



Information Society
Technologies



Title: Deliverable D6.2 Report on final prototypes, network integration and validation	Document Version: 1.2
---	-------------------------------------

Project Number: 027002	Project Acronym: ENABLE	Project Title: Enabling efficient and operational mobility in large heterogeneous IP networks
----------------------------------	-----------------------------------	---

Contractual Delivery Date: 31/12/2007	Actual Delivery Date: 31/12/2007	Deliverable Type* - Security**: R – PU
---	--	--

* Type: P – Prototype, R – Report, D – Demonstrator, O – Other
** Security Class: PU- Public, PP – Restricted to other programme participants (including the Commission), RE – Restricted to a group defined by the consortium (including the Commission), CO – Confidential, only for members of the consortium (including the Commission)

Responsible and Editor/Author: Miguel Ponce de Leon	Organization: WIT-TSSG	Contributing WP: WP6
---	----------------------------------	--------------------------------

Authors (organizations): Karl Mayer (IABG), Wolfgang Fritsche (IABG), Timm Brueck (IABG), Philipp Hofmann (IABG), Alejandro Perez Mendez (UMU), Niklas Steinleitner (UGOE), Miguel A. Diaz (CONSULINTEL), Luca Battistoni (TI), Michele La Monaca (TI), Ting Yang (Huawei), Qazi Bouland Mussabbir (Brunel), Niall Clancy (WIT-TSSG), Leigh Griffin (WIT-TSSG), John Ronan (WIT-TSSG), Eamonn Power (WIT-TSSG)
--

Abstract: <p>This document describes the integration of elements, entities and components from work packages 2-5 and highlights the verification of the MIPv6 service environment architecture and requirements.</p>
--

Keywords: software development, prototypes, test-bed, demonstration, IPv6, Mobile IPv6.

Revision History

The following table describes all the main changes completed on the document since its creation.

Revision	Date	Description	Author (Organization)
V0.1	27/07/2007	Document creation and initial ToC	M. Ponce de Leon (WIT-TSSG)
V0.2	27/11/2007	Added Section 2.1.4: Home Agent load sharing. Added Section 2.1.3 AAA for MIPv6	Karl Mayer, Wolfgang Fritsche, Timm Brueck, Philipp Hofmann (IABG). Alejandro Perez Mendez (UMU)
V0.3	28/11/2007	Added Section 2.2.1: Mobile IPv6 Firewall Traversal and section 3.2: Test bed for NSIS / Mobile IPv6 Firewall Traversal. Modifications to Section 2.1.4 Added Section 2.2.2	Niklas Steinleitner (UGOE) Karl Mayer (IABG) Ting Yang (Huawei)
V0.4	30/11/2007	Updated Figure 2.14 Added 2.1.2 DHCPv6 with EAP and 2.1.5 Interworking with IPv4 networks Modified Section 2.1.3	Karl Mayer (IABG) Miguel A. Diaz (CONSULINTEL), Luca Battistoni (TI) Alejandro Perez Mendez (UMU)
V0.5	03/12/2007	Update to Section 2.1.2, 2.1.5, & 2.2.2	Luca Battistoni (TI), Miguel A. Diaz (CONSULINTEL), Qazi Bouland Mussabbir (Brunel)
V0.6	05/12/2007	Added section 2.1.1 and 3.1	Michele La Monaca (TI)
V0.7	06/12/2007	Added section 5	Miguel Ponce de Leon (TSSG), Niall Clancy (TSSG)
V0.8	07/12/2007	Added Section 1 , Section 4 and Appendix A	Miguel Ponce de Leon (TSSG), Niall Clancy (TSSG)
V0.9	12/12/2007	Revision of Section 2.2.1 and Section 3.2.	Niklas Steinleitner (UGOE)

V0.10	13/12/2007	Revision of Section 2.2.2 Section 3.3, Section 4, section 5.1 & Appendix A	Miguel Ponce de Leon, Niall Clancy, Leigh Griffin (TSSG), Qazi Bouland Mussabbir, Shoaib Khan (Brunel)
V0.11	14/12/2007	Revision to Section 2.1.3, and addition of Section 5.1.2	Alejandro Perez Mendez (UMU) Niklas Steinleitner (UGOE)
V0.12	17/12/2007	Added Executive Summary, Section 5.3, Appendix B and Appendix C	Miguel Ponce de Leon (TSSG)
V0.13	18/12/2007	Added TIs changes and cleaned up References	Niall Clancy (TSSG)
V1.0	21/12/2007	Integrated changes and made version one	Niall Clancy (TSSG), Eamonn Power (TSSG), Leigh Griffin (TSSG), John Ronan (TSSG)
V1.1	21/12/2007	Deleted comments over the document	Miguel A. Diaz (CONSULINTEL)
V1.2	21/12/2006	Updated Summary	Niall Clancy (TSSG)

Executive Summary

With the second year WP6 activities for ENABLE including the completion of software development on specific functional components from the ENABLE work packages 1 to 4, the integration of these elements, entities and components, the conformance testing of these components to the ENABLE architecture and finally the demonstration of this new MIPv6 service environment, this document D6.2 describes the final achievements of WP6.

The document sets forth how ENABLE has realised three of its key seven objectives through WP6. That is to say, the document shows how ENABLE has “*developed the required technologies to enable the deployment of Mobile IPv6 in real-life environments*” (Objective 2). Section 2 provides an in-depth explanation of how the ENABLE software developed technological components were designed and created.

With ENABLE looking to “*investigate solutions to improve the reliability of Mobile IPv6 and enable an optimal usage of network resources for the deployment of Mobile IPv6 in a provider network*” (Objective 3), Section 3 provides an insight into the infrastructure necessary for the deployment of Mobile IPv6 in a provider network with Section 4 elaborating on a service which has high reliability requirements (search and rescue), and provides a demonstration of the solution necessary to enable an optimal usage of the network resources in this scenario.

Finally ENABLE had set an objective to “*validate the results of the developed mechanisms and technologies through prototyping and laboratory testing*” (Objective 6), and in Section 5 the details of the methodology used to validate and verify the ENABLE developments with details of the system tests carried out in this methodology described. Section 6 illustrates the collaboration efforts with IST-ANEMONE.

With the ENABLE software components tested, integrated and demonstrated as the application scenario (search & rescue), WP6 has gone some way to show that ENABLE research has facilitated efficient and operational mobility in large heterogeneous IP networks.

Table of Contents

1.	<i>Introduction</i>	8
2.	<i>Final Prototyping</i>	9
2.1	Integrated Software of ENABLE	10
2.1.1	EAP-based MIPv6 bootstrapping	10
2.1.1.1	Xsupplicant module (MN)	11
2.1.1.2	Freeradius module (ASA)	12
2.1.1.3	Radius User DB module (ASA).....	13
2.1.2	DHCPv6 with EAP.....	14
2.1.2.1	Overview	14
2.1.2.2	Module NAS	17
2.1.2.3	Module ASA/Freeradius and interfaces Pe and Pk	18
2.1.2.4	Module DHCPv6 client and interface Pi.....	19
2.1.2.5	Module DHCPv6 relay and interface Pj.....	22
2.1.2.6	Module DHCPv6 server and interface Pj.....	27
2.1.3	AAA for MIPv6	33
2.1.3.1	Module ike2d-mn (MN).....	34
2.1.3.2	Module ike2d-ha (HA).....	38
2.1.3.3	Module diametermip6 (HA).....	42
2.1.3.4	Module msad (MSA).....	45
2.1.4	Home Agent load sharing.....	47
2.1.4.1	Overview	47
2.1.4.2	Module NETSNMP and interface He	49
2.1.4.3	HA-DB module	50
2.1.4.4	HA Manager Module and interfaces Pc and Da.....	52
2.1.4.5	HA Select module and interfaces Pd and Ac	53
2.1.5	Interworking with IPv4 networks.....	54
2.1.5.1	DSMIP6.....	55
2.1.5.1.1	DSMIP6 Overview	55
2.1.5.1.2	Dual Stack MN and HA	55
2.1.5.1.3	Movement detection.....	56
2.1.5.1.4	Kernel development	57
2.1.5.1.5	MIPL userspace development	57
2.1.5.1.6	MN development.....	58
2.1.5.1.7	HA development	58
2.1.5.2	Softwires.....	59
2.1.5.2.1	Softwires overview.....	59
2.1.5.2.2	Modules SC and SI and interfaces	62
2.2	Additional Software Developments	68
2.2.1	Mobile IPv6 Firewall Traversal	68
2.2.1.1	Mobile IPv6 Firewall Traversal Reference Architecture	68
2.2.1.2	Software modules.....	70
2.2.1.2.1	MIPL (MN)	70
2.2.1.2.2	MIPL (HA).....	77
2.2.1.2.3	MIPL (CN).....	79
2.2.1.2.4	MIP6FWD.....	80

2.2.1.2.5	NSIS	81
2.2.1.2.6	NAT/FW NSLP	82
2.2.1.2.7	ip6tables	85
2.2.2	Mobility optimisations	85
2.2.2.1	Overview of FMIPv6 applicability to the GSABA architecture	86
2.2.2.1.1	MN	87
2.2.2.1.2	AR	87
2.2.2.1.3	GSABA Proxy/Server	87
2.2.2.2	Implemented Software Modules	88
2.2.2.2.1	MN	88
2.2.2.2.2	ARs (pAR & nAR).....	94
2.2.2.2.3	GSABA Server	98
3.	<i>Final test-bed design</i>	99
3.1	Test-bed for Integrated Software	99
3.2	Test bed for NSIS / Mobile IPv6 firewall traversal	100
3.3	Test bed for Mobility Optimisations	101
4.	<i>Instantiation of the Application Scenario</i>	103
4.1	Definition of the ENABLE Demonstration	103
4.1.1	SAR Scene 3 Demonstration.....	103
4.1.2	SAR Scene 6 Demonstration.....	104
4.2	Trial of the ENABLE Demonstration	106
4.2.1	Trial Test bed	106
4.2.2	ENABLE Software Components	106
4.2.3	Demonstration Visualisation Application	106
4.2.4	Ruby Server.....	107
4.2.5	Demonstration Example	108
5.	<i>Test Management and Methodology</i>	109
5.1	Test Cases	110
5.1.1	Summary of Test Case Scenarios.....	110
5.1.2	Integrated Software Test Cases	111
5.1.2.1	Enable Test Case Scenario 001: Split bootstrapping	111
5.1.2.2	Enable Test Case Scenario 002: HA Load Sharing.....	113
5.1.2.3	Enable Test Case Scenario 003: Integrated Bootstrapping EAP.....	114
5.1.2.4	Enable Test Case Scenario 004: Integrated Bootstrapping DHCPv6	117
5.1.2.5	Enable Test Case Scenario 005: Handover IPv4Interworking.....	119
5.1.2.6	Enable Test Case Scenario 006: Split bootstrapping PDA	120
5.1.2.7	Enable Test Case Scenario 007: Handover PDA	121
5.1.2.8	Enable Test Case Scenario 008: VoIP Call with Streaming Audio	123
5.1.2.9	Enable Test Case Scenario 009: Streaming Video with Handover.....	124
5.1.3	Mobile IPv6 Firewall Traversal Test Cases	126
5.1.3.1	Enable Test Case Scenario 010: Pinhole creation for BU/BA after handover	126
5.1.3.2	Enable Test Case Scenario 011: Pinhole creation for BT/RO data traffic	127
5.1.3.3	Enable Test Case Scenario 012: Pinhole deletion.....	128
5.1.4	Mobility Optimisation Test Cases.....	130
5.1.4.1	Enable Test Case Scenario 013: Mobility Optimised Handover.....	130

5.1.4.2	Enable Test Case Scenario 014: Streaming with Mobility Optimised Handover	131
5.1.5	Test Case Evaluation.....	133
5.2	S/W Development Management.....	134
5.3	Test Reporting/Debugging Tools Description.....	134
6.	<i>Collaboration with IST projects (TI).....</i>	<i>136</i>
7.	<i>Conclusion.....</i>	<i>137</i>
8.	<i>References.....</i>	<i>139</i>
<i>Appendix A.</i>		<i>142</i>
<i>Appendix B.</i>		<i>151</i>
<i>Appendix C.</i>		<i>153</i>

1. INTRODUCTION

ENABLE has undertaken a significant programme of software implementation in this second year of the project which has involved the prototyping of a number of functional components as specified in [ENA-D6.1], the network integration and trials of these prototypes and finally the validation and verification of this new MIPv6 service environment.

Section 2 of this document provides an overview of these software prototypes, with a description of how each specific ENABLE technological component was designed, and then in detail shows the six functional components, EAP-based MIPv6 bootstrapping (WP1), AAA for MIPv6 bootstrapping (WP1) Interworking with IPv4 networks (WP2), MIPv6 firewall traversal (WP2), HA load sharing (WP3) and Fast Mobile IPv6 (FMIPv6) (WP4) as functionalities that will be used in the demonstration scenario.

Section 3 provides a description of the research trial infrastructure, which was initially used to check the compatibility and the functionality of the software modules being created by ENABLE and then goes on to describe the projects' MIPv6 service environment test bed, where the component integration and then the final demonstration were carried out.

Section 4 presents the link between the application scenarios as described in [ENA-D6.1], and the instantiation of the ENABLE demonstration. In [ENA-D6.1] two scenes (Scene 3 & Scene 6) from the search & rescue management scenario were chosen to be demonstrated in the final project trial and this Section 4 shows how the main actor in these scenes comes in and goes through the ENABLE MIPv6 service environment test bed, accessing the different networks (IPv6-EAP-capable, IPv4-only and dual-stack) as the search & rescue is being carried out.

Section 5 contains the methodology used to validate and verify the ENABLE developments. It also outlines the system tests carried out in the validation phase.

2. FINAL PROTOTYPING

In order to aid the deployment of an efficient and operational mobility service in large scale IPv6 network environments, as detailed in [ENA-D6.1] ENABLE identified a number of functional components to develop further into working prototypes, as shown in the Table 2-1 below.

