case of selecting "No" the user has the option to modify the parameters and options he wants.

Until this step there has been no orders to any of the testbed nodes, what has been done is to store the options chosen by the user in variables. That means that only after the user confirmation remote commands using ssh are sent to the different nodes of the testbed to successfully launch the scripts that determine a behavior.

### 5.3.3 Use of `/etc/rc.local` + Java GUI + Bash scripts

The third option considered for the testbed automation is the implementation of a Java Graphic User Interface (GUI). This Java GUI checks the node's status under request and manages the start, reboot, and shutdown of them either one per one or by various at same time.

In sections 5.2 and 5.3.2 `/etc/rc.local` has been widely explained so its features and whereabouts will not be discussed again in this section.

What this option offers as a new feature is the use of a Graphic Interface to manage all the nodes (start, reboot, shutdown), this means that to a user non familiar with shell environments it becomes easier to manage nodes because of the GUI, with just one mouse click the nodes are controlled.

The main difference between this option and the one in section 5.3.2 is the absence of a script asking the user input data in order to determine the final behavior. Instead of that there is a GUI. This option aims to provide a better user interface usage.

## 5.4. Final option: Use of /etc/rc.local + Java GUI + Bash scripts

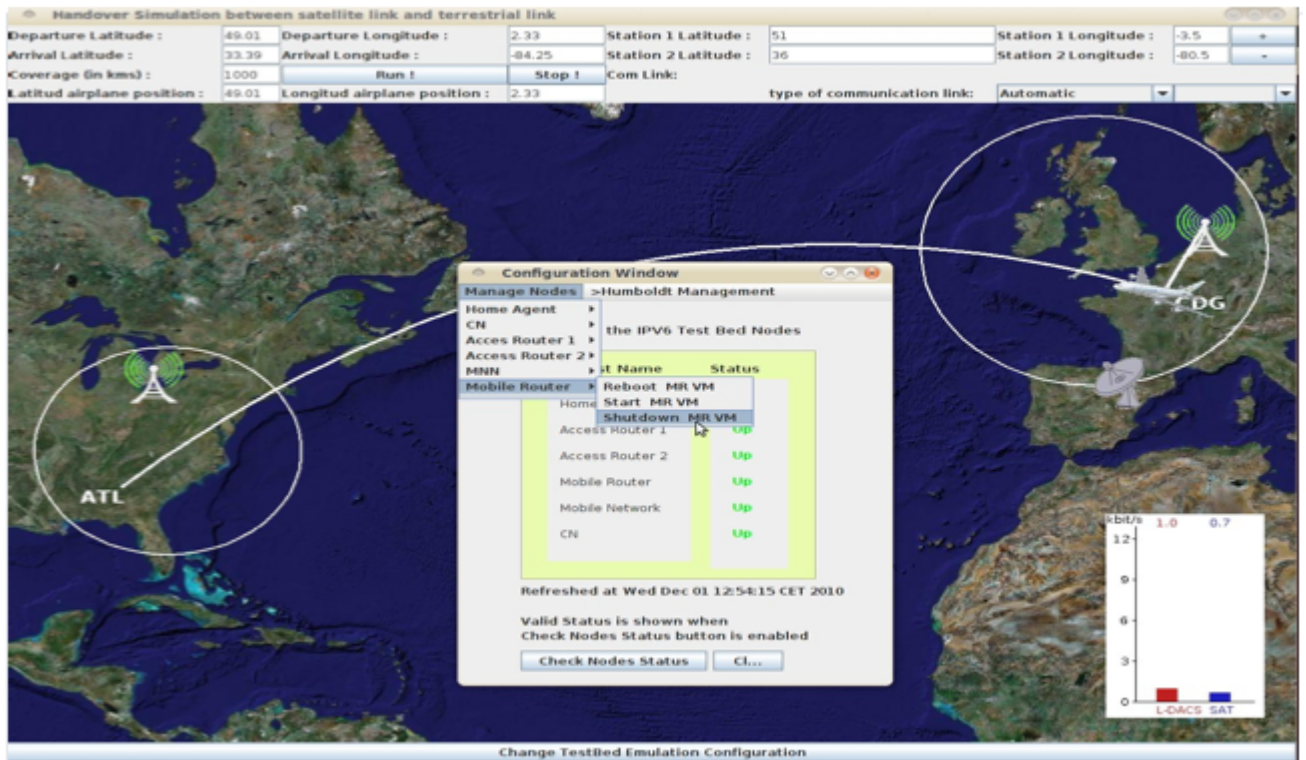


Figure 5.5: Java GUI

## 5.4 Final option: Use of /etc/rc.local + Java GUI + Bash scripts

Three possible options have been described as a solution for the IPv6 Network Mobility Testbed.

1. Use of Cron + Bash scripts
2. Use of /etc/rc.local + Bash scripts
3. Use of /etc/rc.local + Java GUI + Bash scripts

One of the requisites was that the usage of the testbed configuration, switching either physical nodes or VM's on had not to be a hard task. For that reason the use of a Java GUI (*third solution*) has been decided as the best option for those purposes.

### 5.4.1 Different features of the Java GUI

In section 5.3.2 it was said there were two main different scenarios behaviors, "*Simulating a satellite link and "BGAN satellite link"*". It was mentioned that the testbed uses some XEN VMs, depending of which scenario behavior has to be tested those XEN VMs will need a different

configuration file (.cfg) to be loaded. In this .cfg file there are all the information of the VM, the OS image (img) it uses, which interfaces has to be configured, MAC addresses to ensure they are unique and there is no conflict, also which bridges are configured and other things.

However, as the NEWSKY project was going to finish within some weeks and the testbed architecture was going to change in a close future it was finally decided not to devote some many time in that. Therefore some decisions were taken. One of them was the decision of letting as preconfigured the Satellite emulated link behavior, that means the VMs starts automatically with the correspondent .cfg file, in the GUI there is no option to change that, however in a future extension and adaptation of the GUI to the new SANDRA testbed architecture it could be done.

A node is supposed to be up if it fulfills the following requirements:

1. It is reachable through the network (3 ICMP packets are sent using `ping -c 3`)
2. All the processes within a list are running in the node

The flow from below, figure 5.6 exemplifies how the status of a node is checked.

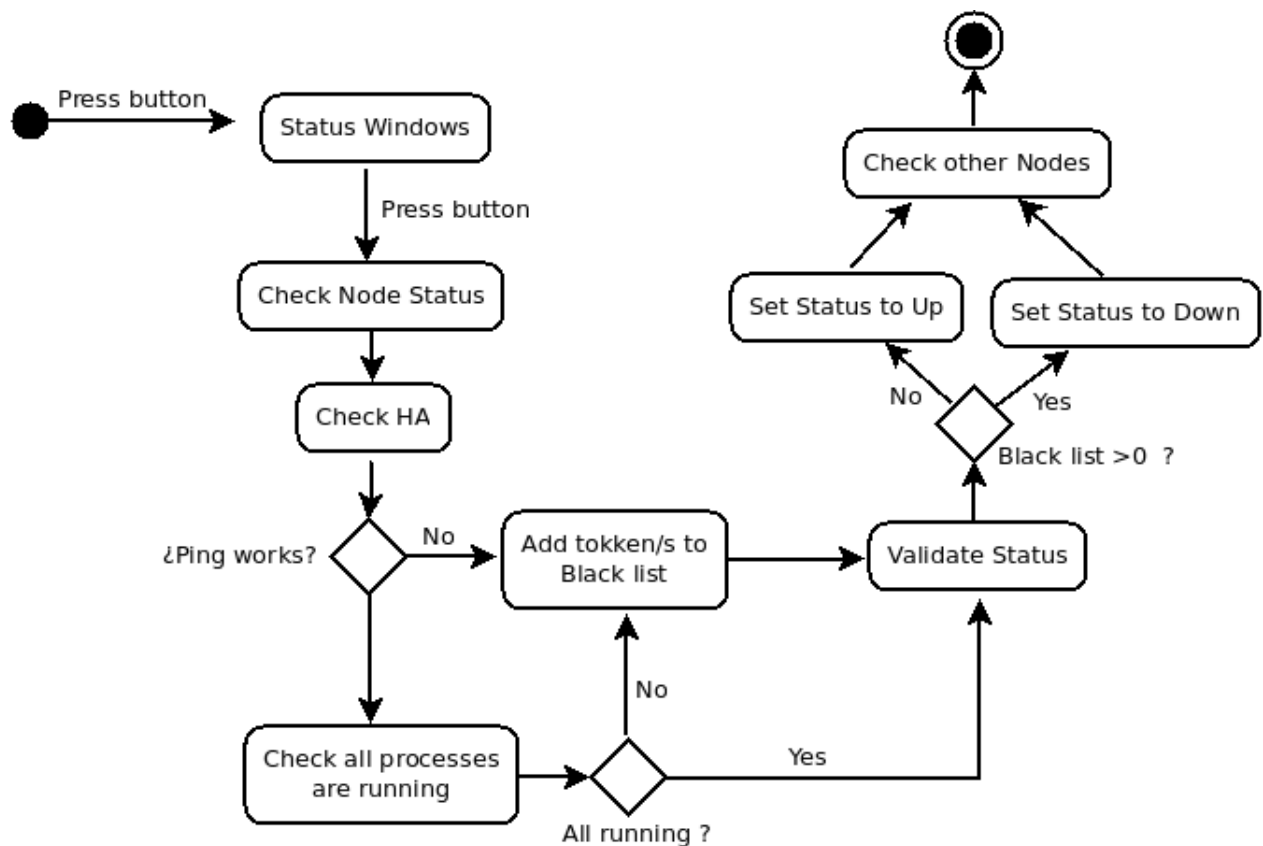


Figure 5.6: Flow Checking Nodes Status

To check if whether a process is running or not a ssh connection is deployed and afterwards a Unix command is executed. This Unix command tries to identify a nameprocess in the remote

#### 5.4. Final option: Use of /etc/rc.local + Java GUI + Bash scripts

---

node, the name of the process is the one that is extracted from `lprocess.get(i)`. The line from below describes what has been explained in this paragraph.

```
ssh root@ha.newsky ps -e | grep " + lprocess.get(i) + " | tr -s "/" "/" | cut -d "/" -f5
```

The following flow chart figure 5.7 shows the flow when a start action is called from the GUI. Bear in mind that in the *Create HA VM* box in the inner code it is established to use Satellite Link Emulation configure file (.cfg). In a future extension it can be programmed a functionality to change among this and the other.

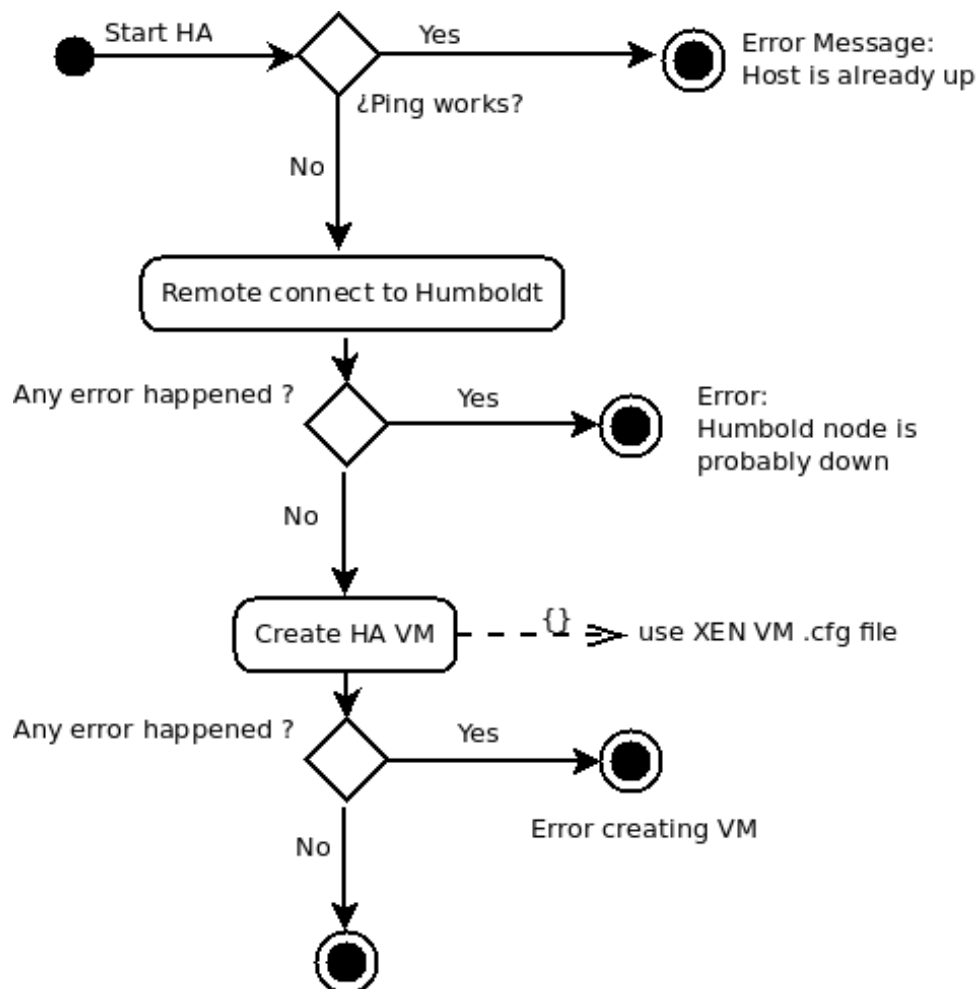


Figure 5.7: Flow Start Node

The following flow chart figure 5.8 shows the flow when a stop action is called from the GUI. In the box "*Shutdown XEN ID VM*" a shutting down action is performed in Humboldt node.

Humboldt node allocates different VMs in its domain, so what is done in the "*Shutdown XEN ID VM*" box is once the identifier of the VM within Humboldt is captured it is used to identify the VM to be stopped. The command to stop the XEN VM is the one in figure 5.9.

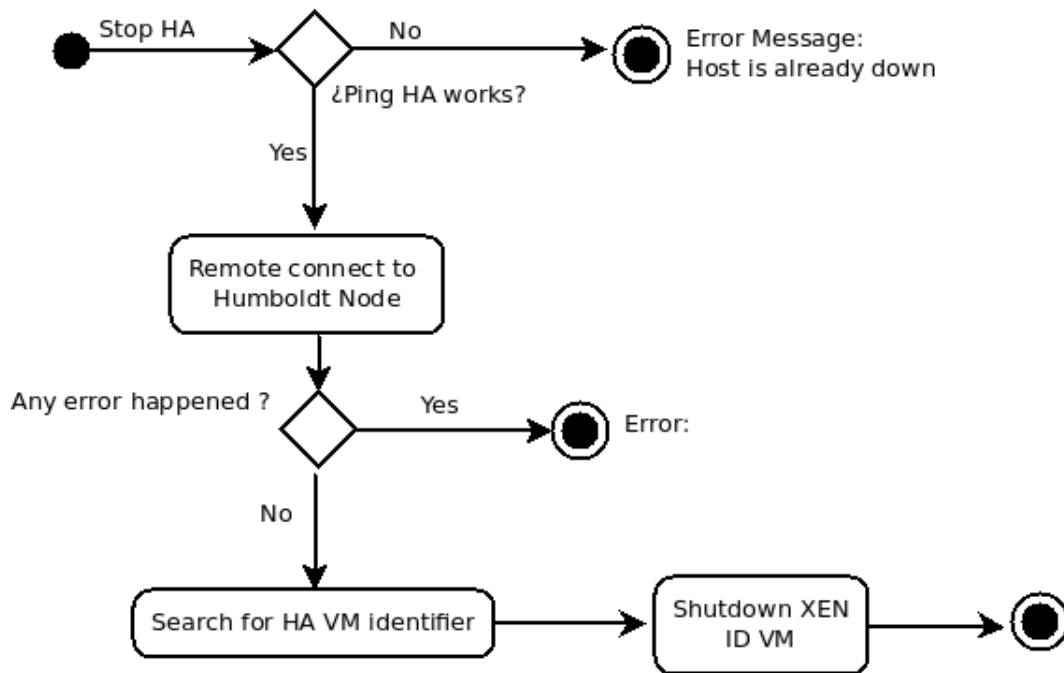


Figure 5.8: Flow Stop Node

```

InputStream is;

process = Runtime.getRuntime().exec("ssh root@humboldt.newsky xm li | grep homeagent | tr -s \" \" | cut -d \" \" -f2");
is = process.getInputStream();

process = Runtime.getRuntime().exec("ssh root@humboldt.newsky xm shutdown "+ convertStreamToString(is));
is = process.getInputStream();
  
```

Figure 5.9: Stop VM command

The flow chart from figure 5.10 exemplifies the case when a node is being requested to be rebooted. If the node is not available because it is down then it can not be rebooted, otherwise the rebooting process is followed

#### 5.4.2 Where are the scripts located and how are they executed ?

Hardly all the scripts in a node are located in a specific path folder directory, this guarantees the scripts to be stored in the subversion, making it possible to easily reestablish all the scripts files in case of a major disaster. It can be said that they are all centralized in a directory instead of being spread along the node path directory tree.

In section 5.3.3 it has been explained that scripts are previously predefined in the `/etc/rc.local` script file. Two things have to be considered, one is that the actual configuration takes into account that as a preconfigured configuration is being used then the required scripts are defined in

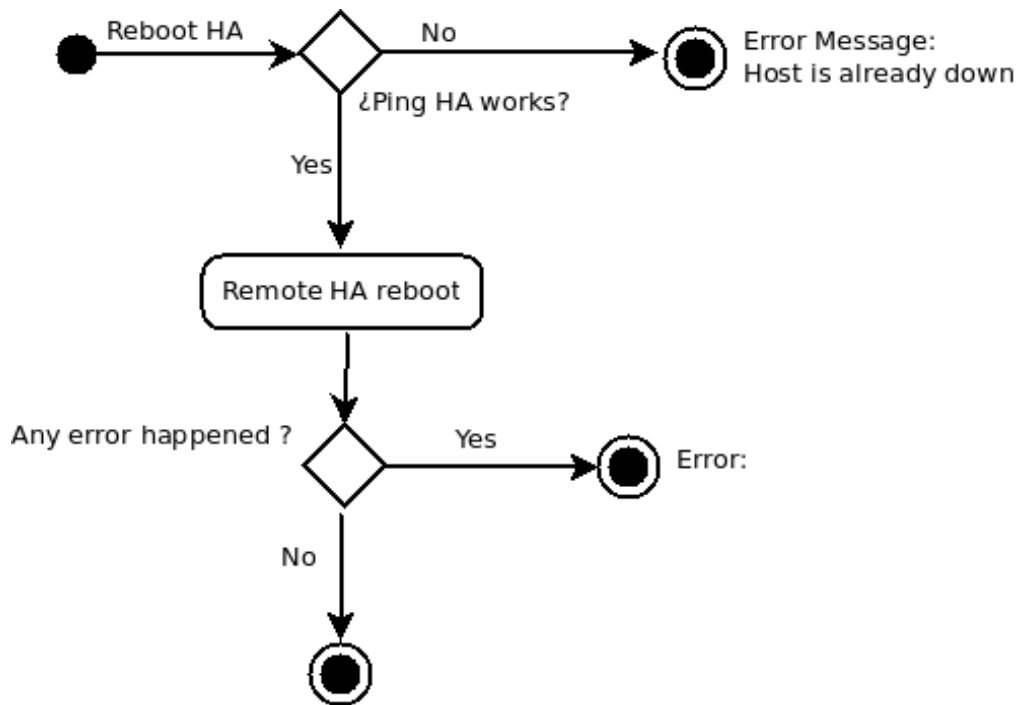


Figure 5.10: Flow Reboot Node

the /etc/rc.local. The other thing to take into account is that it is easily feasible to extend and modify the code to make that instead of defining some scripts in the /etc/rc.local they finally become executed by the GUI application by evoking them remotely.

The Java GUI app can be executed from outside the testbed, there is no need of launching it from a testbed node. However some previous steps are needed to be followed. Those steps are mainly configuring a reliable ssh key intercommunication channel. It is achieved by creating an own ssh key *certificate*<sup>8</sup> copy it to the testbed nodes the computer will connect and also copy the ssh key-certificates from the testbed nodes to the local computer.

Another step that needs to be done is copy the java GUI files from the subversion trunk or from one of the testbed nodes that has the java files in there.

# Chapter 6

---

## NeXT software architecture/design

---

In this chapter some aspects of NeXT first version (the previous one to my project) design will be explained for a better comprehension. Which changes have to be introduced into NeXT first version in order to adapt it to the new SANDRA scenario will be also explained. And finally, two different NeXT new design approaches will be described.

### 6.1 NeXT first protocol version design

As it has been mentioned in section 4.3 in a previous chapter, NeXT first code was thought for a one-to-one communication, it means NeXTMaster has only communication exchange with one NeXTSlave and viceversa. NeXTMaster only sees NeXTSlave and this last one only sees NeXTMaster.

Figure 6.1 shows an example of NeXT deployment in a typical topology network in real world.