Table 2-1 Software developments per partner

		Operational Mobile IPv6 architecture	Implemented by
From WP1	Bootstrapping and control of mobility	Split Scenario. Diameter and EAP	TI/UMU
	Bootstrapping and control of mobility	IKEv2 + EAP	UMU
	Bootstrapping and control of mobility service	EAP-based bootstrapping	TI
	Bootstrapping and control of mobility service	DHCPv6 extensions on access router	CONSULINTEL
From WP2	Middlebox traversal	NSIS solution	UGOE
	Interworking with IPv4 network	DSMIP including moving detection algorithm	TI
	Interworking with IPv4 network	Softwires as tunnelling solution for IPv4 interworking	CONSULINTEL
From WP3	Load sharing	HA decision rule based on weighted	IABG
From WP4	Mobility optimizations	FMIPv6	Huawei, Brunel

The main difference between this Table 2.1 and the one shown in [ENA-D6.1] is the addition of the developments for “DHCPv6 extensions on access router” under WP1 and the “Softwires as tunnelling solution for IPv4 interworking” under WP2. The development which remains uncompleted was the “VRRPv6 Extensions for homeagent-reliability” and so it has been removed for the list.

Five of the developed components from WP1, WP2 and WP3 have integrated seamlessly, namely the EAP-based MIPv6 bootstrapping (with and without MIPv6 DHCPv6 extensions and DNS/IKEv2), AAA for MIPv6 bootstrapping, DSMIP interworking with IPv4 networks, Softwires as tunnelling solution for IPv4 interworking, and HA load sharing. These components will be described in further detail in this section. The MIPv6 firewall traversal and Fast Mobile IPv6 (FMIPv6) while remaining as two components that were developed as separate mobility extension solutions, will also be described in the latter part of this section..

2.1 Integrated Software of ENABLE

2.1.1 EAP-based MIPv6 bootstrapping

Some EAP methods (e.g. [PEAPv2], [EAP-AKA]) are able to convey generic information items along with authentication data. This flexibility allows configuring bootstrapping parameters during the MN’s authentication for network access. Upon the successful completion of the authentication phase Configuration Type-length-values (TLV) are exchanged to deliver the bootstrapping information.

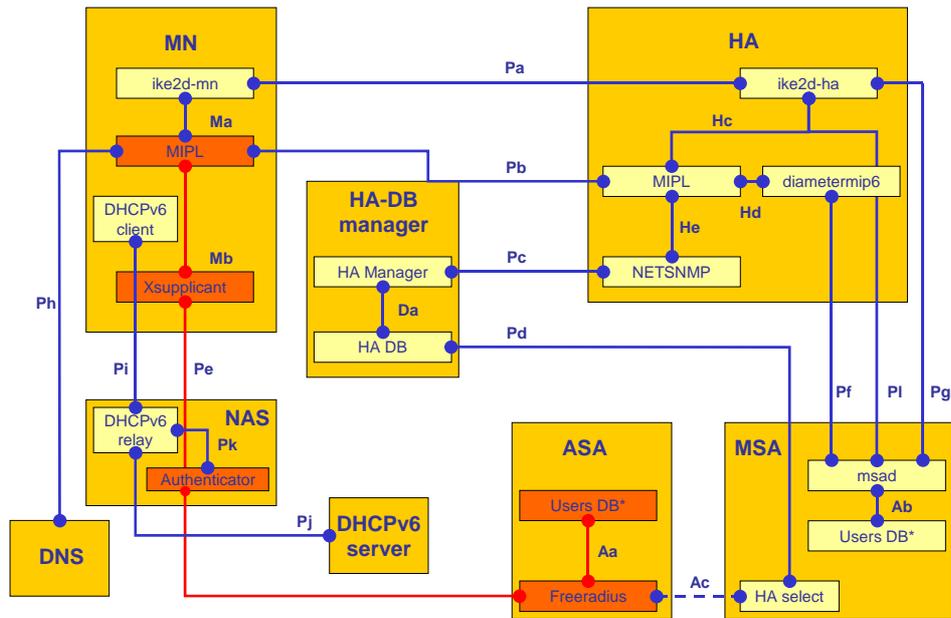


Figure 2-1 Software modules and interfaces related to EAP-based MIP6 bootstrapping

The EAP channel is therefore used to perform user authentication and the bootstrapping of MIPv6 parameters. IEEE 802.1X is used to exchange EAP packets between the MN and the AP, while RADIUS [RFC2138] is used between the AP and the ASA-AAA. The AP (authenticator) is a RADIUS client on the NAS which forwards the EAP messages coming from the supplicant (MN) to the AAA server.

2.1.1.1 Xsupplicant module (MN)

The Xsupplicant module realizes the role of IEEE 802.1x supplicant. The distribution used for development is the v1.0 of the open source Xsupplicant application [Xsupplicant].

In order to be able to perform the operation required to obtain and process the information required to bootstrap the MIPv6 service, the following modification had been done:

- Implementation of the PEAPv2 module (starting from Xsupplicant's already implemented PEAP v0/v1 module).
- Implementation of the code to handle MIPv6 service activation.

Further details on the various interfaces related to this module have been already described in [ENA-D6.1]. A major departure from [ENA-D6.1] is that the interface between Xsupplicant and MIPL (Mb) has been greatly simplified. That interface was planned as an inter process interface

implemented through a UDP socket. Since the main goal of this interface was to convey the HA address to the MIPL daemon, the overhead of keeping a socket up and running just for this purpose has been judged overkill. In the final prototype the Xsupplicant simply inserts the bootstrapped parameters in the configuration file used by the MIPL daemon.

2.1.1.2 Freeradius module (ASA)

The ASA functionality has been implemented with [FreeRADIUS] which is the premiere open source RADIUS server (released under the GNU General Public License). The server natively ships with libraries for interfacing with LDAP, MySQL, PostgreSQL and Oracle databases and it supports EAP with EAP-MD5, EAP-SIM, EAP-TLS, EAP-TTLS, EAP-PEAP, and Cisco LEAP sub-types.

TI maintains an internal version (forked as of Apr 2004 from the CVS repository) of the program to develop custom extensions, this version has been used as the starting point for the developments in ENABLE. FreeRADIUS is developed in the C language and it is an amalgam of a core module, the radiusd daemon, and a set of modules which are linked dynamically at run time (dlopen): the core is in charge of handling the RADIUS protocol, while all other functions (authentication protocols, interaction with external databases, etc.) are implemented within modules.

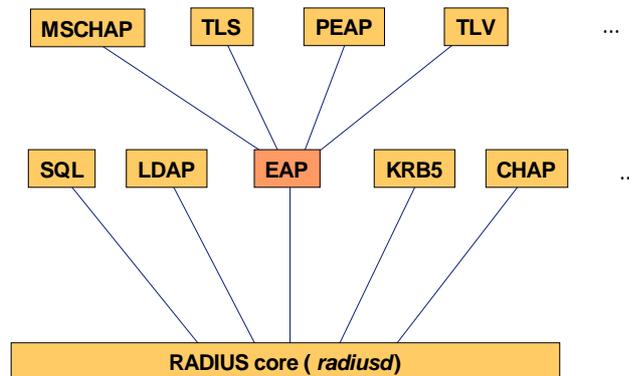


Figure 2-2: FreeRADIUS Software Architecture

Complex functionalities are realised through the usage of a cascade of modules; for example each EAP method is implemented in a separate module. The interface between the RADIUS core and its modules is standardised; the same is true for the EAP module and the modules which implement EAP methods.

In order to implement the required ENABLE functionalities, the PEAP module has been modified to support PEAPv2 and another ad hoc module (TLV) has been created to process the TLVs. A further module linked to the TLV, has been implemented for the MIPv6 bootstrapping.

Care has been taken in order to realise a modular framework: new services can be bootstrapped adding new modules and using the same interface defined for the interaction between the TLV and MIPv6 module.

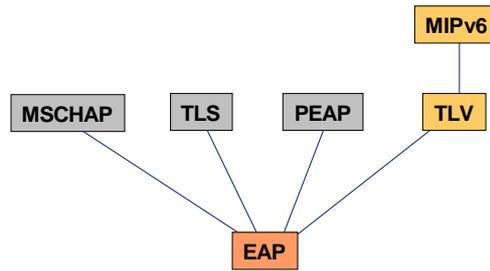


Figure 2-3: MIPv6 bootstrapping module

The SQL module has been also modified to let the MIPv6 module access a SQL database to retrieve users' information for the MIPv6 bootstrapping. Indeed, the original SQL module was only able to perform SQL queries specified in a configuration file and consequently has been adapted to perform dynamic queries using as input the SQL requests passed by the MIPv6 module.

2.1.1.3 Radius User DB module (ASA)

The Radius User DB module is a SQL database which is a compound of two main tables:

- access_profiles: holds the data for network access authorization, plus the information on which services to be bootstrapped.

customer_id	disabled	username	eap_method	credential_type	credential_data	services
00001	no	generic_peap_user	peap	NULL	NULL	NULL
00002	no	generic_tlv_user	tlv	NULL	NULL	NULL
00005	no	zidane	mschapv2	password	coupdeboule	mipv6
00004	no	cassano	mschapv2	password	elgordo	mipv6
00006	no	gattuso	mschapv2	password	ringhio	mipv6

Figure 2-4: access_profiles table

- mipv6_profiles: holds the data for bootstrapping the MIPv6 service.

username	auth_type	psk_key_length	keys_lifetime
zidane	EAP-IKEv2	NULL	NULL
cassano	PSK-IKEv2	128	NULL
gattuso	rfc4285	NULL	NULL

Figure 2-5: mipv6_profiles table

Furthermore, FreeRADIUS SQL queries require the presence of an additional table for each main table containing the name of the attribute to be transmitted and an operator (always “:=”).

attr_name	operator
eap_method	:=
credential_type	:=
credential_data	:=
services	:=

Figure 2-6: access_profiles_attributes table

attr_name	operator
auth_type	:=
psk_key_length	:=
keys_lifetime	:=

Figure 2-7: mipv6_profiles_attributes table

2.1.2 DHCPv6 with EAP

Some new DHCPv6 options have been developed in order to extend the behaviour of all the DHCPv6 agents (i.e. client, relay and server) to support MIPv6 capabilities. This allows the MN bootstrap when no EAP-extensions are available, as explained below.

2.1.2.1 Overview

In this scenario the access network is not provisioned with the EAP extensions which provide the MIPv6 parameters in bootstrapping. In this case the HA address is provisioned by using the DHCPv6 extensions as described in [draft-ietf-mip6-hiopt-02.txt]. Based on the relationships among the MASA, MSP and ASP, the HA can be assigned locally (i.e. in the ASP network), remotely (i.e. in the MASA domain) or in a third-party MSP domain. The details of these three cases are described in [ENA-D1.2], section 2.4. However, from the perspective of the software development description there are no significant differences from both the components architecture and the message flows. The DHCPv6 scenario has been summarized in the following figures below, which represent the case in which the HA is provided by the Home Provider Network (MASA acts as the MSP):

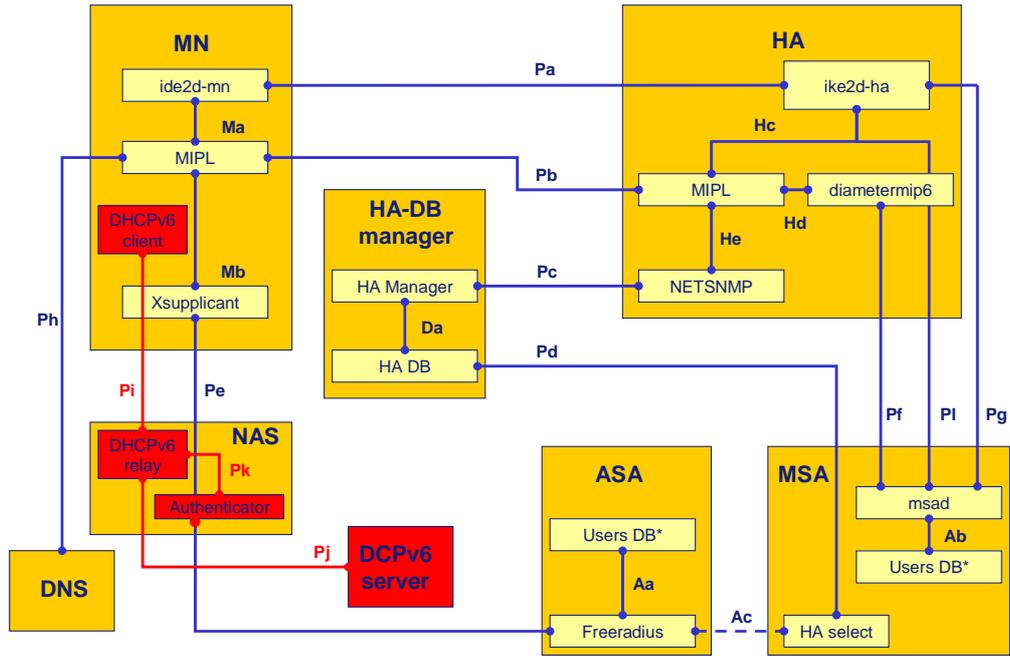


Figure 2-8: DHCPv6 components and interfaces in the reference architecture

As shown there are three DHCPv6 components:

- **DHCPv6 client:** It is installed in the MN and is in charge of requesting network parameters through the DHCPv6 protocol, i.e. DNS server address, HA address, etc.
- **DHCPv6 relay:** It is installed in the NAS and is responsible for relaying the DHCPv6 request messages from the DHCPv6 client to the DHCPv6 server.
- **DHCPv6 server:** It is a network component installed in the ASP domain for providing the network parameters requested by the DHCPv6 client.

All the involved components interact according to the following message flow:

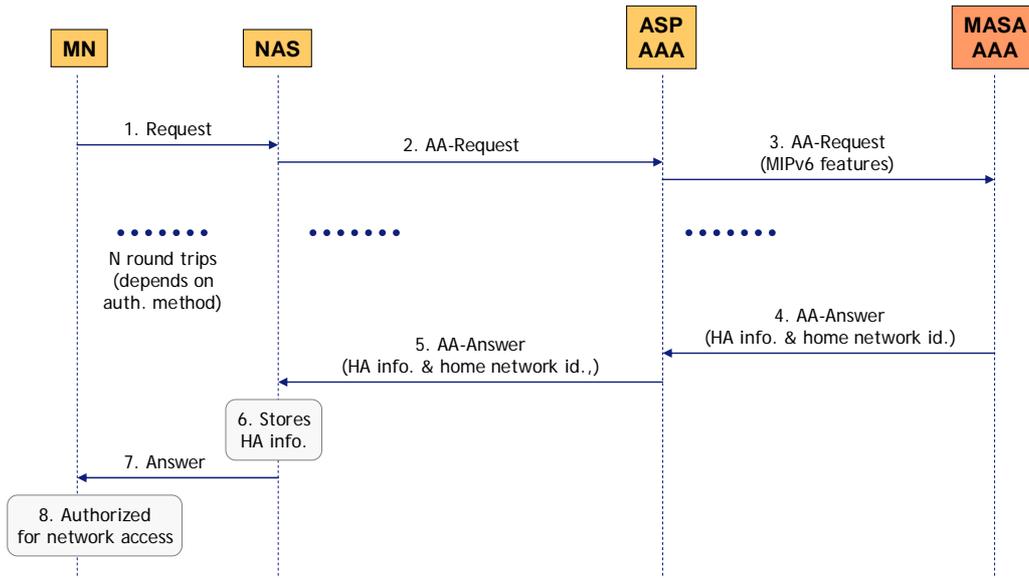


Figure 2-9: EAP not available, DHCPv6 available – message flow (1)

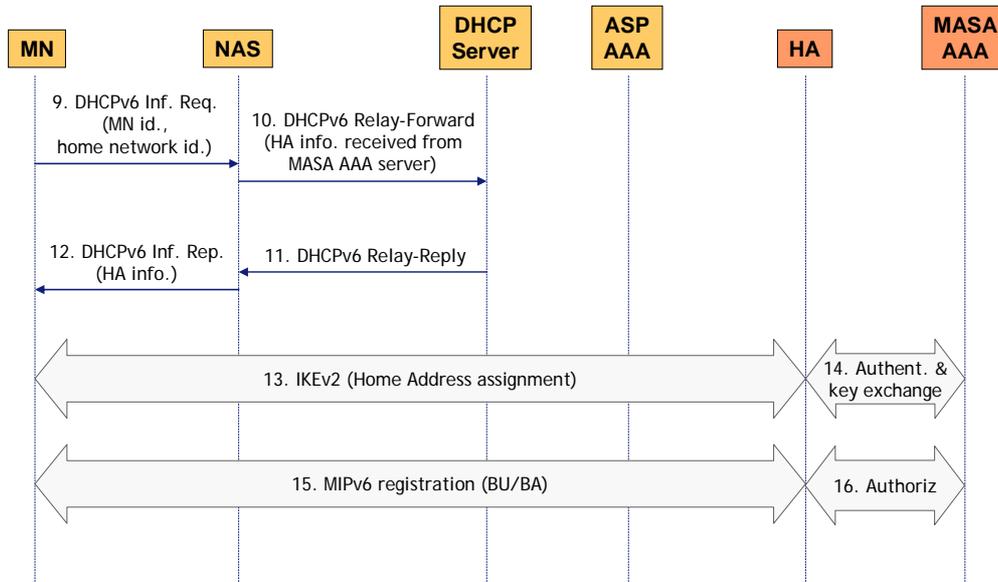


Figure 2-10: EAP not available, DHCPv6 available – message flow (2)

The most significant steps that are specific to this scenario are highlighted as follows:

- Step 6: After the MN is authorized to use the network access, the ASP/AAA sends the NAS the RADIUS Access-Accept message which includes a new AVP with the HA address assigned by the MASA to the MN. The HA is stored in the DHCPv6 relay (NAS).
- Step 9: After the MN realizes that it has been authorized to use the network, it runs the DHCPv6 client in order to get the HA address (and optionally other parameters like other IPv6 address, DNS server address, NTP server address, etc.). The DHCPv6 client includes the new Home Network Identifier Option defined in [draft-ietf-mip6-hiopt-02.txt] intended to ask for the HA address.
- Step 10: The DHCPv6 relay forwards the DHCPv6 message from the MN (DHCPv6 client) and it includes the HA address into the new MIP6 Relay Agent Option in order to let the DHCPv6 server know what the HA was assigned by the MASA.
- Step 11: The DHCPv6 server sends the replay with all the information required by the MN (i.e. DNS server, IPv6 address, etc.), including the HA address sent by the DHCPv6 relay. The HA address is sent by using the new Home Network Information Option.

The base code for developing the MIPv6 DHCPv6 extensions has been the DHCPv6 implementation [WIDE-DHCPV6] done by the [WIDE] project. Release 20061016 provides the three DHCPv6 components required in this scenario: client, relay and server. It has been modified in order to include the MIPv6 extensions required in this scenario.

2.1.2.2 Module NAS

The access point used in the DHCPv6 scenario is the WRT54G wireless router by Linksys because it allows the installation of applications developed by third-parties. To do that, it has been necessary to install a new firmware to replace the original one released by Linksys. The new firmware used is the Kamikaze_20061026_release developed by the [OpenWRT] project [<http://openwrt.org/>]. The new firmware makes the access router a true Linux system with a ssh server to logging into it.

In order to make any binary from the source code it is necessary to use the specific SDK developed for the access router, which includes the cross-compiler and other tools (such as binutils, kernel headers, libraries, scripts, etc.) that are required for compilation.

It has also been required to modify several configuration files and scripts on the access router in order to setup some network parameters, network applications and wireless security (i.e. 802.1X support) at start-up.

After these changes, the access router has been modified to work as a NAS based on 802.1X access and having the DHCPv6 relay with the MIPv6 extensions installed on it as explained in the next sections.

2.1.2.3 Module ASA/Freeradius and interfaces Pe and Pk

The interface Pe is used not only to allow the MN authenticate itself by providing its credentials but also to allow the ASA inform the NAS about the HA address assigned to the MN. This information is provided as a new AVP within the RADIUS Access-Accept message, as shown in the following figure.

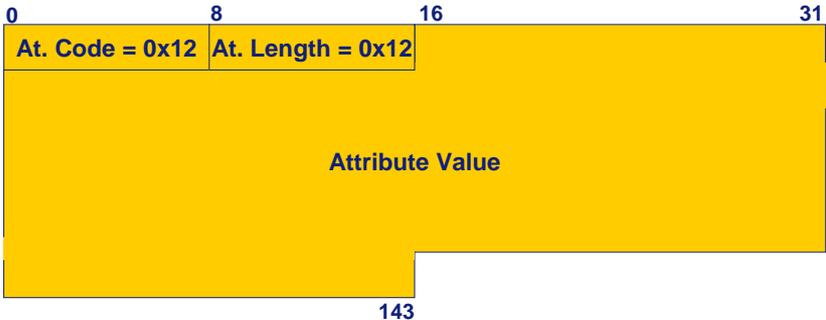


Figure 2-11: New AVP for the RADIUS Access-Accept message

The fields for this AVP are:

- Attribute: 0x12 to indicate that it is carrying the HA address.
- Length: 0x12 (18 bytes) which includes also the code and length fields
- Value: 32 ASCII characters, without colons, representing the HA address. The ASCII characters are on the wire as half bytes (each character as 4 bits).

After the RADIUS Access-Accept message is received, the network Authenticator module within the NAS picks out the HA address and sends it to the DHCPv6 relay through the interface Pk. Indeed it is implemented as a plain text file where both the MN’s MAC address and the assigned HA are stored together. The file is located under `/tmp/dhcp6r.conf` and in Figure 2-12 an example of its content is given.

```

mac 00:00:00:01:aa:bb ha 2001:db8:1000:1::444;
mac 00:06:29:30:a8:d4 ha 2001:db8:1000:1::555;
mac 00:00:00:01:aa:bb ha 2001:db8:1000:1::666;
mac 00:00:00:01:aa:bb ha 2001:db8:1000:1::777;
mac 00:c0:26:50:19:00 ha 2001:db8:1000:1::888;
mac 00:00:00:01:aa:bb ha 2001:db8:1000:1::999;
mac 00:0f:66:54:7d:df ha 2001:7f9:547:1000::3456:baba;
mac 00:40:d0:12:34:56 ha 2001:db8:1000:1::aaa;
mac 00:03:ff:16:34:56 ha 2001:db8:1000:1::bbb;
mac 00:90:96:7e:ba:17 ha 2001:7f9:547:1000::3456:baba;
mac 00:0f:20:94:97:d1 ha 2001:06b8:0020:0186:0000:0000:0000:0000:0ea1;

```

Figure 2-12: Example of file dhcp6r.conf for the DHCPv6 relay

2.1.2.4 Module DHCPv6 client and interface Pi

The DHCPv6 client developed by the WIDE project has been modified in order to carry the new Home Network Identifier Option in both the DHCPv6 Solicit and Request messages. This DHCPv6 messages are sent through the *Pi* interface.

The format of the new option is as specified in [draft-ietf-mip6-hiopt-02.txt]:

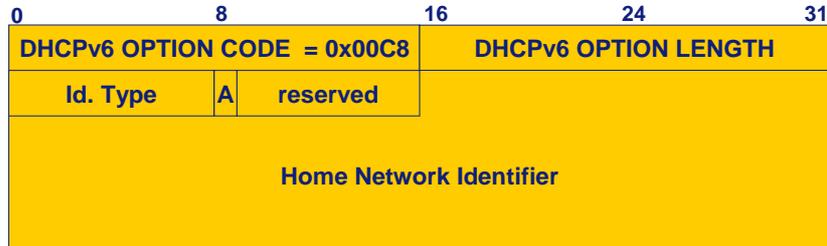


Figure 2-13: DHCPv6 Home Network Identifier Option

The fields for this new option are the following:

- Option code. The code for the new DHCPv6 option is set to 0x00C8 (200)
- Length: Total length of the option in octets.
- Id-type: The type of Home Network Identifier:
 - 0 Visited domain (local ASP)
 - 1 Home domain (home MSP)
 - 2 No preference
- A flag: This flag specifies whether the client requests a home address or not.
- Reserved: 7-bit field reserved for future use. The value is initialized to 0 by the sender and is ignored by the receiver.
- Home Network Identifier: The identifier to specify the requested home network of the MN. This field is set to the network realm as the FQDN when id-type is 0 or 1.

This new option let the DHCPv6 relay/server know that the MN is interested in receiving the HA address assigned to it.

Below is shown a true packet captured when the MN runs the DHCPv6 client:

```

Internet Protocol Version 6
  Version: 6
  Traffic class: 0x00
  Flowlabel: 0x00000
  Payload length: 74
  Next header: UDP (0x11)
  Hop limit: 64
  Source address: fe80::215:e9ff:fe4b:6e00 (fe80::215:e9ff:fe4b:6e00)
  Destination address: ff02::1:2 (ff02::1:2)
User Datagram Protocol, Src Port: 546 (546), Dst Port: 547 (547)
  Source port: 546 (546)
  Destination port: 547 (547)
  Length: 74 (bogus, should be 0)
  Checksum: 0xb734 [correct]
DHCPv6
  Message type: Solicit (1)
  Transaction-ID: 0x006fb909
  Client Identifier
    option type: 1
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: Ethernet (1)
    Time: 249213110
    Link-layer address: 00:15:e9:4b:6e:00
  Rapid Commit
    option type: 14
    option length: 0
  Elapsed time
    option type: 8
    option length: 2
    elapsed-time: 0 sec
  Option Request
    option type: 6
    option length: 4
    Requested Option code: Unknown (31)
    Requested Option code: Unknown (200)
DHCP option 200
  option type: 200
  option length: 22

```

Figure 2-14: Solicit DHCPv6 message with MIPv6 extensions

The DHCPv6 client is configured through the *dhcp6c.conf* file. The label used for carrying the Home Network Identifier Option in the Solicit/Requests DHCPv6 messages is *mip6-home-network-identifier* which must be inserted in the interface specification section as follows:

```
interface eth1 {
    request domain-name-servers;
    send rapid-commit;
    request mip6-home-network-identifier "mip6.consulintel.es";
    script "/usr/local/sbin/bootstrapping_script.sh";
};
```

Figure 2-15: Configuration of dhcp6c.conf file for the DHCPv6 client

The label *script* is used for running a command/application after a DHCPv6 reply is received.

The command syntax for running the DHCPv6 client is as follows:

```
dhcp6c [-c configfile] [-Ddfl] [-p pid-file] interface [interfaces...]
```

The command line options are:

```
-c    configfile
      Use configfile as the configuration file.
-d    Print debugging messages.
-D    Even more debugging information is printed.
-f    Foreground mode.
-I    Info-req mode.
      In this mode, stateless DHCPv6 is executed with an automatic
      configuration, and the obtained info is written to stdout. After
      this output, the program is terminated.
-p    pid-file
      Use pid-file to dump the process ID of dhcp6c.
```

Figure 2-16: Command syntax for the DHCPv6 client

2.1.2.5 Module DHCPv6 relay and interface Pj

The DHCPv6 relay has been modified in order to look for the Home Network Identifier Option in either the DHCPv6 Solicit or DHCPv6 Request messages, both received through the *Pi* interface. In the event that the option is found, the relay gets the MN's MAC address of the DHCPv6 Solicit/Request message and looks for the proper entry in the *dhcp6r.conf* file. Such a file was updated (interface *Pk*) with the RADIUS attributes that were received at the NAS from the ASA when the MN was authorized to use the access network, as explained above.

The relay builds the new MIP6 Relay Agent Option with such information in order to inform the DHCPv6 server about the MIPv6 information to include in the DHCPv6 replay through the *Pj*

interface. In the event that no HA address is found in the *dhcp6r.conf* file (interface *Pk*) then the relay does not build the MIP6 Relay Agent Option.

According to [draft-ietf-mip6-hiopt-02.txt], the format of the new option is as follows:

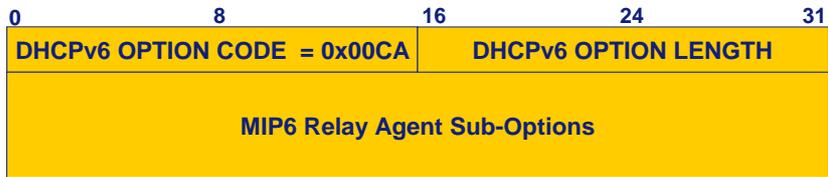


Figure 2-17: DHCPv6 MIP6 Relay Agent Option

The fields for this new option are the following:

- Option code. The code for the new DHCPv6 option is set to 0x00CA (202)
- Length: Total length of the option in octets.
- Sub-options: Different types of MIPv6 information might be received by the ASA (HA address, HoA, etc.). All this information is carried in different sub-options as follows.