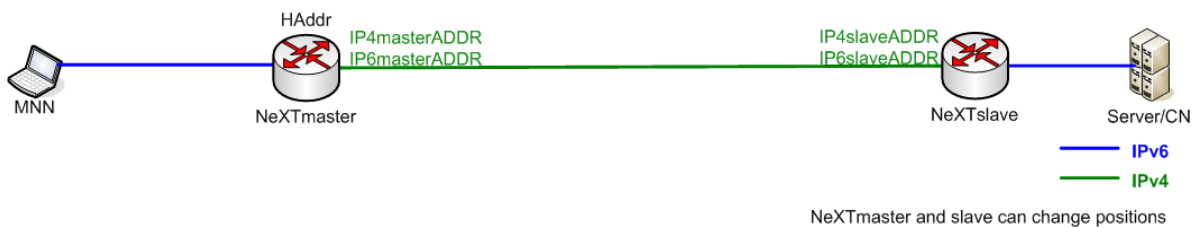


Figure 6.1: NeXT real world architecture

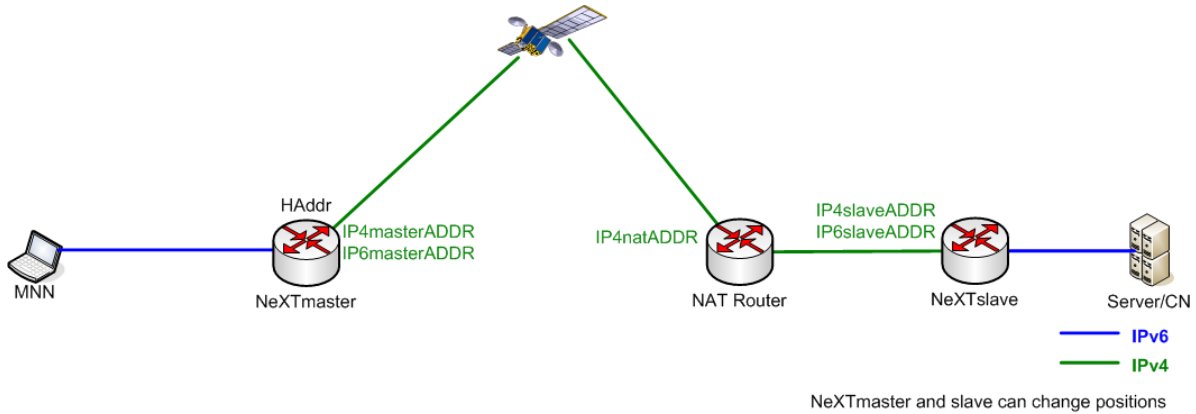


Figure 6.2: NeXT real world using satellite link

Figure 6.2 shows a scenario using a satellite link within TriaGnoSys GmbH company network, using an AR with NAT as an entry gateway to the company network.

NeXT uses some resources in order to make the translation from IPv6 packet to an IPv4 packet and the way round. These resources are related with processes identifiers, addresses information and also some data structures that holds the information required to have an IPv6-IPv4 mapping as commented in section 4.1.3.

NeXT64(either NeXT64\_Master or NeXT64\_Slave) and NeXT46 (either NeXT64\_Slave or NeXT46\_Slave) holds its own data structure which is in charge of keeping the information required for a packet translation (one way IPv6 to IPv4 for NeXT64 and the other way, IPv4 to IPv6 for NeXT46).

Apart from that NeXT uses *SIGNALS*<sup>9</sup> to let NeXT46 entity and NeXT64 entity exchange information among them. Moreover SIGNALS are used for a proper termination of NeXT protocol under demand, when an order of termination is sent to NeXTMaster this one sends a notification to its NeXTSlave interlocutor saying that it is going to go down until new notification. NeXTSlave then cleans its context related with its NeXTMaster interlocutor.

Some flows, will be shown to give a glance of how NeXT works, but just as example given. Please see figures 6.3 and 6.4

## 6.2 New NeXT requirements in SANDRA scenario

As mentioned in section 4.3 NeXT has been chosen as the option to be deployed in SANDRA scenario. As a result of this it can be seen in figure 4.9 how does NeXT look like in this new scenario. Things to point out, assuming that only one MR is being considered, are:

1. There are multiple NeXTMaster instances in the same MR
2. There is one NeXTSlave in each Region Network Area (see figure 4.9)



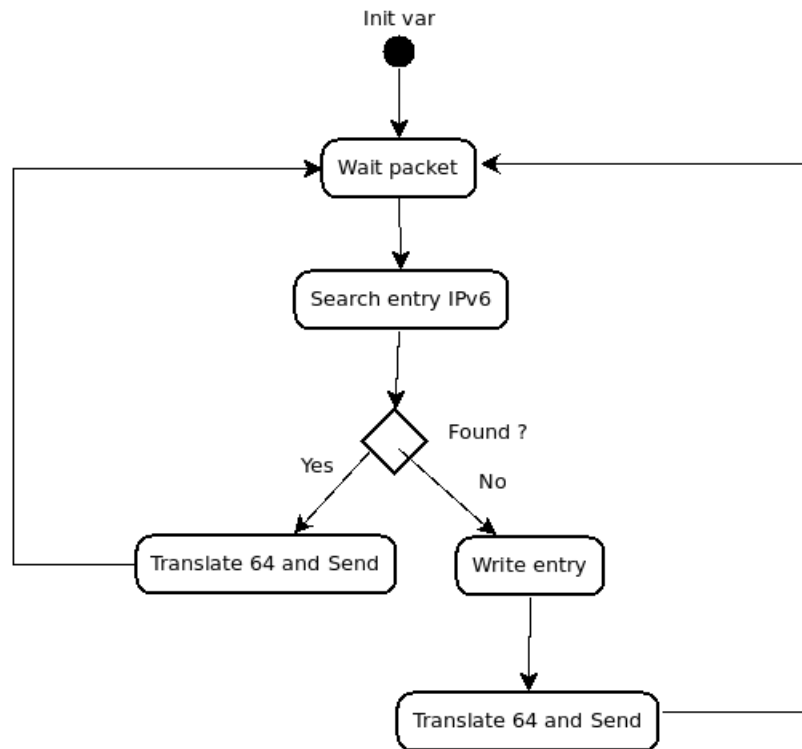


Figure 6.3: NeXT64 Master flow

What do these two last points mean? It means that the way NeXT worked before in its first release version is no longer valid anymore due to the existence of multiple NeXTMaster instances in MR. In section 6.1 it has been said that NeXT was originally thought of as a one-to-one protocol, this has some coding implications in terms of NeXT own resources management and NeXT inter-entities communication (NeXT46-NeXT64 communication flow).

As an example given, if NeXT46 wanted to communicate some information to NeXT64 it looked in the Linux process list by using the command *ps* piped with other commands, and it searched for a process called "NeXT64\_Master". This proceeding didn't represent a problem of behavior in a one-to-one mode operation, however, as MR has turned into an entity with multiple NeXTMaster instances the proceeding previously explained would be problematic and would lead to a non-stable behavior and performance. For instance, for a better comprehension in the following example it is assumed that MR has two NeXTMaster instances (2 x NeXT64Master and 2 x NeXT46Master), let's call them:

- NeXT64Master\_1
- NeXT64Master\_2
- NeXT46Master\_1
- NeXT46Master\_2

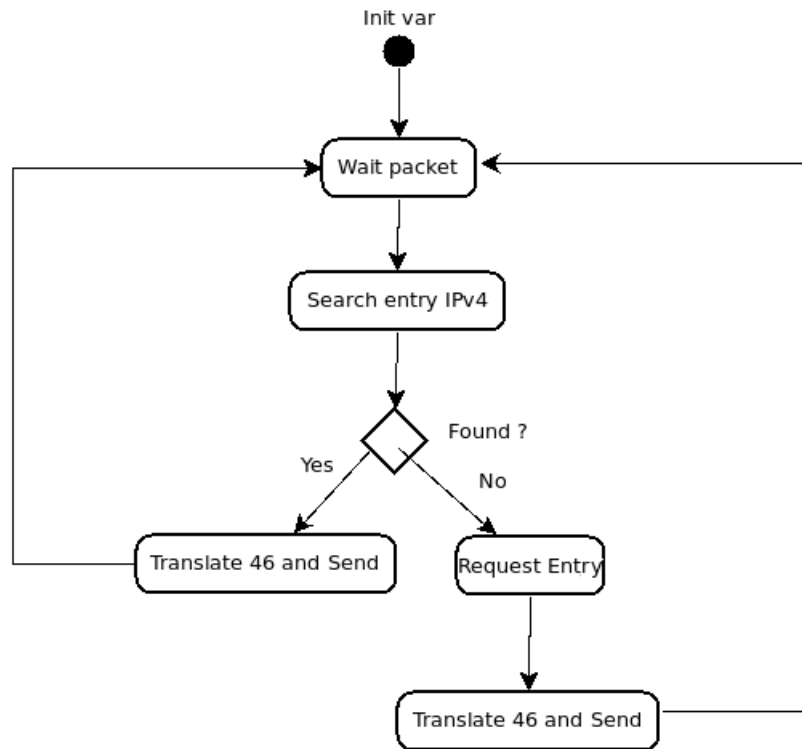


Figure 6.4: NeXT46 Master flow

Imagine that NeXT46Master\_1 wants to send a SIGNAL to its NeXT64Master, NeXT64Master\_1 in this case. If the original way of sending signals (searching a process with name “NeXT64\_Master”) is used then there is no 100% guarantee that the SIGNAL will reach NeXT64Master\_1, because there will be two processes in Linux with the same name, so the behavior will be random, either NeXT64Master\_1 or NeXT64Master\_2 will received the SIGNAL.

This last example given can be applied to some other resources within NeXT code during the process of NeXT adaptation in SANDRA scenario and its new requirements.

To sum up this section, it has to be said that the main and new requirement to take into account is the existence of multiple NeXTMaster instances in MR. This introduces the management of multiple connections with different NeXTSlaves placed in different region network areas and it also introduces the fact that multiple resources from different NeXTMaster instances will be held in MR.

### 6.3 Two different NeXT design approaches

The starting point for a first reasoning was the necessity of MR to host multiple NeXTMasters, one NeXTMaster per each NeXTSlave placed in one region area network.

What would happen if there would be only one instance of NeXTMaster? NeXTMaster would be listening only to one queue giving the fact that it is only allowed to listening into one

queue. Thus, it is likely that some policy routing would need to be replicated in order to know to which interface and thus CoAv4 address packets are routed to. This option didn't was not easy going in terms of internal NeXT code interface configuration and it was not as intuitive and simpler as the options considered in sections 6.3.1 and 6.5.

Apart from this last one considered option two other approaches were thought, they are explained in the following sections 6.3.1 and 6.5. The first intention and, after analyze all the new requirements, was to develop a design for NeXT using threads. However, I was not sure at all if it would be possible due to technology reasons. Programming with threads usually introduces a factor of complexity at coding, shared memory is one positive feature of threads and also an advantage but it is sometimes tricky. For these reasons I thought that it would be appealing and very appropriate to think of a previous design to NeXT threads, a design as good as this last one that serve as an intermediate solution. A design that allows leading to a NeXT threads design without many modifications and in case of non success allows a rollback to the previous design solution (6.3.1).

These two approaches have been developed and implemented. NeXT threads implementation was nearly 100%accomplished. Shared memory problems were dealt and overcame, but there was an issue that made the NeXT protocol having a non correct behavior, however the issue was not clear enough to determine its cause. It is explained in section 6.6.