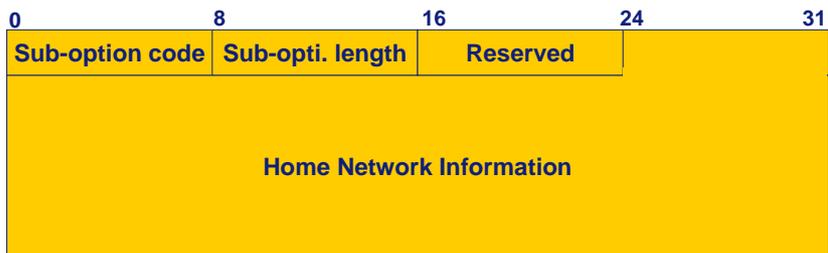


Figure 2-18: DHCPv6 MIP6 Relay Agent Sub-option

The fields for this new sub-option are the following:

- Sub-option code. It identifies the type of the following Home Network Information field. Possible values are:
 - 0 Home subnet prefix
 - 1 Complete IPv6 address of the HA
 - 2 FQDN of the HA
 - 3 IPv6 HoA
- Length: Total length of the following Home Network Information field.
- Reserved: 8-bit field reserved for future use. The value is initialized to 0 by the sender, and is ignored by the receiver.
- Home Network Information: A home subnet prefix, home agent IP address, FQDN or home address to be delivered to the DHCP server.

It is important to note that the ENABLE relay implementation builds the DHCPv6 MIP6 Relay Agent Option according to the format specified in [draft-ietf-mip6-hiopt-02.txt], but it always

configures the sub-option code to 1 because in the ENABLE tests the MN is only interested in receiving the HA address.

Below is shown a true packet captured when the DHCPv6 relay sends the DHCPv6 server the DHCPv6 Relay-Forward packet:

```
Internet Protocol Version 6
  Version: 6
  Traffic class: 0x00
  Flowlabel: 0x00000
  Payload length: 143
  Next header: UDP (0x11)
  Hop limit: 61
  Source address: 2a01:48:24:60::baba (2a01:48:24:60::baba)
  Destination address: 2a01:48:1:2:205:1cff:fe17:e495
(2a01:48:1:2:205:1cff:fe17:e495)
  User Datagram Protocol, Src Port: 546 (546), Dst Port: 547 (547)
  Source port: 546 (546)
  Destination port: 547 (547)
  Length: 143 (bogus, should be 0)
  Checksum: 0x2913 [correct]
DHCPv6
  Message type: Relay-forw (12)
  Hop count: 0
  Link-address: 2a01:48:24:80::baba
  Peer-address: fe80::215:e9ff:fe4b:6e00
  Relay Message
    option type: 9
    option length: 66
  DHCPv6
    Message type: Solicit (1)
    Transaction-ID: 0x006fb909
    Client Identifier
      option type: 1
      option length: 14
      DUID type: link-layer address plus time (1)
      Hardware type: Ethernet (1)
      Time: 249213110
      Link-layer address: 00:15:e9:4b:6e:00
    Rapid Commit
      option type: 14
      option length: 0
    Elapsed time
      option type: 8
      option length: 2
      elapsed-time: 0 sec
    Option Request
      option type: 6
      option length: 4
      Requested Option code: Unknown (31)
      Requested Option code: Unknown (200)
  DHCP option 200
    option type: 200
```

```

                option length: 22
Interface-Id
    option type: 18
    option length: 4
Interface-ID
DHCP option 202
    option type: 202
    option length: 19

```

Figure 2-19: Relay-Forward DHCPv6 message with MIPv6 extensions

The DHCPv6 relay is also in charge of sending both the DHCPv6 Advertise and DHCPv6 Reply messages according to the information received from the DHCPv6 server through the DHCPv6 Forward-Reply message. In this procedure there is nothing different to the behaviour that is standardized in [RFC3315].

The command syntax for running the DHCPv6 relay is as follows:

```

dhcp6relay [-c configfile] [-Ddf] [-b boundaddr] [-H hoplim]
           [-r relay-IF] [-s serveraddr] [-p pid-file] interface ...

```

The command line options are:

- c configfile
Use configfile as the file for storing the HA address received from the ASA.
- d Print debugging messages.
- D Even more debugging information is printed.
- f Foreground mode.
- b boundaddr
Specifies the source address to relay packets to servers (or other agents).
- H hoplim
Specifies the hop limit of DHCPv6 Solicit messages forwarded to servers.
- r relay-IF
Specifies the interface on which messages to servers are sent. When omitted, the same interface as interface will be used. When multiple interface are specified, this option cannot be omitted.
- s serveraddr
Specifies the DHCPv6 server address to relay packets to. If not specified, packets are relayed to ff05::1:3 (All DHCPv6 servers).
- p pid-file
Use pid-file to dump the process ID of dhcp6relay.

Figure 2-20: Command syntax for the DHCPv6 relay

2.1.2.6 Module DHCPv6 server and interface Pj

The DHCPv6 server has been modified to support the MIPv6 DHCPv6 extensions. In this way, the DHCPv6 Relay-Forward message sent by the relay includes the MIP6 Relay Agent Option. The server builds the DHCPv6 Relay-Replay message and inserts the new Home Network Information Option with the information received in the MIP6 Relay Agent Option.

According to [draft-ietf-mip6-hiopt-02.txt], the format of the new option is as follows:

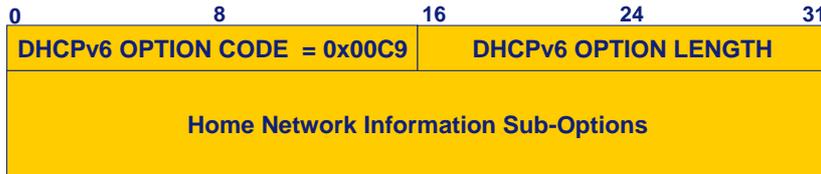


Figure 2-21: DHCPv6 Home Network Information Option

The fields for this new option are the following:

- Option code. The code for the new DHCPv6 option is set to 0x00C9 (201)
- Length: Total length of the option in octets.
- Sub-options: Different types of MIPv6 information might be received from the DHCPv6 relay (HA address, HoA, etc.). All this information is carried in different sub-options as follows.



Figure 2-22: DHCPv6 Home Network Information Sub-option

The fields for this new sub-option are the following:

- Sub-option code. It identifies the type of the following Home Network Information field. Possible values are:
 - 0 Home subnet prefix
 - 1 Complete IPv6 address of the HA
 - 2 FQDN of the HA
 - 3 IPv6 HoA
- Length: Total length of the following Home Network Information field.
- V flag: This flag specifies whether the information is assigned by the visited network or not.
- Reserved: 7-bit field reserved for future use. The value is initialized to 0 by the sender, and is ignored by the receiver.
- Home Network Information: A home subnet prefix, home agent IP address, FQDN or home address to be delivered to the DHCP server. This information is provided by the DHCPv6 relay through the MIP6 Relay Agent Sub-option.

Below is shown a true packet captured when the DHCPv6 server sends the DHCPv6 relay the DHCPv6 Relay-Replay packet:

```
Internet Protocol Version 6
  Version: 6
  Traffic class: 0x00
  Flowlabel: 0x00000
  Payload length: 266
  Next header: UDP (0x11)
  Hop limit: 64
  Source address: 2a01:48:1:2:205:1cff:fe17:e495
                  (2a01:48:1:2:205:1cff:fe17:e495)
  Destination address: 2a01:48:24:60::baba (2a01:48:24:60::baba)
User Datagram Protocol, Src Port: 34752 (34752), Dst Port: 547 (547)
  Source port: 34752 (34752)
  Destination port: 547 (547)
  Length: 266 (bogus, should be 0)
  Checksum: 0x9ad0 [correct]
DHCPv6
  Message type: Relay-reply (13)
  Hop count: 0
  Link-address: 2a01:48:24:80::baba
  Peer-address: fe80::215:e9ff:fe4b:6e00
  Relay Message
    option type: 9
    option length: 212
    DHCPv6
      Message type: Advertise (2)
      Transaction-ID: 0x006fb909
      Client Identifier
        option type: 1
        option length: 14
        DUID type: link-layer address plus time (1)
        Hardware type: Ethernet (1)
        Time: 249213110
        Link-layer address: 00:15:e9:4b:6e:00
      Server Identifier
        option type: 2
        option length: 14
        DUID type: link-layer address plus time (1)
        Hardware type: Ethernet (1)
        Time: 249155216
        Link-layer address: 00:05:1c:17:e4:95
      DNS recursive name server
        option type: 23
        option length: 32
        DNS servers address: 2001:7f9:1000:1::103
        DNS servers address: 2001:7f9:1000:1::947c
    DHCP option 201
      option type: 201
```

```
      option length: 19
    Domain Search List
      option type: 24
      option length: 89
    DNS Domain Search List
      Domain: dns1.novagnet.com
      Domain: ns1.euro6ix.org
      Domain: dns1.consulintel.com
    DHCP option 31
      option type: 31
      option length: 16
  Interface-Id
    option type: 18
    option length: 4
  Interface-ID
```

Figure 2-23: Relay-Replay DHCPv6 message with MIPv6 extensions

The DHCPv6 Advertise message with the Home Network Information Option resulting from that DHCPv6 Relay-Replay message (in the DHCPv6 relay, interface *Pi*) would be the following:

```
Internet Protocol Version 6
  Version: 6
  Traffic class: 0x00
  Flowlabel: 0x00000
  Payload length: 220
  Next header: UDP (0x11)
  Hop limit: 64
  Source address: fe80::290:4cff:fe5f:2a (fe80::290:4cff:fe5f:2a)
  Destination address: fe80::215:e9ff:fe4b:6e00
                      (fe80::215:e9ff:fe4b:6e00)
User Datagram Protocol, Src Port: 547 (547), Dst Port: 546 (546)
  Source port: 547 (547)
  Destination port: 546 (546)
  Length: 220 (bogus, should be 0)
  Checksum: 0xd957 [correct]
DHCPv6
  Message type: Advertise (2)
  Transaction-ID: 0x006fb909
  Client Identifier
    option type: 1
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: Ethernet (1)
    Time: 249213110
    Link-layer address: 00:15:e9:4b:6e:00
  Server Identifier
    option type: 2
    option length: 14
    DUID type: link-layer address plus time (1)
    Hardware type: Ethernet (1)
    Time: 249155216
    Link-layer address: 00:05:1c:17:e4:95
  DNS recursive name server
    option type: 23
    option length: 32
    DNS servers address: 2001:7f9:1000:1::103
    DNS servers address: 2001:7f9:1000:1::947c
DHCP option 201
  option type: 201
  option length: 19
  Domain Search List
    option type: 24
    option length: 89
  DNS Domain Search List
  Domain: dns1.novagnet.com
  Domain: ns1.euro6ix.org
  Domain: dns1.consulintel.com
```

```

DHCP option 31
  option type: 31
  option length: 16

```

Figure 2-24: Advertise DHCPv6 message with MIPv6 extensions

The DHCPv6 server can be configured through the `dhcp6s.conf` file with a default HA address to be used in case no MIPv6 Relay Agent Option is received. This forces the build of a DHCPv6 Relay-Reply message with that HA address in the Home Network Information Option. The label used for configuring such a HA address is `mip6-hninf-ha-address` which must be inserted out of the interface specification section as follows:

```
option mip6-hninf-ha-address 2001:7f9:1000:1::444;
```

The command syntax for running the DHCPv6 server is as follows:

```

dhcp6s [-c configfile] [-Ddf] [-k ctlkeyfile] [-p ctlport]
      [-P pid-file] interface

```

The command line options are:

```

-c    configfile
      Use configfile as the configuration file.
-d    Print debugging messages.
-D    Even more debugging information is printed.
-f    Foreground mode.
-k    ctlkeyfile
      Use ctlkeyfile to store the shared secret to authenticate the
      communication with dhcp6sctl. The default file name used when
      unspecified is /usr/local/etc/dhcp6sctlkey. The default name is
      intentionally same as that for dhcp6sctl so that the server and
      the control command can share the file when dhcp6sctl controls
      the server on the same node, which should be the typical case.
-p    ctlport
      Use ctlport as the port number listening on to communicate with
      dhcp6sctl.
-P    pid-file
      Use pid-file to dump the process ID of dhcp6s.

```

Figure 2-25: Command syntax for the DHCPv6 server

2.1.3 AAA for MIPv6

When bootstrapping MIPv6 two differentiated scenarios may be presented, each one having special requirements that must be kept in mind.

- Integrated scenario: the MSA and the ASA are the same entity.
- Split scenario: the MSA and the ASA are separated entities.

In the integrated scenario, the MSA + ASA (MASA) controls the entire bootstrapping procedure, so it can provide mobility configuration parameters piggybacked on the network authentication process. In this scenario there are two different possibilities to provide the HA address to the MN: the MASA can deliver the HA directly within the EAP tunnel (if the MN access network allows it) or it can be delivered via DHCPv6.

In the split scenario, the ASA does not know anything about mobility so the MN must discover the HA address using DNS queries.

Once the HA address is known by the MN, the rest of the bootstrapping steps are the same in both scenarios. First, the MN needs to authenticate with the HA, obtain a HoA and establish the needed security associations (SAs) to protect the mobility signalling. All these actions are performed by using the IKEv2 protocol. The authentication is performed in one of these two ways:

1. IKEv2 and EAP. In the most general scenario the MN needs to authenticate with the MSA using an EAP method. The HA acts as a pass-through authenticator, forwarding the EAP packets from/to the MN (transported by IKE_AUTH messages) to/from the MSA (transported by the Diameter EAP Application).
2. IKEv2 and PSK optimization. When the integrated scenario is being used and the MN uses EAP to be authenticated in the network, the MN can use a key derived from the EAP keying material to authenticate with the IKEv2 protocol (instead of performing a full EAP method). The HA would need to retrieve the key from the MASA server, which is able to generate exactly the same key as the MN does. This allows for savings on the roundtrips needed to perform an EAP authentication.

After that, the MN can then send a Binding Update to the HA. The IKEv2 protocol only authenticates the MN, so the mobility service must be explicitly authorized by the MSA upon reception of this first BU (a new experimental authorization application is defined for this purpose). In order to perform the authorization of the mobility service, the HA sends a Diameter MIP6-Authorization-Request (MAR) to the MSA AAA server. This message has the Auth-

Request-Type AVP set to AUTHORIZE_ONLY, and includes at least a User-Name AVP containing the identity that will be used to authorize the mobility service on behalf of the MN.