### 6.3.1 NeXT multiprocess

¿How many instances of NeXTMaster can there be in MR? The answer to this question in this approach is:

One NeXTMaster instance process per each MasterY - SlaveY communication pair. (See section 6.5)

As it has been mentioned in section 4.1.3 NeXT protocol uses libnetfilter\_queue library, NeXT is a protocol that grab packets and makes modifications to them. In terms of design and implementation it means both NeXT64 and NeXT46 entities are listening to a specific kernel queue which holds packets, the process for storing one packet in a specific kernel queue (nfqueue) is made by using iptables and ip6tables.

Those two lines are an example of how packets are redirected to a specific nfqueue. QUEUE\_MAST64 is a defined variable that holds a value extracted from a xml configuration file, it is an int value specifying the nfqueue number identifier.

- `a = "iptables -A OUTPUT -o "+IFACE4+" -j NFQUEUE --queue-num "+QUEUE_MAST64;`
- `system(a.c_str());`

**-j NFQUEUE --queue-num** is the sintaxis used in ip6tables to specify the nfqueue number.

Adding this libnetfilter\_queue library feature in the figure 6.5 turns it into figure 6.6. In figure 6.6 it can be seen that each NeXTMaster instance (NeXT64Master and NeXT46Master)

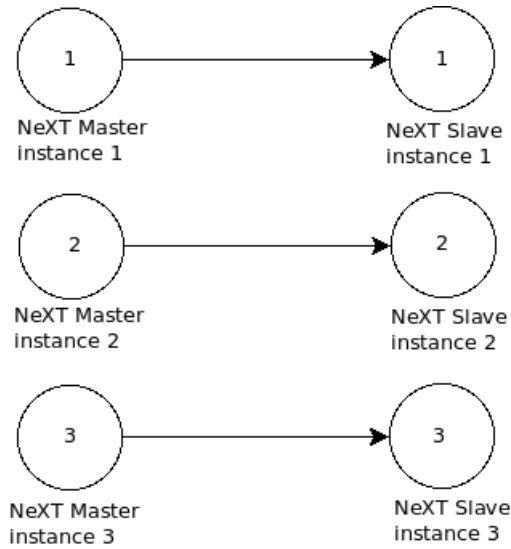


Figure 6.5: Multiple instances

listens to a different nfqueue, each entity is only permitted to listen to only one nfqueue due to technology restrictions. One nfqueue is used for NeXT64Master and another for NeXT46Master. Using multiple instances allow to do that. Of course what it has been explained for NeXTMaster entity is equally valid to NeXTSlave. Doing in that way it can be guaranteed that all NeXT instances will grab the correct packets, thus preventing from any possible problem on sending and forwarding packets to the right destination.

Identifier numbers of nfqueue in NeXTSlave in figure 6.6 can be perfectly numbered from 1 due to the fact that as they are usually in different nodes that hosts NeXTMaster instances, there is no possible conflict. However to make it more clear they have been numbered from 7. The concept of multiple NeXTMaster instances has been introduced, multiple launches of NeXTMaster (multiple processes).

In order to turn NeXT into a multiprocess protocol a new architecture design was developed, taking the need of allowing an easy transition to threads design into account.

In figure 6.7 there is a main process named “NeXT“, NeXT main process in the picture, this is the launcher of the new NeXT version protocol, a binary file. NeXT main process creates as many child processes as ARs in Regional Network Area the MR has to deal with. This figure takes as an example the launching of only NeXTMaster entities, various NeXTSlave entities can also be launched. NeXTMaster entities are used in the following explanations, but keep in mind that NeXTSlave could also be used NeXTSlave instead.

Each child process is in charge of launching its NeXT64 and NeXT46 entity, bear in mind that each child process corresponds to a specific NeXTMaster entity bound to a certain regional AR with a NeXTSlave entity. NeXTMaster\_1 entity is bound to NeXTSlave\_1 entity in regional AR\_1, there can not be any NeXTMaster entity apart from NeXTMaster\_1 bound to NeXTSlave\_1.

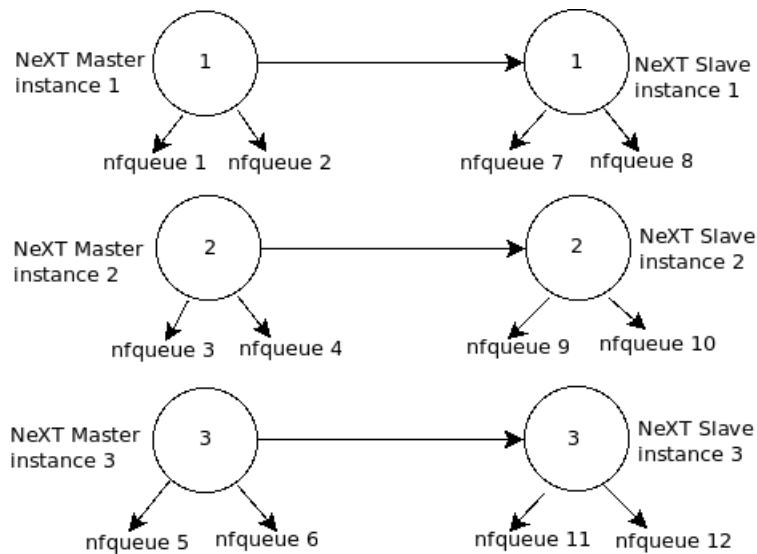


Figure 6.6: Multiple instances with libnetfilter

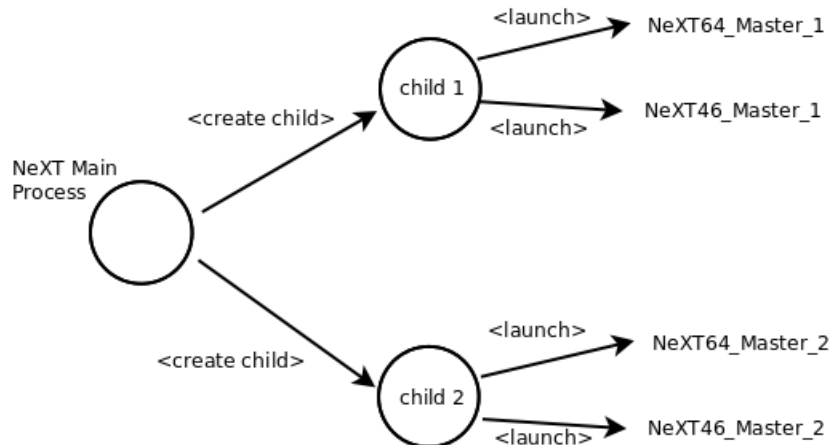


Figure 6.7: NeXT multiprocessing architecture

Once NeXT Main process finishes to create its child processes it terminates itself because there is no longer necessity of its use. When child processes finish launching their NeXT entities they terminates themselves too, like their parent. Why then creating one child process for each AR to be bound? The answer is simple, I have said that the intention was to develop a design architecture solution that allows an easy transition to a NeXT design architecture using threads. It will be explained in more details in section 6.5 but the main reason to create a child process to launch a pair of NeXT64 and NeXT46 is to create a dedicated memory space to this two processes instead of sharing the memory space with others NeXT instances.

Some technical NeXT whereabouts will be introduced in later paragraphs within this section,

### 6.3. Two different NeXT design approaches

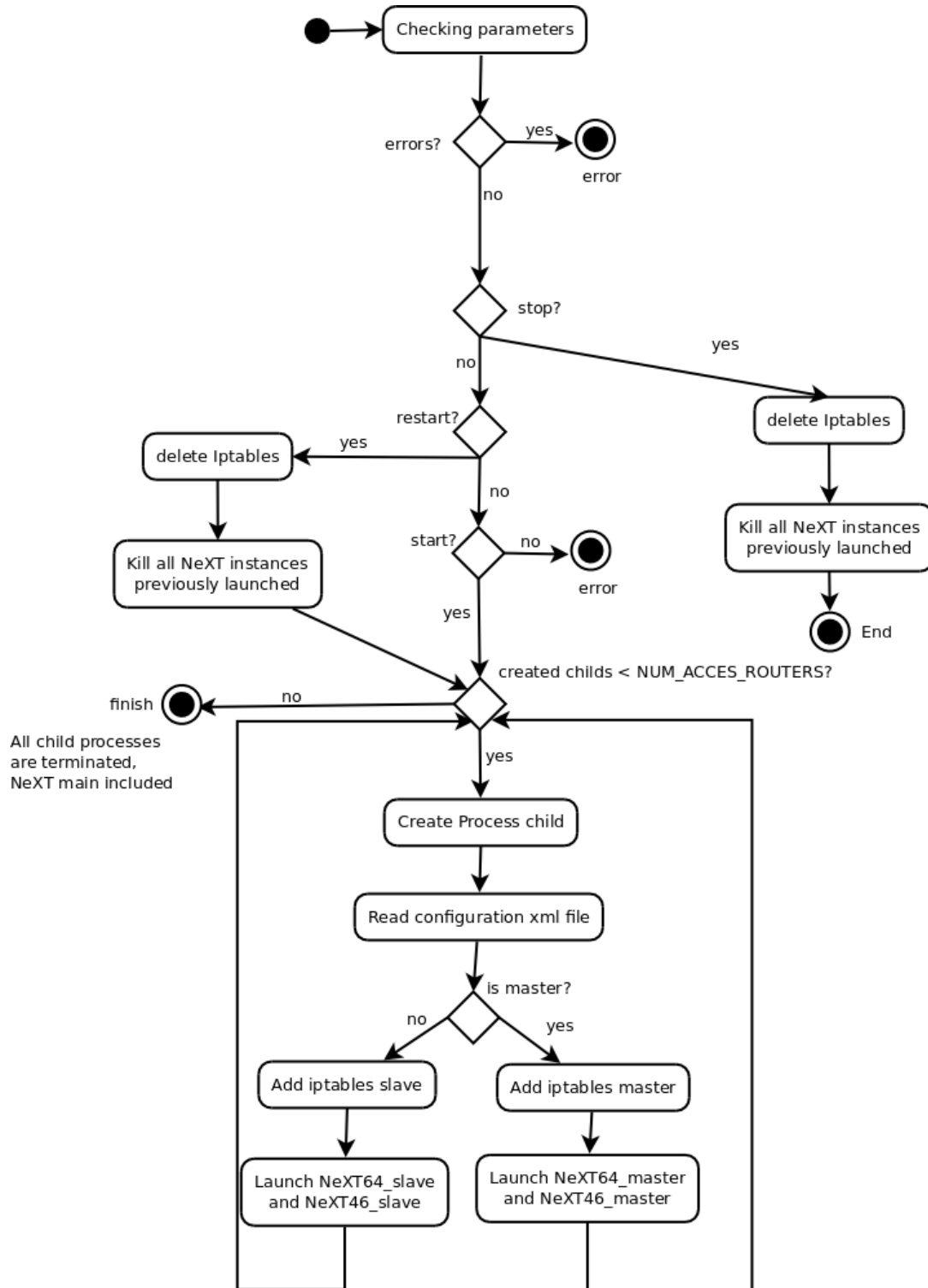


Figure 6.8: NeXT launcher flow

however, it is recommended to first have a glance of how NeXT multiprocess flow looks like.

Figure 6.8 shows the flow chart that illustrates NeXT new version main steps. Three main parts can be differentiated from the flow:

1. Check of parameters
2. Removal of iptables rules and kill of previous existent NeXT instances
3. Addition of iptables rules, creation and launching of NeXT instances

The first one is related with the integrity of the parameters given as input to the *NeXT launcher*. To execute NeXT launcher the following command is typed in a shell:

- `./NeXT start 3 master sat 1`

”./NeXT” refers to the launcher binary, “start” refers to the operation mode, “3” defines the debug messages level of the protocol, ”master | slave “ defines the mode entity of NeXT, sat | emul defines whether NeXT will have satellite or satellite emulation configuration. Number ”1” defines the number of NeXT instances to be launched. Those are the parameters that the launcher deals with.

In the *check of parameters* part, those given parameters are checked whether they fulfill the required format or not. If there is any of them that do not match then an error is prompted saying what data input is expected to be received.

After this integrity checking parameter part, whether NeXT has to be stopped or restarted is checked. It deletes possible iptables established from a previous NeXT execution. How it is known that a previous execution of NeXT has been deployed? When either a NeXT64 or NeXT46 instance is launched its PID process are stored in files within the system, this fact allows the launcher to read those files, to obtain the pid process and afterwards kill them. If there is no file with PID information no NeXT instance is killed. If the operation mode is *stop* after deleting iptables rules and kill all NeXT instances it finishes its execution. If operation mode is *restart* then instead of finishing its execution it begins adding new rules to iptables and creating child processes to launch NeXT instances. It makes this last two steps as many times as NeXT instances had been defined as an input parameter to the NeXT launcher. While the number of NeXT instances are less than the number given, the NeXT launcher will continue creating child processes. When the NeXT launcher finishes at creating child processes, it terminates itself. It has been said that there is no need of maintaining its memory anymore.

An xml configuration file is used by NeXT launcher, it provides interfaces information, prefix and addresses information and it also provides the nfqueue number identifiers that NeXT64 and NeXT64 entity will use in the iptables. This xml file has an important role within NeXT which is to add dynamism to the code. NeXT first version had interface management hardcoded, this means that there was a predefined set of interfaces within the code. This fact implied that as a result of having interfaces management hardcoded, multiple instances of NeXT was not possible because all NeXT launched instances would share the same interface mapping (the one

### 6.3. Two different NeXT design approaches

---

hardcoded) and each instance needs a different interface configuration in order to work. This hardcoding of the interfaces was done by defining in a “.h“ file a set of interfaces using *defines*<sup>10</sup>, as this file is included in almost all the files they can be used.

NeXT uses *pugixml*<sup>11</sup>, it is a light-weight C++ XML processing library. It is developed and maintained since 2006 and has many users. All code is distributed under the MIT license, making it completely free to use in both open-source and proprietary applications. NeXT protocol is coded in C language, however the NeXT launcher is coded in C++, and it is the NeXT launcher that uses pugixml, to C files it is something transparent. The reason for using C++ in the launcher instead of C language is due to the intention of using threads, using the C library *pthread.h*, in NeXT threads an encapsulation using this library was made in order to make the code more object oriented.

Leaving this matter of pugixml usage, in order to add this dynamism to NeXT the following modifications were done, figure 6.9 shows a portion of the original code in the “.h“ file previously mentioned.

```
#define IFACE4S "eth1"
#define IFACE6S "eth0"
```

Figure 6.9: Interfaces hardcoded

To add dynamism *extern variables* are used, they get the value from upper *c files* (NeXT64 and NeXT46 entities which received the information as input parameters, the ones read from the xml configuration file). In figure 6.10 it can be seen.

```
#define IFACE4S iface4_slave
#define IFACE6S iface6_slave

extern char iface4_slave[6];
extern char iface6_slave[6];
```

Figure 6.10: Interfaces hardcoded with dynamism

When an *extern variable* is declared, the compiler is told that the variable was defined elsewhere. The compiler is told that a variable by that name and type exists, but the compiler should not allocate memory for it since it is done somewhere else. The *extern* keyword means "declare without defining". In other words, it is a way to explicitly declare a variable, or to force a declaration without a definition. When a *#define* is done it allocates memory that remains until the program finishes its execution.

It exists a xml configuration file per each NeXT working mode, an xml file is loaded depending on the parameters given to the NeXT launcher execution: *./NeXT start 3 master sat 1* If *master* and *sat* then *Satellite master* mode is loaded. All the possible modes are listed below:

- Satellite master



- Satellite slave
- Satellite emulation master
- Satellite emulation slave

Before this *NeXT multiprocessing approach* NeXT had four launching scripts, each one for each of its working mode. In this scripts some parameters had also to be given. So, whenever NeXT had to be launched it had to be thought which scripts was the correct one. Now, by using the NeXT launcher this is reduced to a only one binary file.

In the earlier paragraphs of this section it has been mentioned the use of *process number identifier* (PID) in order to both kill NeXT instances running and send a SIGNAL to a specific NeXT entity. This PID information was stored in file systems and read by NeXT64 and NeXT46 entities. Sending a SIGNAL to a specific NeXT instance now is not as easy as searching one name process, now it is done by reading the PID from a file in the system, this introduces one problematic situation which is "¿How can it be guaranteed that the file that holds the PID has been created?". If the file has not been created then there is no way of reading the PID of the entity it is intended to send SIGNALS. It is solved by applying something similar to a handshaking process. Figure 6.11 shows how it is done.

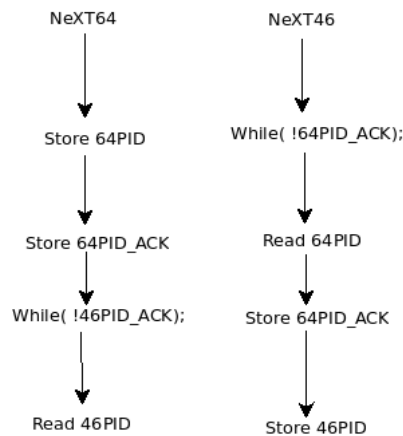


Figure 6.11: PID Handshaking

## 6.4 Threads Introduction

### 6.4.1 What is a Thread

A thread is an application task that is executed by a host computer at the same time as others. It is an independent stream of instructions that can be scheduled to run as such by the operating system. For instance, imagine a main program (threadi.bin) that contains a number of

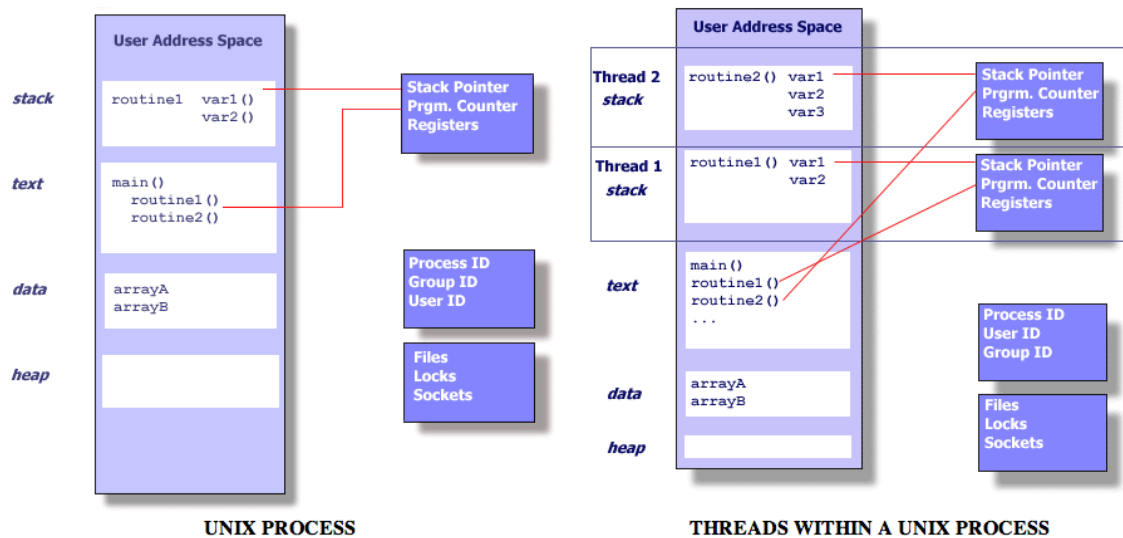


Figure 6.12: UNIX thread

procedures. Then imagine all of these procedures being able to be scheduled to run simultaneously and/or independently by the operating system. That would describe a "multi-threaded" program. In figure 6.12 an UNIX process structure and an UNIX THREAD is shown.

Threads shares memory among themselves, it means almost all variables and information are shared. Threads can have its private data though. In figure 6.13 shows in large scale this last assertion.

Things to bear in mind about threads:

- Has its own independent flow of control as long as its parent process exists and the OS supports it
- May share the process resources with other threads that act equally independently
- Dies if the parent process dies
- Is "lightweight" because most of the overhead has already been accomplished through the creation of its process.
- Changes made by one thread to shared system resources (such as closing a file) will be seen by all other threads.
- Reading and writing to the same memory locations is possible, therefore requires explicit synchronization.

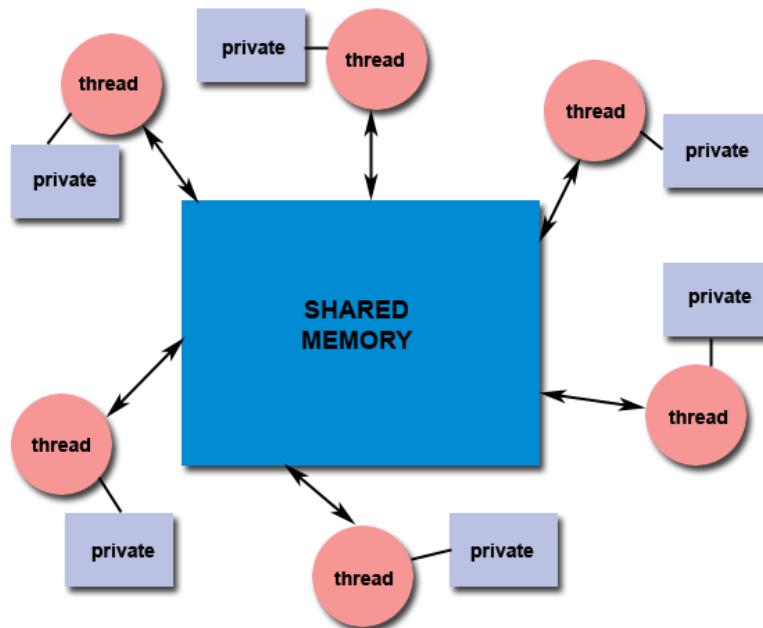


Figure 6.13: Thread Memory

### 6.4.2 Why using threads?

It has become a growing up option during last decade and it provides appealing performance results in multithreading capable nodes and nowadays almost all nodes are multithreading. It allows to have a better resource management.