After checking the MN's profile, the MSA-AAA replies with a Diameter MIP6-Authorization-Answer which contains a Result-Code AVP with the authorization decision. This message may also contain additional AVPs to enforce specific policies for the mobility service. The HA can then send a BA to the MN, completing the bootstrapping process.

The involved software modules and interfaces are shown in the Figure 2-26 (in red colour).

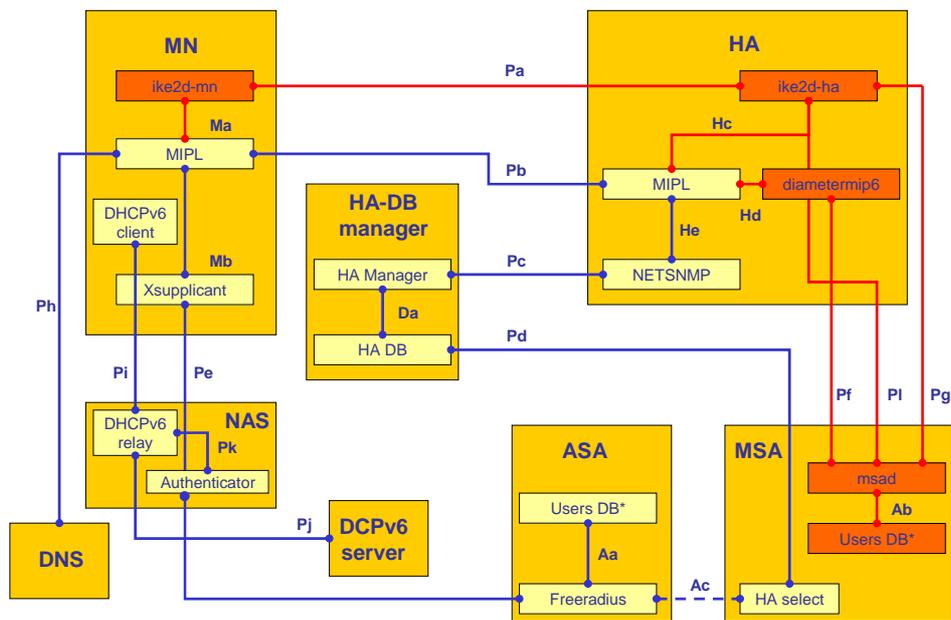


Figure 2-26 AAA for MIP6 software modules and interfaces in the reference architecture

2.1.3.1 Module ike2d-mn (MN)

The MN needs to perform IKEv2 exchanges to create the IPsec security associations between HA and MN, to obtain the assigned HoA and to get authenticated for MIPv6 service. A software module called ike2d-mn has been developed to provide this functionality. This module makes use of the libopenikev2 and libopenikev2_impl libraries, but including some modifications to accomplish the additional ENABLE behaviour:

- A new EAP client controller has been developed in order to support client authentication via EAP TLS.

- The XFRM Ipsec controller has been improved in order to manage correctly the selectors including the Ipv6 Mobility Header.

This new daemon has the following behaviour:

- When the ike2d-mn starts, it keeps waiting on the interface Ma until a “HoA Request” message is received from the MIPL daemon.
- When a “HoA Request” message is received, the needed IKEv2 exchanges are performed (using the Pa interface) in order to authenticate the MN, obtain the HoA and create the needed security associations.
- Once all the IKEv2 exchanges have been performed, ike2d-mn sends a “HoA response” message to the MIPL daemon containing the new assigned address.

MIPL and ike2d-mn communicates by the use of the Ma interface. Ma is an inter process interface implemented using a UDP socket, used to trigger the ike2d-mn daemon in order to perform the IKEv2 exchanges. Within this trigger message the MIPL daemon communicates the IPv6 and IPv4 Home Agent addresses (IPv4 HA address is included in case of Interworking enabled functionalities, see section 2.1.5.1.2), addresses previously obtained (through EAP or DNS) with which the MN has to perform the IKEv2 negotiation. Additionally, the MIPL daemon can also indicate to ike2d-mn the PSK to be used when the IKEv2-PSK optimization is desired. Within the response ike2d-mn communicates to MIPL the Home Address (HoA) assigned to the MN during the IKEv2 exchange. The hypothesis is that the ike2d-mn is started before MIPL.

The MIPL Configuration message has the following format (Figure 2-27).

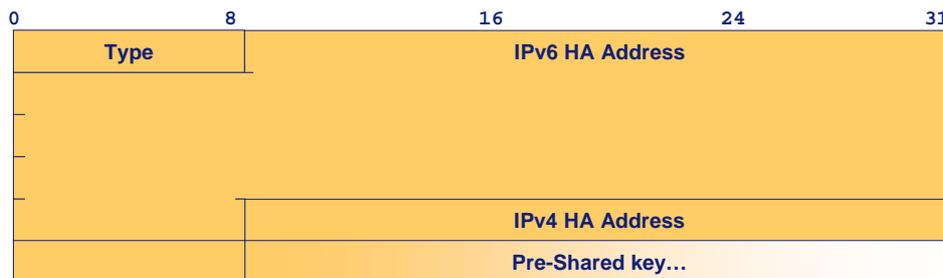


Figure 2-27 MIPL Configuration message

The Type field is one octet, and indicates the type of MIPL Configuration message. The types used by this interface are:

- 4 – HoA Request: sent by MIPL to OpenIKEv2; it triggers the IKEv2 exchange with the specified HA.

- Address = HA address.
- Pre-shared key = Pre-shared key to be used (only if IKEv2-PSK optimization is selected).
- 5 – HoA Reply: sent by OpenIKEv2 to MIPL and used to deliver the HoA.
 - Address = HoA.
 - Pre-shared key = Not included

The rules that the MIPL and the OpenIKEv2 modules have to observe are:

- As soon as MIPL daemon knows the HA addresses, it sends to OpenIKEv2 the HoA Request inserting the just obtained HA addresses and the PSK to be used if IKEv2-PSK optimization is desired.
- When the OpenIKEv2 daemon receives the HoA Request it starts the IKEv2 exchange with the suitable HA address (the IPv6 one if the MN is connected to an IPv6 network, the IPv4 one vice versa) taken from the Request.
- When the OpenIKEv2 daemon obtains the HoA it sends the HoA Reply message to the MIPL daemon.
- When MIPL daemon receives the HoA Reply it sets up the Secure Policy Database (SPD) using the HoA received and sends the first BU.

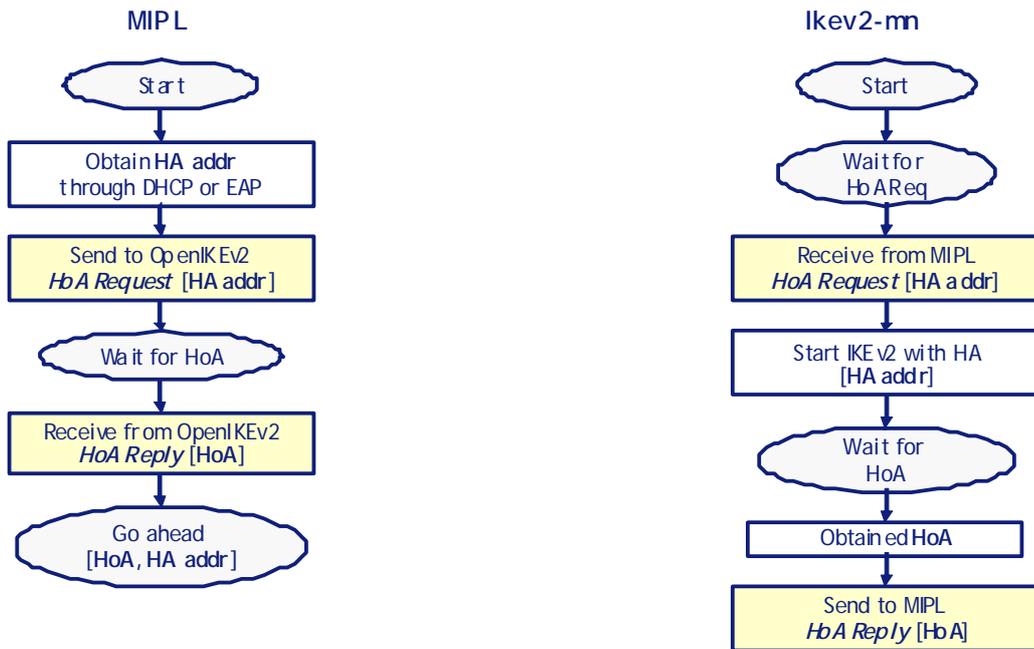


Figure 2-28 Interaction between MIPL and ike2d-mn

The ike2d-mn daemon communicates with the HA over the Pa interface. Pa is a network interface between the MN and the HA using the IKEv2 protocol. The process has the following steps, illustrated in Figure 2-29 (note that the figure only depicts the case with EAP authentication):

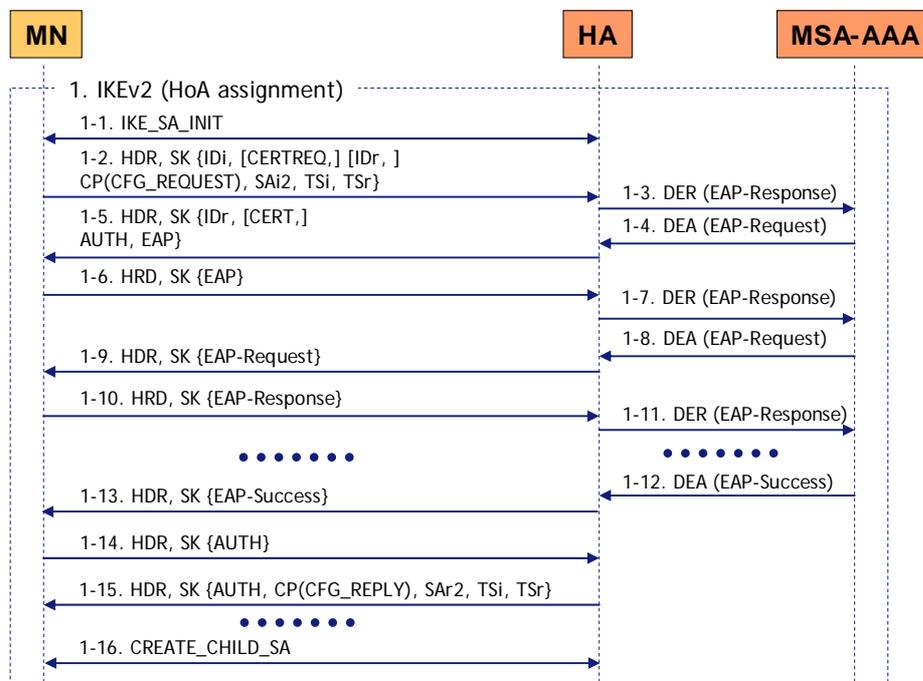


Figure 2-29 IKEv2 exchanges (EAP authentication)

- The ike2d-mn client performs an IKE_SA_INIT exchange with the HA, to derive cryptographic material for the IKE SA (step 1-1)
- The ike2d-mn client sends the IKE_AUTH request message including its identity, the desired authentication method (EAP or PSK) and the request for the assignation of a remote internal address (in the CP payload) (step 1-2).
- If EAP authentication is selected, the HA's OpenIKEv2 server acts as an EAP pass-through, forwarding the EAP packets between the MN and the AAA server and vice-versa, using the interface Pg and IKE_AUTH messages for EAP transport (steps 1-3 to 1-13).
- If PSK authentication is selected, then the HA's OpenIKEv2 server tries to retrieve the PSK to be used from the MSA-AAA, using the PI interface.
- Once the MN is authenticated, the HA's OpenIKEv2 server obtains a HoA from the internal address pool, stores it in the HoAs file using the Hc interface and sends it to the MN (using the CP payload) in the last IKE_AUTH message. As result of the whole IKE_AUTH exchange, the first Ipsec SA is established (step 1-15)
- After that, OpenIKEv2 creates all the remaining Ipsec Sas needed to protect the MIPv6 signalling and the traffic between the MN and the HA. In order to do this, the MN OpenIKEv2 client initiates all the needed CREATE_CHILD_SA exchanges with the HA (step 1-16).
- The two OpenIKEv2 daemons keep working in order to maintain the Ipsec Sas (rekeyings, deletions...).

2.1.3.2 Module ike2d-ha (HA)

The HA needs to perform IKEv2 exchanges with the MN:

- To assign the HoA
- To authenticate the MN for MIPv6 service.

This is in order to create the IPsec security associations.

A software module called ike2d-ha has been developed to provide this functionality. This module makes use of the libopenikev2 and libopenikev2_impl libraries, but including some modifications to accomplish the additional ENABLE behaviour:

- A new address configuration method has been implemented in order to obtain the addresses to be assigned from a special file (called NAI_HoAs file).
- A new EAP server controller has been developed in order to implement an EAP pass-through authenticator.
- The method to obtain the PSK has been modified in order to retrieve it from the MSA-AAA server.
- The XFRM IPsec controller has been improved in order to correctly manage the selectors including the IPv6 Mobility Header.
- The standard behaviour when assigning a new address has been modified since there is no need to install any dynamic policy in the SPD (MIPL has done this already).

This module communicates with the MN by using the interface Pa (already described in the previous paragraph). In order to perform the MN authentication, the daemon uses the interfaces Pg (for EAP authentication) and Pl (for PSK authentication) with the MSA-AAA server.

Pg is a network interface between the HA and the MSA-AAA using the Diameter EAP Application. This interface is used to perform the MN authentication only (since the authorization is initiated by MIPL upon receiving a Binding Update). There are three involved roles: the User (ike2d-ha), the NAS (ike2d-ha) and the Server (msad).

The HA starts the application by sending a Diameter-EAP-Request (DER) message containing an Identity EAP-Response with the MN identity (extracted from the IKEv2 IDi payload). After that, the HA acts as an EAP pass-through, forwarding the EAP packets from the MN to the MSA-AAA and the other way around. The EAP packets between the MN and the HA are transported using EAP payloads into IKE_AUTH messages. The EAP packets between the MN and the MSA-AAA are transported using EAP-Payload AVPs into Diameter-EAP-Request/Diameter-EAP-Response messages.

The EAP authentication may take more than one roundtrip, depending on the used EAP method. In addition, the selected EAP method might generate an EAP-Master-Session-Key as result of a successful authentication. When this key is generated, OpenIKEv2 (in both MN and HA) will use this key to sign the AUTH payload in the last messages of the IKE_AUTH exchange.