Historically, threading was first exploited to make certain programs easier to write. If a program can be split into separate tasks, it's often easier to program the algorithm as separate tasks or threads. NeXT didn't match this description since what it did was follow a sequence of instructions for treating a packet. The idea of using threads in NeXT was to save space in terms of memory and a better performance taking advantage of nowadays multithreading nodes capabilities.

Why saving space? NeXT new version needs to launch multiple NeXTMaster entities, this means for each NeXTMaster entity there is a memory space to be allocated, depending of the cardinality of the #NeXTMaster entities this can become something critical if the number of ARs is big enough.

### 6.4.3 What is pthread library?

Pthreads are a set of C language programming types and procedure calls that NeXT threads uses.

Hardware vendors have implemented their own proprietary versions of threads for years. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications. In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required. For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995). Implementations adhering to this standard are referred to as POSIX threads, or Pthreads. Nowadays most hardware vendors now offer Pthreads in addition to their proprietary API's. The latest version is known as IEEE Std 1003.1, 2004 Edition.

## 6.5 NeXT threads

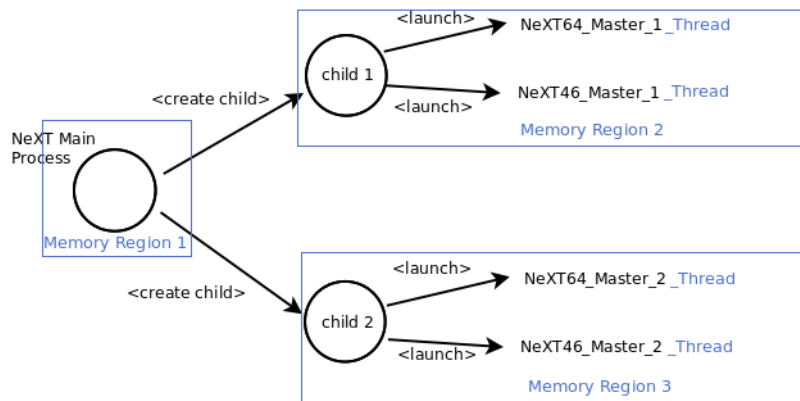


Figure 6.14: Thread Memory

Figure 6.14 shows the NeXT thread design architecture, it can be appreciated that it is similar to the one in *NeXT multiprocess* (see section 6.7). In *NeXT threads* both NeXT Main Process and child processes remains alive until a terminate signal is sent. The reason is that as a thread uses the process memory space, if its process (the one that has created it) disappears then the thread has not memory space.

NeXT64 and NeXT46 entity have each one a data structure that holds the information required to translate packets from IPv6 to IPv4 and viceversa. This data structures follows the concept of *extern variables* explained in section 6.3.1, they are defined as global variables in NeXT64 and NeXT46 entities (both has it declared with the same name) and whenever someone wants to use this data structure it has to declare it as extern. In *NeXT multiprocess design* this means that every data structure is unique and there is no conflict of names due to the fact that NeXT64 is a process and NeXT46 is another independent process. The result is the following one from the figure 6.16, there is a link or union between NeXT64 and NeXT library (inside is called "extern type\_data\_structure name"), and another link different between NeXT46 and the NeXT library. These different links avoid name conflicts with the data structure.

However in NeXT threads there is one big problem, both NeXT64 and NeXT46 have a data structure declared with the same name "struct tab table" in a global way. As in threads

environment there is the shared concept introduced in section 6.4.1, both declarations generates a "name conflict declaration" because there are two variables with the same name, "struct tab table" from NeXT64 sees the one from NeXT46 and viceversa. It also happens with the methods in the NeXT library as it can be seen in the figure 6.16.

In order to solve this problem, I thought of two options:

1. Use a common table for both NeXT64 and NeXT46
2. Declare one "struct tab" with a unique name, "struct tab table" for NeXT64 and "struct tab table\_4" for NeXT46

The first option was thought because of the shared memory feature, this allowed the two entities share the same resource and it simplified some code aspect modifications in NeXT library. However, sharing the same resource had the following implications:

- Same resource for both entities makes the concurrency of reads and writes raise, (this adds such a level of complexity that the number of reads and writes in the data structure are less than if two data structures are used due to locks for data integrity).
- NeXT64 "struct tab table" had an order inside its specification of each of its fields. NeXT46 "struct tab table" had other order.

Point two implied that almost all the code had to be changed, and there was no guarantee that it would have worked because there were so many things to be changed. NeXT code made a strong use of its data structure that hosts the information required for packets translation, thus it was like rewriting a big portion of the code. Doing this would have represented that identifying bugs in case of showing them up would have been difficult. The priority was to have a working NeXT solution in time. This option mention decreased the overall of writes and reads in a certain period time due to the fighting for the common resource and locking aspects.

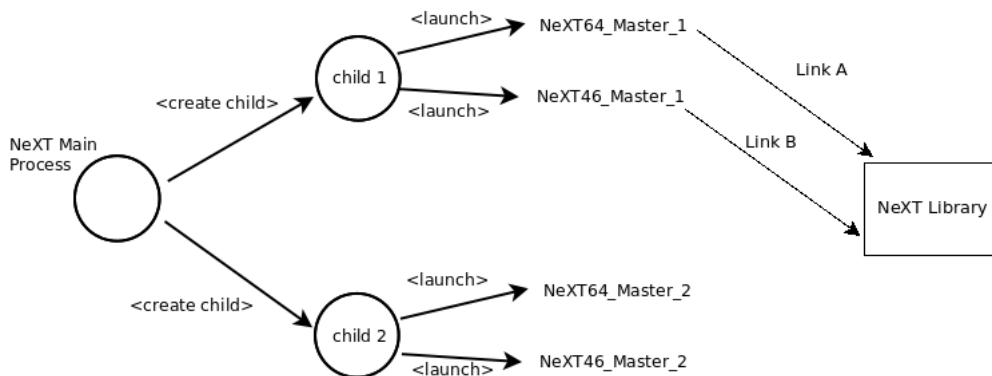


Figure 6.15: NeXT multiprocessing table linkage

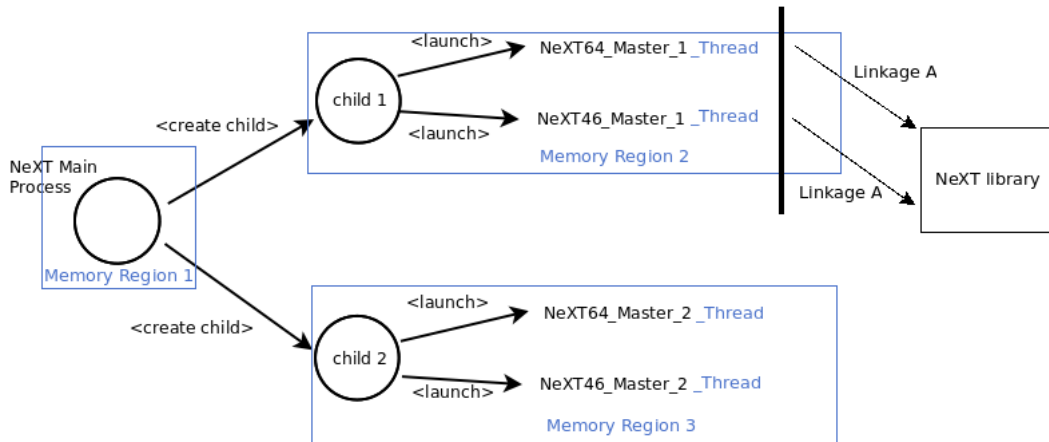


Figure 6.16: NeXT threads table linkage

Declare one "struct tab" with a unique name, "struct tab table" for NeXT64 and "struct tab table\_4" for NeXT46 was what was more feasible and was developed in a final term. Within NeXT library there were some methods exclusively used by NeXT64 and others only used by NeXT46, this allowed a differentiation between "struct tab table" for NeXT64 and "struct tab table" for NeXT46.

There was a problem with the *file descriptor variable*, it happened the same problem as the "struct tab table". There was a global file descriptor variable defined in NeXT64 named *fd\_log*, but there was another global file descriptor variable named *fd\_log* in NeXT46. Due to the shared memory concept there was a *name collision*. For that reason the differentiation between *fd\_log* in both NeXT64 and NeXT46 was done, but only in those two. The file descriptor variable is used in a function inside the NeXT library methods, this method is:

- *int WriteLog( DebugLev debuglev, char \*str,...)*

Inside this function an *extern FILE \*fd\_log* is done. To solve the problem the WriteLog function had to change, turning into:

- *int WriteLog( DebugLev debuglev, FILE\* fd\_log, char \*str,...)*

Therefore, the *fd\_log* variable had to be spread among the methods that used the WriteLog function in order to be used, *extern* variable was not used anymore.

There is a well known saying that says, try not to merge SIGNALS and threads because it tends to be very tricky and a good performance and correct execution of the code can never be guaranteed. SIGNALS with threads is not something very recommended, at least there are so many opinions about that. One of the main reasons, not the only one, is the following one. If there are three threads, let's say *T1*, *T2*, *T3*, waiting to a specific SIGNAL, let's say *SIGUSR1* SIGNAL. If the three threads have been created in the same process then, if a *SIGUSR1* is

intended to be sent to T1, it can not be done the assumption that the SIGNAL will reach T1. The result will be something random. When a thread is created in a process the process is the one that handles SIGNAL reception. Whenever the process receives a SIGNAL it looks if there is some thread waiting to this specific type of SIGNAL, *SIGUSR1* in our case. If the process has the information that there is at least one thread waiting to the specific type of SIGNAL, the process delivers the signal randomly to one of the threads that are waiting to the SIGNAL.

By dividing the memory space for each NeXT pair the problem scale is reduced to two threads, both NeXT64 and NeXT46 share some SIGNALS type, thus an emulation of SIGNAL proceeding was done.

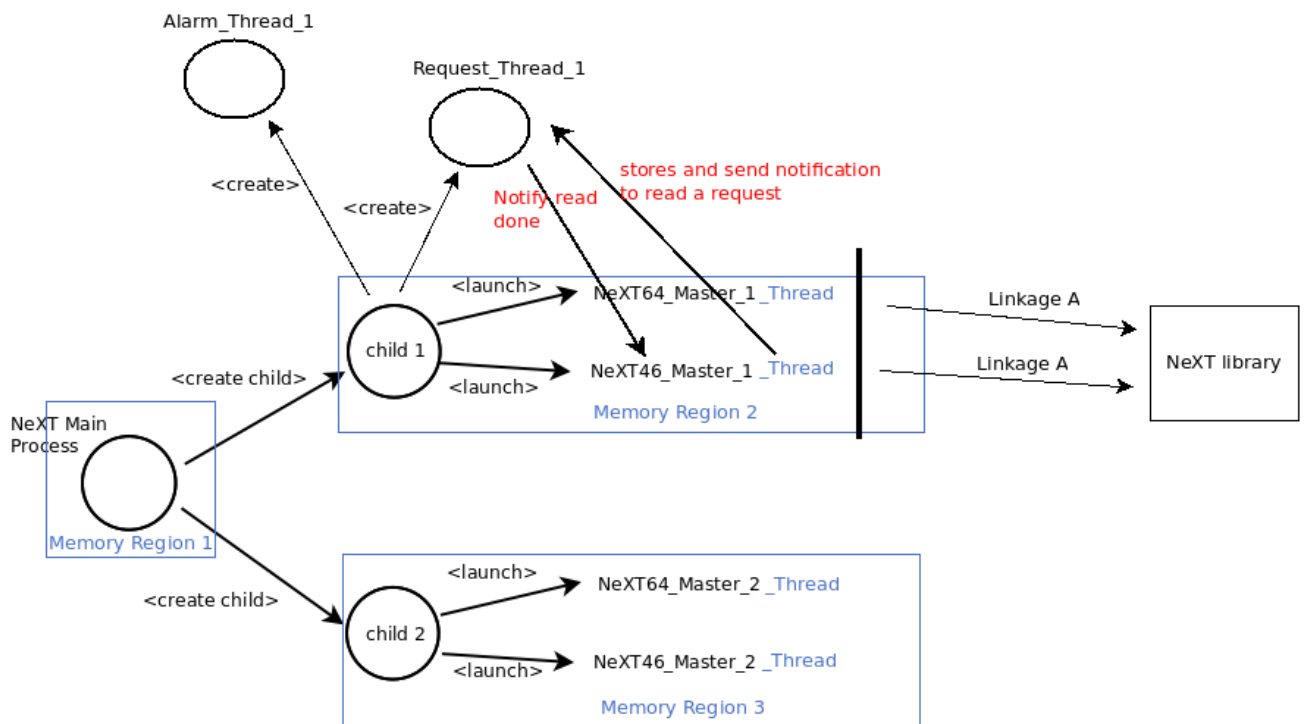


Figure 6.17: NeXT threads SIGNALS

In figure 6.17 it can be seen the design with this new SIGNAL emulation. The way it works is the following one.

- NeXT64 creates the NeXTalarm thread which periodically sends information of the MR to NeXTSlave in order to keep the communication alive and its status. This information is sent via UDP packets upon NeXTalarm request.
- NeXT NeXT64 creates the NeXTrequest thread and when NeXT46 receives a request from NeXTSlave, NeXT46 notifies the *NeXTrequest thread* that a request has been stored in it and that it can read it now. This notification is sent by using "*pthread\_cond\_signal(pthread\_cond\_t \*cond)*" pthread function and an emulation of semaphores with locks. For further information visit the "*pthread\_cond\_signal(3) Linux man page*".

- NeXTrequest thread reads the request, makes its local tasks and notify NeXT46 that it has finished and that it can continue again with its tasks.
- NeXTrequest thread waits for another notification

Doing like this the SIGNAL behavior is emulated. It has to be said that SIGNAL emulation of SIGTERM was not implemented. There were some strange behaviors such as that after NeXTMaster had sent the first packet (the one that tells NeXTSlave some information about NeXTMaster) to its NeXTSlave, no routers advertisements were observed. Instead of router advertisements there were some packets that seemed UDP data packets but they were considered by the NeXTMaster as router advertisements (Ra) when in the Wireshark sniffing tool they didn't appear as Ra. My guess is that the problem was due to a shared resource within the code, in charge of handling requests.

## 6.6 NeXT design chosen

NeXT multiprocess has been chosen as the final option. It is simpler than NeXT threads, it works, and it can be easily extended, modified, adapted and also maintained. NeXT threads implementation implied a non easy going code and a code not easily maintainable. However thread technology can contribute to good performance in a future NeXT version which is something good to consider in a close future. One of the objectives was to make NeXT and ROHC working together at the same time in the test-bed and it has been accomplished.

Robust Header Compression (ROHC) is a standardized method to compress the IP, UDP, RTP, and TCP headers of Internet packets. It performs well over links where the packet loss rate is high, such as wireless links.

In streaming applications, the overhead of IP, UDP, and RTP is 40 bytes for IPv4, or 60 bytes for IPv6. For VoIP this corresponds to around 60% of the total amount of data sent. Such large overheads may be tolerable in wired links where capacity is often not an issue, but are excessive for wireless systems where bandwidth is scarce.

There is information in headers that can be predicted, there is information that is always the same, there is other information that can be predicted such as the sequence number of a packet in a specific flow. However there is other information that can not be predicted because there is no pattern to predict. ROHC take advantage of this dynamic fields in headers to save space, it converts them into a sequence number in the compressor and the decompressor interprets them.

ROHC compresses these 40 bytes or 60 bytes of overhead typically into only 1 or 3 bytes by placing a compressor before the link that has limited capacity, and a decompressor after that link. The compressor converts the large overhead to only a few bytes, while the decompressor does the opposite. In SANDRA scenario using NeXT the compressor was placed at MR and the decompressor at HA. ROHC headers are placed in lower bits than NeXT headers, thus NeXT headers are applied after ROHC protocol is applied to the packet. It can be said that NeXT headers acts as a tunnel carrying ROHC headers. Both MR and HA can act as either compressor or decompressor.



# Chapter 7

---

## Test Cases

---

The test cases that will be analyzed are two, *Case1 (7.1)* and *Case2 (7.2)*. Initially There were more test cases, however all of them could be summarize as Case 1 (7.1), so they were not specified in the report. Test cases analyzed are structured as follows.

An explanation of which is the *Aim* of doing the test in the specific case is initially presented. There is afterwards an enumerated list of steps describing the configuration of the scenario and the steps that are taken to produce changes in that behavior. After this list it is shown the *expected results* of the testing and the *real results observed* after applying all steps in the testing. Finally there is a technical discussion about *real results observed*, and if it does not match with the *expected results* an explanation of why both two results do not match is given.

In this report a figure of the current test-bed was previously presented, it is figure 5.1. Consulting figure 5.1 allows to follow easily explanations given after the *results observed* part in the specific case. It also helps on the understanding of the initial configuration of the test-bed specified in the enumerated list. It is recommended to consult figure 5.1, it can be used as support material.

## 7.1 Case 1

**Aim of the test case:** Demonstrate that with a starting scenario where two ARv4 are up and traffic pass through them, if one AR goes down traffic is redirected to the other AR link.

1. MR-AR1 and MR-AR2 links using IPv4 are up.
2. Ping from MNN SCgW to CN SCgW.

### 3. Disconnect either AR1 or AR2 link.

- Result expected: Traffic goes through the remaining link.
- Result observed: There is traffic from MNN SCgW to MR but there is no traffic from MR to ARs.

The result observed is due to a known bug in the mip6d software. But before explaining what this bug is about some aspects about mip6d software are going to be introduced for a better understanding. When MR configures a Care-of address (CoA) after receiving Ra from AR, a tunnel interface is created automatically by the mip6d software, this tunnel will be named from now on *tun1*. *Tun1* is the NEMO tunnel, the one that has the MR Home Address and HA Home Address. Assume that *tun1* is the NEMO tunnel that MR-AR1 link uses, all packets that have a CoA configured from AR1 prefix are routed to *tun1* NEMO tunnel. MR-AR2 link has its own NEMO tunnel too, this tunnel will be named from now on *tun2*. Theoretically all mobility signaling packets between MR and HA traverses its correspondent NEMO tunnel interface. Thus BU related to MR-AR1 link goes through *tun1* and BU related to MR-AR2 link goes through *tun2*.

However, all Binding Updates (both from MR-AR1 and MR-AR2 link) go only through one NEMO tunnel. This tunnel is the first one configured by mip6d. This means that all mobility signaling is routed via one NEMO tunnel. This is the bug in mip6d software that causes the final result to not matching with the result that was expected.

Theory about the mip6d bug has been explained, now it will be explained how it can be identified that this bug is the cause of why when one MR-ARx link is put down then data traffic disappear in the remaining MR-ARx link.

Starting from point one from the enumerated list from above (AR1 and AR2 links are up...), there are two version4 links up. Then afterwards it comes step 2 (Ping from MNN SCgW to CN SCgW...), traffic starts to pass through AR links. But suddenly one MR-ARx link is put down, it means that one NEMO tunnel disappears, therefore the only path that mobility signals had to go through disappear. It can be easily checked by listing the Binding Cache in HA and the BU list in MR. In MR it appears that the BU related with the remaining MR-AR link appears as it has errors (it could not be delivered to HA) due to the NEMO tunnel it was using disappeared. As HA has not an updated BC, when a packet coming from the remaining NEMO tunnel reaches HA, HA looks into its BC and searches for an entry that matches to the packet CoA. As the entry related with this CoA has expired (because no new BU renewal signaling arrived), HA discards the packet because it has no information to route the packet with the CoA to a recipient.

This mip6d bug has a known solution that can be applied, at least it is commented in the mip6d mailing list that this solution works. To solve the problem some patches would have to be applied. There was scarcity of time when this bug was identified, and as the results of applying the patches were not fully reliable, those patches were not deployed in order to have the testbed operative and allow to modify some of its resources that implied having the test-bed operative.

## 7.2 Case 2

**Aim of the test case:** Demonstrate that two instances of NeXTMaster works together in MR.

1. MR-AR1 and MR-AR2 links using IPv4 are up.
  2. Server-PT LFN (MN) pings Server-PT-CN (CN)
  3. MN ScGw pings CN ScGw
  4. Add a mark to packets with iptables
  5. Put AR1 down
- Result expected when AR1 and AR2 are up: IPsec traffic from MN goes via MR-AR1 link and UDP traffic goes through MR-AR2 link.
  - Result observed when AR1 and AR2 are up: IPsec traffic from MN goes via MR-AR1 link and UDP traffic goes through MR-AR2 link.
  - Result expected when AR1 is down and AR2 is up: There is no IPsec traffic in MR-AR1 link and UDP traffic goes through MR-AR2 link.
  - Result observed when AR1 is down and AR2 is up: IPsec traffic from MN goes via MR-AR1 link and UDP traffic goes through MR-AR2 link.

This test case introduces an important and interesting aspect related with an IP field header which changes from IPv4 to IPv6. This field is the *Type of Service* in IPv4, in IPv6 it replaced by the IPv6 header field *Type of Class*. *Type of service* field was used to identify if a packet was from a specific service and then do some actions according to that fact. *Type of service* marking is supported by *iptables*, a snapshot of the Linux iptables man page is shown in figure 7.1 and figure 7.2.

The field *Type of Service* marking was not supported in the iptables version installed in the Linux running in the test-bed, thus marking packets in *Type of Service* field was not possible.