Pl is a network interface between the HA and the MSA-AAA using the NASREQ Diameter Application. This interface is used to retrieve the MN pre-shared key from the MSA-AAA, in order to authenticate the user.

The message flow for the case of IKEv2 in PSK mode is depicted in Figure 2-30. The MN accesses the network and performs EAP authentication (step 1); then, it derives a key for IKEv2 shared with its AAA server (step 2).

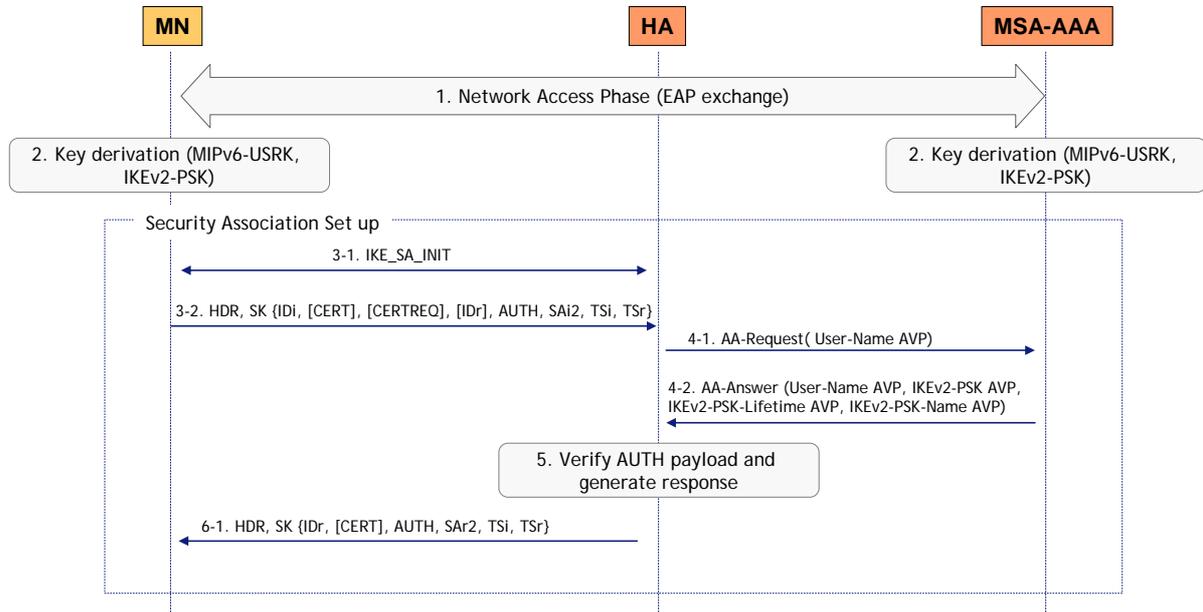


Figure 2-30 Message flow for IKEv2 in PSK mode

The MN and the HA start the interaction with the IKE_SA_INIT exchange (step 3-1) used to negotiate cryptographic algorithm, nonces and Diffie-Hellman parameters. The next phase (IKE_AUTH) authenticates the previous messages and exchanges identities and certificates. In this phase MN and HA mutually authenticate each other. In the first message of this phase (step 3-2) the MN inserts the AUTH payload calculated with the shared key derived from EAP. The HA needs to retrieve the key from the AAA server in order to verify this payload. To this end it sends to the AAA server a AA-Request message [RFC4005] with Auth-Request-Type set to AUTHENTICATE_ONLY since this exchange will only authenticate the MN and does not authorize the MIPv6 service. The HA must insert in the AA-Request message the User-Name AVP filled with the IDi identity from the IKEv2 message.

The AAA server receives the request and, based on the given identity, retrieves the IKEv2 pre-shared key previously derived from EAP. It then sends the corresponding AA-Answer (steps 4-1 and 4-2) containing the requested key (IKEv2-PSK AVP), the corresponding lifetime and key name (IKEv2-PSK-Lifetime AVP and IKEv2-PSK-Name AVP) and the identity (User-Name AVP) that the HA must use to send the subsequent MIPv6 Authorization Requests (MAR), so that the MSA AAA server can map the request to the right MN.

The HA, having the key, can verify the AUTH payload and generate the corresponding response terminating the IKE_AUTH exchange (step 6-1).

The ike2d-ha daemon also communicates with the MIPL module, by the use of the interface Hc. This interface is implemented using a shared file (NAI-HoAs). The file is a text file composed by two columns one containing the HoA and one that may contain the user's NAI, each row represents an assignable HoA. If a NAI is present in the second column it means that the HoA has been assigned.

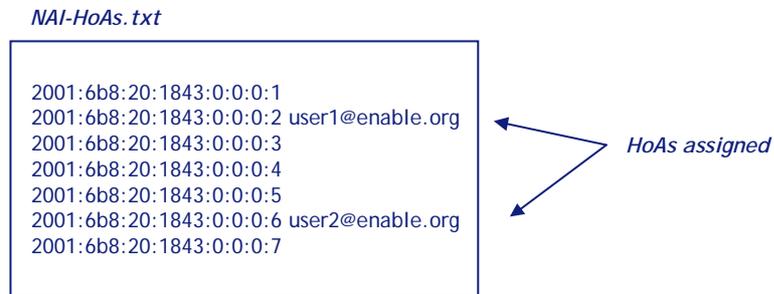


Figure 2-31 NAI-HoAs file example

When MIPL is started on the HA, it reads the standard configuration file which specifies the location of the NAI-HoAs file. After this reading operation of the configuration file, the MIPL daemon reads the NAI-HoAs file in order to initialise the SPD. MIPL has to insert in the SPD an entry for each assignable HoA.

The same file (NAI-HoAs) is used by MIPL to obtain the NAI associated to the HoA received within a BU. The MIPL module searches into the NAI-HoA file to find the NAI associated to the HoA received.

When ike2d-ha needs to assign a HoA, it reads the NAI-HoAs file looking for an unassigned HoA (a HoA is unassigned if its second column is empty). If any free HoA is found, ike2d-ha assigns the HoA and writes the MN NAI in its second column to ensure that MIPL can map the HoA with the NAI in order to authorise the mobility service for that MN.

Rules for the access to the file:

- When ike2d-ha needs to assign a HoA, it reads the NAI-HoAs file looking for an unassigned HoA. If any free HoA is found, ike2d-ha writes the MN NAI into its second column.
- When MIPL receives the first Binding Update from a new user, it search the NAI associated to the received HoA within the NAI-HoA file.
- When the MIPL recovers the NAI, it starts the user authorization using that NAI.

- The NAI is stored within the new BC entry created in order to be able to re-authenticate the user without accessing again to the NAI-HoA file.

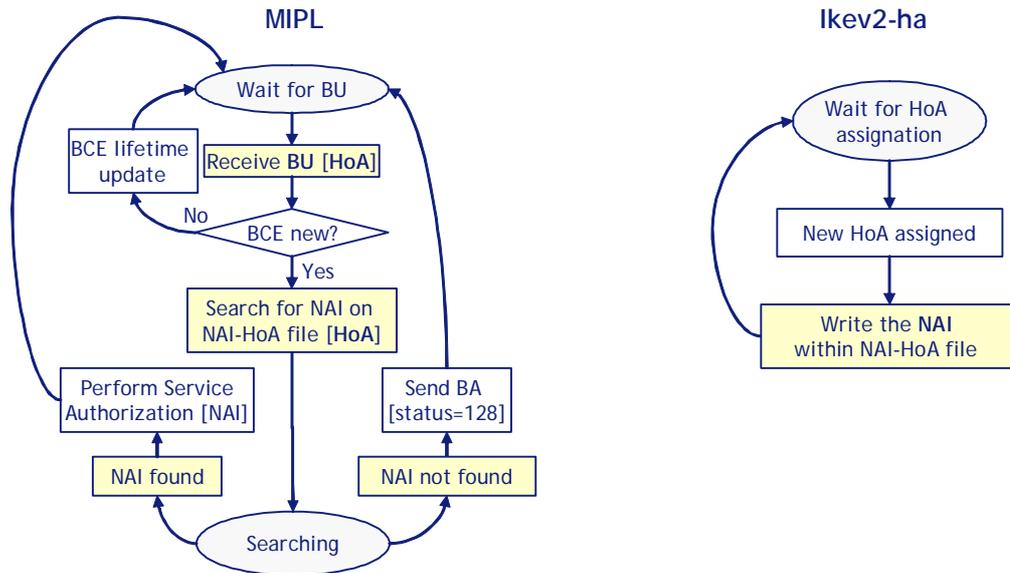


Figure 2-32 Interaction between MIPL and ikev2d-ha

2.1.3.3 Module diamettermip6 (HA)

This module, called diamettermip6, consists of an OpenDiameter peer intended to perform the MIPv6 service authorization on behalf of MIPL module. It is needed because OpenDiameter libraries cannot be directly used with MIPL, since they are written in C++ code but MIPL is written in C.

In order to communicate with the MIPL module it uses the Hd interface. Hd is an inter process interface implemented using a UDP socket.

Messages defined for this interface are:

- User Authorization Request (Figure 2-33): message sent by MIPL to OpenDiameter client for Mobile User's authorization of MIP6 service.

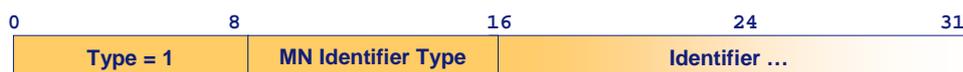


Figure 2-33 User Authorization Request

- o MN Identifier Type = 1 (NAI).

- Identifier = NAI.
- User Authorization Reply (Figure 2-34): message sent by OpenDiameter client to MIPL, communicates the result of the User's authorization of MIP6 service

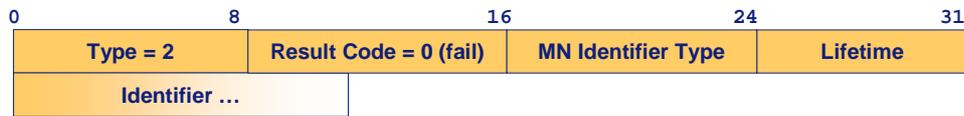


Figure 2-34 User Authorization Reply

- Result Code = 1 (on success), 0 (on failure).
- MN Identifier Type = 1 (NAI).
- Identifier = NAI.
- Lifetime = user authorization lifetime.

To perform the MIPv6 service authorization, the `diametermip6` module uses the Pf interface to communicate with the MSA-AAA server. This interface is based on DIAMETER protocol. The two peers are the Home Agent, which acts as Diameter client, and the Home AAA Server (MSA) that acts as the Diameter server. On the Home Agent, the Diameter application is triggered by the reception of the first Binding Update message received from a MN: when the BU arrives, the Diameter client asks the MSA server whether the user is authorised for the MIPv6 service or not. If authorisation succeeds, the Home Agent sends a Binding Acknowledgement with status 0 (Binding Update accepted) to the mobile user; otherwise, if authorisation fails, the Home Agent issues a Binding Acknowledgement message with status code 129 (Administratively prohibited).

The Diameter application requests and answers have the `Auth-Request-Type` AVP set to `AUTHORIZE_ONLY`. Some mandatory AVPs have been defined for request messages:

- `UserName` AVP to carry the user identity (NAIs)
- `MN-HomeAddress` AVP to inform the AAAH server of the binding Home Address of the user to allow DNS updates (made by AAAH on behalf of HA to fit also scenarios in which HA doesn't have a security association with user's home domain DNS).

Answer messages contain the result code (success or failure) and authorisation AVPs (at least the authorisation lifetime).

The behaviour of these modules is the following:

- When MIPL receives the first Binding Update from a new user, MIPL obtains the NAI associated to the received HoA from the NAI-HoA file. MIPL sends a User Auth. Req. to diamettermip6 inserting the obtained mobile node's NAI into MN_ID_NAI field.
- When diamettermip6 receives a User Auth. Req., it performs user authorization contacting a backend AAA server:
 - 1. In case of success diamettermip6 sends to MIPL a User Auth. Rep. with result_code set to 1 and the NAI of authorized user into MN_ID_NAI field to let MIPL match this success message with the pending request. The MIPL stores the NAI within the BC.
 - 2. In the case of authorization failure, diamettermip6 sends back to MIPL a User Auth. Rep. message with result_code 0.
- When the authorization timer for a user's Binding Cache Entry (BCE) expires, MIPL sends a User Auth. Req. to diamettermip6 with user's NAI, previously obtained and stored within the BCE, into MN_ID_NAI field to re-authorise the user; then Open Diameter sends back a User Auth. Rep. message with the answer of the Diameter server.
- When BCE lifetime expires the MIPL delete the BCE.