This field was important in order to differentiate two type of traffic packets:

- IPsec type packets coming from MN going to CN.
- UDP type packets coming from MN SCgW going to CN SCgW.

When a packet from MN enters to the Security Gateway the outer IPv6 header is encrypted by IPsec, thus the IPv6 field *Type of Service* can not be read. When the packet is encrypted an outer IPv6 header is added but with the SCgW addresses (MN SCgW as source address and CN SCgW as destination). This kind of traffic it can only be identified as IPsec traffic because after the new outer IPv6 header there is an IPsec header.

**TOS**  
This module sets the Type of Service field in the IPv4 header (including the 'precedence' bits) or the Priority field in the IPv6 header. Note that TOS shares the same bits as DSCP and ECN. The TOS target is only valid in the **mangle** table.

**--set-tos value[/mask]**  
Zeroes out the bits given by mask and XORs value into the TOS/Priority field. If mask is omitted, 0xFF is assumed.

**--set-tos symbol**  
You can specify a symbolic name when using the TOS target for IPv4. It implies a mask of 0xFF. The list of recognized TOS names can be obtained by calling iptables with **-j TOS -h**.

The following mnemonics are available:

**--and-tos bits**  
Binary AND the TOS value with bits. (Mnemonic for **--set-tos 0/invbits**, where invbits is the binary negation of bits.)

**--or-tos bits**  
Binary OR the TOS value with bits. (Mnemonic for **--set-tos bits/bits**.)

**--xor-tos bits**  
Binary XOR the TOS value with bits. (Mnemonic for **--set-tos bits/0**.)

Figure 7.1: extract of iptables man Linux page - Marking Section

**tos**  
This module matches the 8-bit Type of Service field in the IPv4 header (i.e. including the "Precedence" bits) or the (also 8-bit) Priority field in the IPv6 header.

[!] **--tos value[/mask]**  
Matches packets with the given TOS mark value. If a mask is specified, it is logically ANDed with the TOS mark before the comparison.

[!] **--tos symbol**  
You can specify a symbolic name when using the tos match for IPv4. The list of recognized TOS names can be obtained by calling iptables with **-m tos -h**. Note that this implies a mask of 0x3F, i.e. all but the ECN bits.

Figure 7.2: extract of iptables man Linux page - Matching Section

If traffic goes from one SCgW to the other SCgW and this traffic is not mobility signaling (BU) then no IPsec encryption is done and if a ping is done it can be identified as UDP traffic.

To overcome this another option in IPv6 ip6tables is used, this option is **-j MARK --set-mark**, it is used to add a kernel mark to the packet. The following lines shows the ip6tables rules added to the MR node in order to mark the packets and differentiate the traffic and route it to a specific MR-ARx link.

```
#Non Ipsec Marking packets
```

- ip6tables -A PREROUTING -t mangle -s 2001:5c0:1505:6101::/64 -d 2001:a:1::/64 -j MARK --set-mark 150

```
Interface "eth4" {  
  Bid 100;  
  BidPriority 10;  
  Reliable true;  
}  
  
Interface "eth7" {  
  Bid 150;  
  BidPriority 10;  
  Reliable true;  
}
```

Figure 7.3: Mip6d.conf interfaces with BID

```
#IPsec Marking packets
```

- `ip6tables -A PREROUTING -t mangle -s 2001:5c0:1505:6101::/64 -d 2001:a:1::/64 -protocol esp -j MARK --set-mark 100`

Mip6d software uses `mip6d.conf`, in this file there are interfaces and other aspects related to mobility. One of them is the interface information and its *Bid identifier number (BID)*, the one used in `--set-mark option`. It can be appreciated below in figure 7.3 how the BID is used in `mip6d.conf` file.

When MR-AR1 link is put down then the traffic that went via this link is discarded because `mip6d` can not identify the IPv6 interface with BID 100 in charge of Mobility, the one `mip6d` is listening to. For that reason as `mip6d` doesn't find any match it discards packets that should go via MR-AR1 link. Traffic in MR-AR2 continues because `mip6d` has still an operative IPv6 interface to listen with a valid CoA.



# Chapter 8

---

## Conclusions and future Work

---

### 8.1 Conclusions

The main objective of this project was to adapt NeXT to the new requirements and topology introduced by SANDRA and guarantee that it worked with ROHC protocol running at the same time. Before choosing NeXT others protocols were considered but discarded. In my opinion, nowadays an efficient solution for IPv4 link traversal for IPv6 packets has to be developed. DSMIP is a very attractive concept, and it introduces good enough ideas, however, it makes the same mistake as M6T which is the non controlled addition of IPv4+UDP header if not in all in almost all packets. This have a significant impact in terms of efficiency in IPv4 links such as Satellite links in which bandwidth is very important and also time/price of consume.

NeXT overcome this issue, NeXT adds IPv4+UDP header in its IPv4 packets, it is true, however, the amount of bytes that traverses the IPv4 link is fewer than other existing protocol solutions due to NeXT first sends in the first packet all IPv6 information needed to make the IP version translation . This allows to send later packets with fewer bytes. Sooner or later there will be a protocol that would have together the best characteristics of both DSMIPv6 and NeXT, until then NeXT stands for being the most appealing option.

The process of adapting NeXT to the new scenario has been achieved. It has been proven that *Robust Header Compressor protocol* and NeXT worked together without any problem. It was tested by using a VoIP application (*linphone*) with headphones and microphones between CN and MN. The new NeXT version is more dynamic than before, the fact of having extracted the interfaces from the code and made them exchangeable adds a dynamic and independent factor that allow NeXT to easily adapt to news scenarios with exchangeable interfaces in a real scenario.

The other main objective was to automate the test-bed in order to reduce human interaction. The option chosen, a *Graphical User Interface (GUI)* has been proved as an appealing solution

and has shown positive results. It provides an interface to the final user of something that is technical indeed but it can be seen as something less complex and easy to manage. This GUI is not a final release, it is a young tool that has been conceived, a tool thought for being extended and also easy maintained and modified for other purposes (e.g. changes in the test-bed topology architecture, process of passing from NEWSKY project to SANDRA project).

## 8.2 Future work

The *Graphic User Interface* is a tool that can be extended and adapted to new test-bed topologies. Some functionalities in the current GUI were discarded due time and objectives, the main objective was to define an initial and operative base tool. Actions such as changing the type of the link that the MR uses can be done from the GUI (this function was implemented but was not full operative).

Other tasks that can be done is to provide SNMP capabilities to the test-bed nodes for an easy monitored control.

The test-bed uses XEN hypervisor, it uses XEN VM thus these VM has a limitation of 8 interfaces each one. This is a technical limitation documented in XEN web page that limits the testbed because it only allows to have two NEMO tunnel interfaces, this has been explained in section 7.1. It would be of interest to find a way to replace the two interfaces dedicated to Mip6d, the idea would be to use tap interfaces. A tap is a loopback interface that works in layer 2. Some attempts were done to make it work but they were unsuccessful.

An extension to the current NeXT implementation could be adding the functionality of launching and specific NeXT instance bound to an AR instead of always relaunching all previously defined in the xml configuration file.

One of the objectives that has not been fulfilled is turning NeXT into threads due to technical reasons. Turning it into threads is something appealing however, things such as easy code upkeeping and code easy to understand have to be considered to decide whether it is worth or not.

NeXT is an efficient technology applied to IPv4 traversal, and it is without any doubt a solution that can help into the intermediate process of migration from IPv4 to IPv6.



# Chapter 9

---

## References and Bibliography

---

### References

1. RFC 2460, Internet Protocol, Version 6 (IPv6) Specification
2. RFC 2374, An IPv6 Aggregatable Global Unicast Address Format
3. RFC 4301, Security Architecture for the Internet Protocol
4. <http://www.inmarsat.com/Services/Land/BGAN/default.aspx>
5. <http://natisbad.org/m6t/>
6. RFC 5555, Mobile IPv6 Support for Dual Stack Hosts and Routers
7. <http://www.nautilus6.org/>
8. Linux man page “ssh-keygen”
9. Linux man “page signal“
10. <http://www.cprogramming.com/reference/preprocessor/define.html>
11. <http://pugixml.org/>

---

## Bibliography

12. RFC3095 (ROHC)
13. RFC 3963 (NEMO)
14. RFC 3775 (Mobile IPv6)
15. Understanding IPv6, Author: Joseph Davies, Microsoft Press (Book)
16. Programming with POSIX(R) Threads, Author: David R. Butenhof (Book)
17. Quadern de Laboratori de Xarxes de Computador (Book)  
Autors: Llorenç Cerdà-Alabern i José M. Barceló-Ordinas  
Departament d'Arquitectura de Computadors, Enginyeria en Informàtica  
Ref. 44301
18. Xarxes de Computadors, Conceptes bàsics (Book)  
Autor: Llorenç Cerdà Alabern  
Edicions UPC
19. Linux IPv6 Stack Implementation Based on Serialized Data State Processing (Paper)  
Hideaki YOSHIFUJI, Kazunori MIYAZAWA, Masahide NAKAMURA, Yuki SEKIYA, Hiroshi ESAKI, Jun MURAI.  
VOL.E87-B,NO.3 MARCH 2004
20. USAGI IPv6 IPsec Development for Linux (Paper)  
Mitsuro Kanda, Kazunori Miyazawa, Hiroshi Esaki
21. Versatile IPv6 Mobility Deployment with Dual Stack Mobile IPv6 (Paper)  
Romain Kuntz, Jean Lorchat
22. IPv6 IPsec and Mobile IPv6 implementation of Linux (Paper)  
Kazunori Miyazawa, Masahide Nakamura
23. Air Traffic Management Network Based on IPv6 Protocol Stack (Paper)  
Eriza Hafid Fazli, Àngels Via Estrem, Núria Riera Díaz
24. IP Overhead Comparison in a Test-bed for Air Traffic Management Services (Paper)  
Eriza Hafid Fazli, Àngels Via Estrem, Núria Riera Díaz, Sébastien Dufлот, Markus Werner
25. The concept of robust header compression, ROHC (White Paper)  
EFFNET AB
26. An introduction to IP header compression (White Paper)  
EFFNET AB
27. Mobile IPv6 Technology Review (Document)  
Lancaster University

28. IPv6 Networking Over Satellite For Mobile User Group (Paper)  
Àngels Via Estrem, Axel Jahn
29. <http://umip.linux-ipv6.org/index.php?n=Main.HomePage>, [mip6d webpage]
30. <http://www.cplusplus.com>
31. <http://stackoverflow.com>
32. <http://www.triagnosys.com>
33. <http://www.debian.org>
34. <http://www.linuxquestions.org>
35. <http://www.xen.org>
36. <http://www.netbeans.org>
37. <http://www.eclipse.org>
38. <http://live.gnome.org/Dia>

Note:

- All RFC in both *References* and *Bibliography* have been read
- (Links last visit - 23-03-2011)