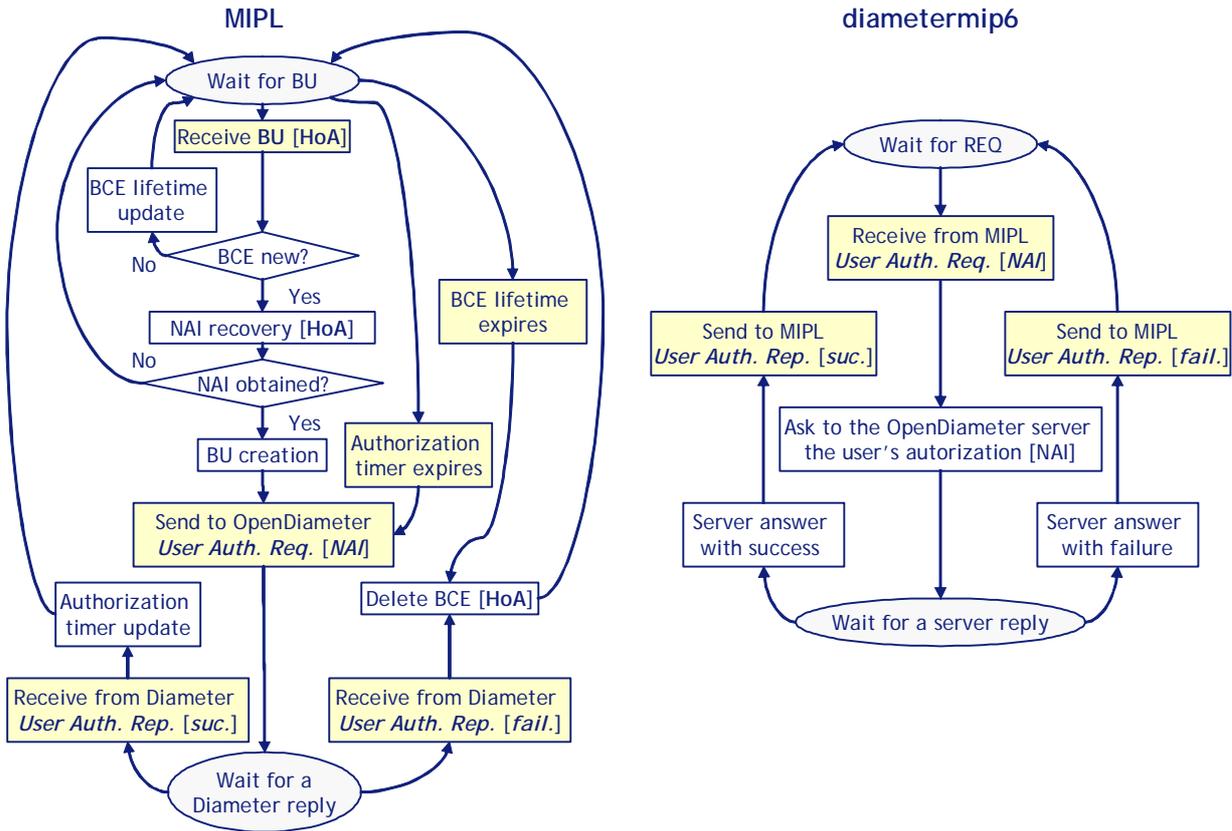


Figure 2-35 Interaction between MIPL and diametermip6

2.1.3.4 Module msad (MSA)

All the Open Diameter code used in ENABLE is based on Open Diameter 1.0.7-h. This release contains the following libraries:

- Framework. Contains the general state machine framework classes, as well as tasks, jobs, queues and other classes used by the base protocol.
- Libdiamparser. This library contains the classes used to parse diameter messages, including headers and AVPs.
- Libdiameter. This is the library that implements the Diameter Base Protocol.
- Libdiameternasreq. Implements the Diameter NAS application. This library is used together with libdiametereap and libeap to perform the MIPv6 authentication and authorization and to integrate Open Diameter within Openikev2.
- Libdiametereap. Implements the Diameter EAP application.

- Libeap. Implements the EAP state machines and EAP messages, using functionality from the OpenSSL library.

A customized diameter daemon has been developed in order to integrate Open Diameter with OpenIKEv2. The MSA Open Diameter server (msad) takes care of the EAP-based authentication (interface Pg), pre-shared key providing when integrated scenario is being used (interface Pl) and MIPv6 service authorization (interface Pf).

In order to do this, two different databases are deployed:

- OpenDiameter IKEv2 DB: used to authenticate the user and to authorise the SA creation between the HA and the MN.
- OpenDiameter MIPv6 DB: used to authorise the MIPv6 service.

The OpenDiameter MIPv6 DB and OpenDiameter IKEv2 DB are implemented through the same XML file containing users' profile for MIPv6 authentication and authorization. This file is named "aaa_user_db.xml" and it is compliant with XML version 1.0 W3C recommendation [W3C]. An XSD file is associated to that file, named "aaa_user_db.xsd", containing the schema of the XML file. The msad daemon uses the Ab interface to store and retrieve data from the MIPv6/IKEv2 DBs. Currently this interface is based on direct file access to the "aaa_user_db.xml" file, although it may be modified in the future to use a TCP socket in order to fetch the data from a MySQL database via SQL queries

Below there is an example of the user DB XML file including only the sub elements used by the authorization application since the entries for EAP authentication are the same ones used by the standard OpenDiameter EAP application.

aaa_user_db.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<user_db xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation='aaa_user_db.xsd'>

  <user_entry>
    <name_match>luca.battistoni@telecomitalia.it</name_match>
    <authz_lifetime>60</authz_lifetime>
  </user_entry>

  <user_entry>
    <name_match>michele.lamonaca@telecomitalia.it</name_match>
    <authz_lifetime>100</authz_lifetime>
  </user_entry>

</user_db>
```

Figure 2-36 Example of user DB XML file

The main element (users' database) is bounded by tags <user_db> and </user_db>. The users DB is divided into sub elements, one for each the user's entry (<user_entry> ... </user_entry>). The user entry contains two sub elements:

- name_match: this element contains the NAI of the user.
- authz_lifetime: this element stores the lifetime associated with the user authorization, the lifetime is expressed in seconds.

The XSD file associated to the XML user DB file is shown below.

aaa_user_db.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="user_entry_type">
    <xs:sequence>
      <xs:element name="name_match" type="xs:string" />
      <xs:element name="authz_lifetime" type="xs:integer" />
    </xs:sequence>
  </xs:complexType>

  <xs:element name="user_db">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="user_entry"
          type="user_entry_type"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figure 2-37 Example of XML schema file

2.1.4 Home Agent load sharing

The HA load sharing process developed in ENABLE is described in detail in [ENA-D1.2], section 3, and [ENA-D6.1], section 3.2.3. In this section, the focus is the HA load sharing prototype implementation.

2.1.4.1 Overview

Figure 2-38 illustrates the message flow diagram of the HA load sharing mechanism. The HA-DB Manager entity combines two entities: the HA-Manager which is responsible for retrieving

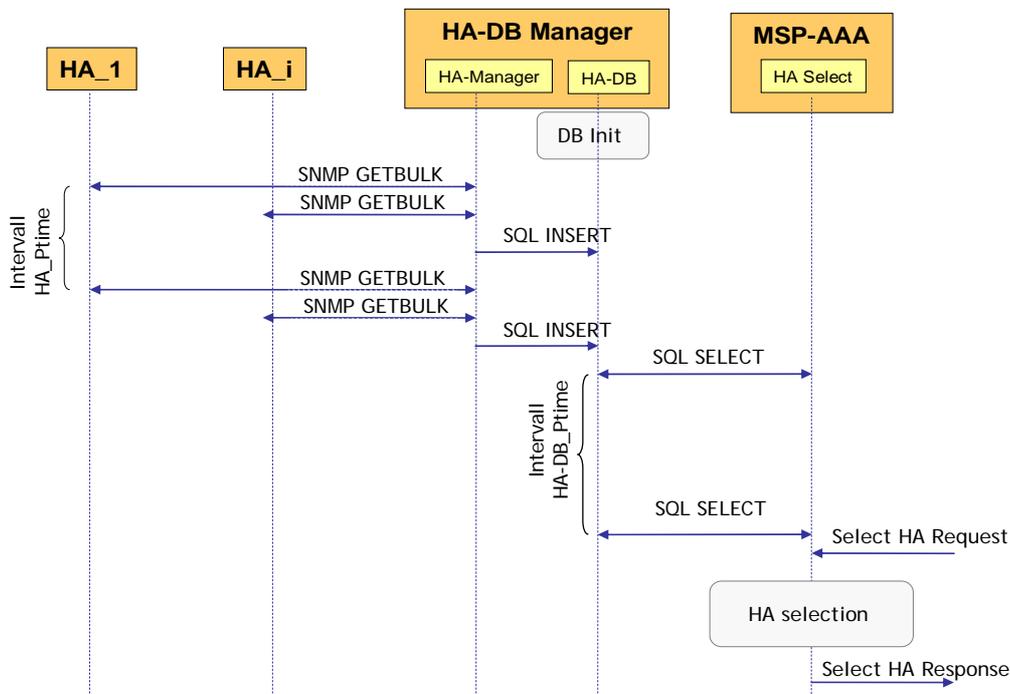
load parameters from the home agents and the HA-DB, a database in which all HA parameters relevant for load sharing are stored.

After start up of the HA-DB Manager, the HA-DB is initialized and the static parameters are set, which are for each HA its IP address (IP_HA), the maximum home registrations (Max_Reg), the maximum bandwidth on its interface (Max_Band), the ID of the Region (Region_ID) the home agent is located in, the maintenance flag (M_Flag), and the polling interval (HA_Ptime).

With an interval of HA_Ptime, the HA-Manager process periodically queries each HA (HA_1, .. HA_i) via SNMP for the current selection parameters (number of home registrations and currently consumed average bandwidth). The HA-Manager normalizes the parameters, and stores the parameters in the database HA-DB via SQL statements.

With an interval of HA-DB_Ptime, the HA Select process on the MSP-AAA entity queries the HA-DB via SQL for the current normalized parameters and stores them in its local Home-Agent-Parameter-Matrix. Having these selection parameters locally speeds up the selection process.

The HA Select process also provides an interface for other processes to invoke the selection mechanism. The requesting process sends a *Select HA Request* message towards the HA Select process, containing several parameters that specify and adjust the selection process. The HA Select process evaluates the load of each HA according to the values given in its local Home-Agent-Parameter-Matrix and returns a message *Select HA Response* that contains the IPv6 address of the selected HA.



One selection parameter that has to be retrieved from the home agents is the number of current home registrations. Therefore, the **interface He** between the NETSNMP agent and the MIPL process had to be realized. Since SNMP agents read from MIBs and neither the MIPv6 MIB has been implemented nor the required object is present in the current MIPv6 MIB specification, the required MIB object has been specified and implemented in ENABLE. The object denoted as **REGISTRATION** is linked within the enterprises.netSNMP.netSnmExamples subtree to the OID 1.3.6.1.4.1.8072.2.5.1.

A variable that stores the current number of home registrations is available in the MIPL file bcache.c, denoted as **bcache_count**. The file bcache.c was modified in order to store the variable bcache_count in the MIB object REGISTRATION.

Besides the number of home registrations the average consumed bandwidth on the HA interface is used as a selection parameter, which is calculated by using several available MIB objects defined in [RFC 1213]:

- IfInOctets (OID: 1.3.6.1.2.1.2.2.1.10): This MIB object represents the count of the inbound octets of traffic pertaining to the chosen Interface.
- IfOutOctets (OID: 1.3.6.1.2.1.2.2.1.16): The MIB object gives the total number of bytes sent on the chosen interface.

From these MIB objects the HA-Manager can calculate the average, currently consumed bandwidth using the following algorithm:

$$Bandwidth = \frac{\max[\Delta ifInOctets + \Delta ifOutOctets] * 8}{\Delta time * Max_Band}$$

with

$$\Delta ifInOctets = ifInOctets(t) - ifInOctets(t-1)$$

$$\Delta ifOutOctets = ifOutOctets(t) - ifOutOctets(t-1)$$

$$\Delta time = time(t) - time(t-1)$$

2.1.4.3 HA-DB module

The **HA-DB module** is implemented on the HA-DB Manager entity. The HA database (HA-DB) is realized in MySQL (Debian package “mysql-5.0” combined with the “mysql-connector-java-3.1.13” software). MySQL already supports the standard computer language: SQL, to provide data handling. For the interface Da to the HA-DB, JDBC is used (see next section). The JDBC function InitDB() is used to create the database table structure and to import HA parameters to

the database that are needed for the load sharing mechanism. The function is executable in a Java Runtime Environment and extracts line by line a configuration file (mipl-lh.conf). An example of the configuration file is displayed below:

```
# /etc/mipl-lh.conf

# The home agent list:

hip 2001:1b10:1001:2000:0000:0000:0000:0100 mreg 50 mband 10 region 1

hip 2001:1b10:1001:2000:0000:0000:0000:0101 mreg 10 mband 50 region 2

#ENDE
```

Figure 2-40: mipl-lh.conf

Each relevant row, indexed by the string “hip”, determines at least the IP address, and if required also the Max_Reg, the Max_Band and the Region_ID value, of a HA that has to be monitored. If a variable parameter is not set, a default value is set.

Figure 2-41 outlines the content stored in this HA-DB. The content contains for each HA:

- the selection parameters collected on the HA.
- the selection parameters already available on the HA-DB.
- as well as additional parameters required for HA load sharing.

The database is divided into four different tables. The table SELECTION PARAMETER STATIC contains the static parameters available on HA-DB which are configured by the administrator. The table SELECTION PARAMETER DYNAMIC contains all dynamic selection parameters collected periodically from the respective HAs (number of home registration and current bandwidth consumption). The table ADDITIONAL PARAMETER stores parameters that are not specifically used as selection parameters but are required for performing load sharing. These parameters are only relevant for the HA-DB Manager and therefore need not be transported to the HA Select process on MSP-AAA. The last table CACHE PARAMETER contains the last interface values needed to calculate the current average consumed bandwidth.

Each data set within these tables is indexed by the unique HA interface address (IP_HA).

Database tables

SELECTION PARAMETER STATIC		
IP_HA	Region_ID	M_Flag
⋮	⋮	⋮

SELECTION PARAMETER DYNAMIC		
IP_HA	Registrations	Bandwidth
⋮	⋮	⋮

ADDITIONAL PARAMETER		
IP_HA	Max_Reg	Max_Band
⋮	⋮	⋮

CACHE PARAMETER			
IP_HA	OutOctets (t-1)	InOctets (t-1)	Systeme (t-1)
⋮	⋮	⋮	⋮

Figure 2-41: Database tables on HA-DB

2.1.4.4 HA Manager Module and interfaces Pc and Da

The **HA Manager** module is realised in Java and runs on the HA-DB Manager entity. The Debian Gnu/Linux SARGE operating system and the Java Runtime Environment Version 1_5_0_06 is used on this entity.

The HA Manager uses the SNMP **interface Pc** to get selection parameters from the NETSNMP agents running on the home agents. The Pc interface uses a SNMPv2/v3 compatible GETBULK request that reduces the message overhead by obtaining a bundle of information with only one request message instead of many individual get request messages. If required, this management traffic can be secured using the security features of SNMPv3. For the SNMP support, we use NETSNMP from the sourceforge.net webpage (net-snmp-5.3.1) and we add the Westhawk's Java SNMP 5.1 stack to query SNMP information per Java code.

The obtained parameters are normalized to values between 0.00 and 1.00, e.g. in case the current number of home registrations is 1000 and Max_Reg is set to 4000 the normalize value for Registrations is 0.25. In case a home agent is not reachable, for example as a result of a reboot, its parameters will be set to 1.00 (the maximum) during normalization. These values are stored in the HA-DB using the **Da interface**. The interface is realised in Java and SQL commands are used to exchange data. The HA Manager process uses SQL INSERT and UPDATE commands, respectively, to insert and update the parameters of the database HA-DB. Since the HA Manager

is realised as a Java process, a HA-DB JDBC interface is used, which is a MySQL compatible API. (see <http://java.sun.com/javase/technologies/database/index.jsp>).

2.1.4.5 HA Select module and interfaces Pd and Ac

The **HA Select** module is a Java process that runs on MSP (here MSP = MSA = MASA). This process periodically queries the HA-DB for the current selection parameters via the **Pd interface**. The interface Pd between the HA-DB and the HA Select process on MSP-AAA is realized via **SQL commands**, especially the HA Select process uses SQL SELECT queries to retrieve the data from the database. Since the HA Select process is realised in Java, the HA-DB JDBC interface is used, which is a MySQL compatible API.

Upon request via the **Ac interface**, the HA Select module starts the selection process. The **interface Ac** is a local interface between the Freeradius instance and the HA Select instance on the MASA entity. The interface is realized as UDP socket, using port 5100, which has been chosen arbitrarily and should not conflict with other applications. The Freeradius instance sends a message *Select HA Request* to the HA Select instance. The HA Select instance selects a HA and sends back a message *Select HA Response* that contains the IPv6 address of the selected HA.

Messages defined for this Ac interface are:

- **Select HA Request** (Figure 2-42): message sent by the invoking process (e.g. the Freeradius instance) to the HA Select instance:

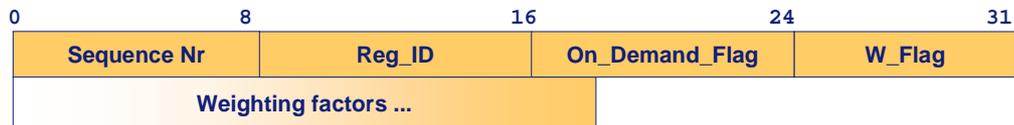


Figure 2-42: Select HA Request

- Sequence Nr (1 byte): The Sequence Nr is replied back in the Select HA Response message in order to relate request and response message.
- Reg_ID (1 byte): Region ID of the region within scope. A value of 0 signals that all regions are relevant.
- On_Demand_Flag (1 byte): If this value is set to 1, this signals to the HA Select process to update the parameter matrix. The default value is 0.
- W_Flag (1 byte): This value signals how many weighting parameters are following. A value of 0 indicates that no weighting factor follows and that the

load calculation should be done by using default weighting factors configured on the MSP-AAA.

- Weighting factors: This field contains a sequence of W_Flag weighting factors, each 2 bytes long.
- **Select HA Response** (Figure 2-43): message sent by the HA Select process to the invoking process (e.g. the Freeradius instance):

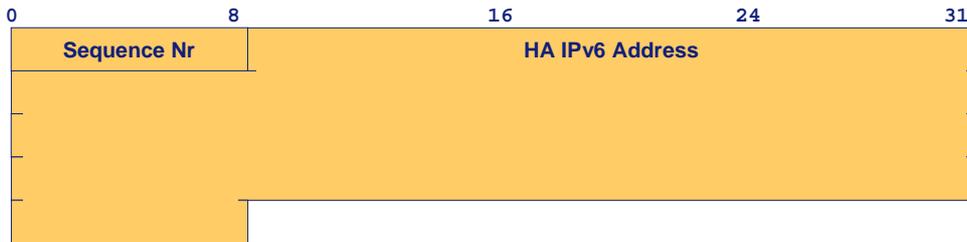


Figure 2-43: Select HA Response

- Sequence Nr (1 byte): Sequence number that is taken from the respective Select HA Request message in order to relate request and response message.
- HA IPv6 address (16 byte): IPv6 address of the selected HA.

For the selection of a HA, the HA Select module uses the parameter of its local Home-Agent-Parameter-Matrix. Thereby, the module does not take into account HAs that have the maintenance flag (M_Flag) set and, in the case were the Region_ID parameter in the *Select HA Request* message is not zero, considers only HAs with an appropriate Region_ID.

For each remaining HA, the respective current load is calculated via the following formula:

$$\text{load}_{\text{HA}_i} = W_1 * P_{i1} + W_2 * P_{i2} + \dots + W_n * P_{in}$$

P_{i1}, \dots, P_{in} are the HA selection parameters for HA_i

W_1, \dots, W_n are the weighting factors.

The least loaded HA is selected and its IP address (IP_HA) is returned in the *Select HA Response* message.

2.1.5 Interworking with IPv4 networks

Since basic MIPv6 [RFC3775] is only standardized to support IPv6, in the case were the access network is only IPv4 capable some extra work is required in order to allow the MN run MIPv6.

Deliverable [ENA-D2.2] has analyzed different alternatives for this scenario and two of them have been developed within the ENABLE project: DSMIPv6 and Softwires-based tunnelling.

2.1.5.1 DSMIP6

In these sub-sections we describe the DSMIP6 development carried out within the ENABLE project.

2.1.5.1.1 DSMIP6 Overview

Mobile IPv6 allows a Mobile Node (MN) to roam between IPv6 only networks. Interworking with IPv4 networks was implemented in order to allow within IPv4 networks:

- the exchange of MIPv6 signalling messages;
- IPv6 data transportation.

For a theoretical description of the solution we refer to the [ENA-D1.1], section 5.6.2. Here we describe the effective implemented solution for the Dual Stack MN and its movement detection and Dual Stack HA, extending what is stated in [ENA-D6.1], section 3.2.4.

2.1.5.1.2 Dual Stack MN and HA

A dual stacked MN has the ability to send and receive MIPv6 signalling messages and IPv6 data packets while it is attached within an IPv4 network.

A requirement for this approach is the presence of a HA and a Home network which is dual stacked. This assumption is needed since this solution is based on the tunnelling of IPv6 packets, for both signalling and/or data, within an IPv6-in-IPv4 tunnel, the end points of the tunnel being the MN and the HA.

The format of the Binding Update (BU) and Binding Acknowledge (BA) messages will depend on the type of access network the MN is attached to. We will refer to the case of a MN attached to an IPv4 network and configured with an IPv4 public address, as assumed in [ENA-D6.1], section 3.2.4.1. In our deployment the message format is shown in Figure 2-44:

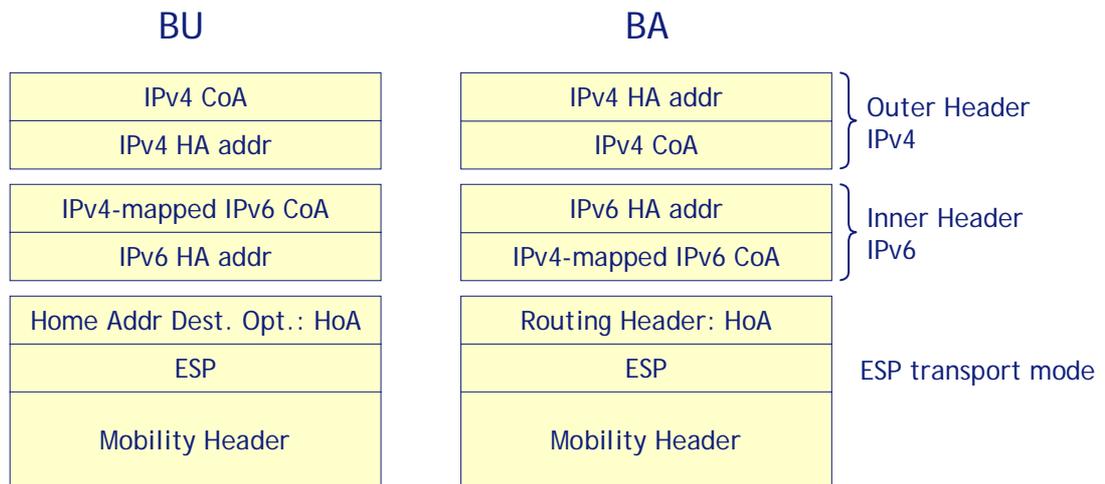


Figure 2-44: BU/BA messages within IPv4 only public network

The destination address of the outer header is the IPv4 address of the HA and the source address is the IPv4 public address that the MN has. In the inner packet there is a standard IPv6 BU/BA message in which an IPv6 CoA is announced. Since the MN owns only an IPv4 CoA, the IPv6 CoA field is filled with the IPv4 address represented in the IPv4-mapped IPv6 form.

The processing of the BU containing the registration of the IPv4-mapped IPv6 CoA is the same as specified within [RFC3775]. When the registration is accepted, the HA creates a new Binding Cache Entry (BCE) where the CoA address (IPv4-mapped IPv6) is inserted in association with the IPv6 HoA. Following the standard [RFC3775], once the BCE is inserted, the HA replies to the MN sending a BA. The destination address of the BA is the IPv4-mapped IPv6 address. The HA, recognising these types of addresses, sends the BA through the IPv6-in-IPv4 tunnel using as destination address the IPv4 address of the MN and as source address his IPv4 address. Upon receiving this BA, the MN updates the Binding Update List (BUL) entry referring to the CoA IPv4-mapped IPv6 contained within the BA.

Hence, all packets addressed to the mobile node's IPv6 home address will be encapsulated in an IPv4 packet that includes the home agent's IPv4 address in the header's source address field and the mobile node's IPv4 care-of address in the destination address field.

2.1.5.1.3 Movement detection

The Movement Detection (MD) algorithm was implemented based on Neighbour Discovery procedures, within IPv6 access networks, and on Dynamic Host Configuration Protocol (DHCP), within IPv4-only access networks.

Since in the considered scenario the MN is able to move among both IPv6 and IPv4 subnets, the movement detection procedure has to be able, in both cases, to understand when the MN changes IP subnet in order to trigger the binding management procedure.

In order to define an algorithm we made some assumptions:

- when an interface of the MN is attached to an IPv6 network, then stateless auto-configuration is used [RFC2462] to obtain a global IPv6 address and the Neighbour Discovery procedure [RFC2461] to probe the reachability of the default router;
- when an interface of the MN is attached to an IPv4 network, then the DHCP protocol [RFC2131] is used to configure a public IPv4 address and the ARP protocol to probe the reachability of the default router.

This algorithm has been designed to support MNs that own multiple interfaces. A preference value is assigned to each interface to select which interface must be used when two or more interfaces have connectivity. If an interface has both IPv4 and IPv6 connectivity the IPv6 protocol is the preferred one.

2.1.5.1.4 Kernel development

The MN and HA required a fix to the kernel for the proper handling of IPv4-mapped IPv6 CoAs in MIPv6 messages. The changes were made in the file *net/ipv6/xfrm6_policy.c* and collected into a patch displayed in Figure 2-45.

```
# net/ipv6/xfrm6_policy.c
# 2007/07/30 09:22:31+01:00
# changes for the IPv4 Interworking
#
--- a/net/ipv6/xfrm6_policy.c 2007-07-30 11:14:51.000000000 +0200
+++ b/net/ipv6/xfrm6_policy.c 2007-07-30 11:15:31.000000000 +0200
@@ -173,8 +173,10 @@
         tunnel = 1;
         break;
     case IPPROTO_DSTOPTS:
+    if (!(ipv6_addr_type((struct in6_addr *)xfrm[i]->coaddr)&IPV6_ADDR_MAPPED)) {
+        local = (struct in6_addr*)xfrm[i]->coaddr;
+        tunnel = 1;
+    }
     break;
     default:
     break;
```

Figure 2-45: Kernel patch for IPv4 Interworking

This patch has to be applied to a MIPL ready kernel 2.6.16, i.e. a vanilla 2.6.16 kernel previously patched with standard MIPL 2.0.2 patch.

2.1.5.1.5 MIPL userspace development

The new functions and routines needed to implement the movement detection algorithm were derived from the MIPL original ones, while the original data structures were modified with suitable fields for IPv4 data.

In particular, management of tunnels between MN and HA was extended with the use of IPv6-in-IPv4 tunnels, which prototyped under Linux has the named *sit0*. This type of tunnel is required for MIPL signalling between the MN and HA and IPv6 traffic exchange between MN and CNs forwarded by HA.

All modifications and enhancements can be turned on or switched off via the MIPL configuration file.

2.1.5.1.6 MN development

A DHCP client was embedded into the MIPL userspace to manage IPv4 connectivity and perform the movement detection algorithm. During the configuration of internal MIPL data structures for mobility interfaces (i.e. interfaces used by MIPL and included into the MIPL configuration file), a DHCP thread is created per mobility interface to verify IPv4 connectivity via DHCPDISCOVER broadcast messages. Once an IPv4 address is obtained and configured on the interface, meaning that the MN is in an IPv4 or DS network, the thread starts and manages the timer for ARP requests to probe IPv4 router reachability.

In the MN configuration file two new tokens were added, shown in bold characters in Figure 2-46 below:

```
...
Bootstrap enabled;
HomeAgentName "ha1.ist-enable.tilab.com";

InterworkingIPv4 enabled;

Ikev2Psk "";

MnHomeLink "ath0" {
    HomeAgentAddress ::;
    HomeAddress 2001:6b8:20:186::1:0:ea1/64;
    HomeAgentAddressIPv4 10.0.0.0;
}
...
```

Figure 2-46: Example of MN's configuration file with Interworking

InterworkingIPv4 enables interworking capabilities, while *HomeAgentAddressIPv4* is used to initialise the IPv4 HA address field during parsing of the configuration file at MIPL start up.

2.1.5.1.7 HA development

The HA sets up an IPv6-in-IPv4 (*sit*) tunnel at bootstrap which is between its own local IPv4 address and any remote IPv4 address, to receive BUs sent from Interworking enabled MNs.

The Binding Cache management was modified to deal with such BUs. If an Interworking enabled MN sends a BU from an IPv4 network, HA sets up a new sit tunnel between its own local IPv4 address and the MN's remote IPv4 address, inferred from the IPv4-mapped IPv6 CoA. The new dedicated tunnel is then used either for MIPv6 signalling or for traffic forwarding.

When a MN performs a handover from an IPv4 network to an IPv6 network, it sends a BU with the new IPv6 CoA: then the HA modifies the Binding Cache Entry according to new CoA, deletes the sit tunnel and sets up a new IPv6-in-IPv6 tunnel between itself and the MN as in the MIPL standard. Creation and deletion of tunnels is due not to have more than one tunnel active per MN. The behaviour is the other way round when MN roams from an IPv6 network to an IPv4 one.

To enable Interworking functionalities, the HA configuration file needs two new tokens, highlighted in bold in Figure 2-47 below. The former is the same as in the MN to enable Interworking, the latter is the IPv4 address of the HA's interface used for MIPv6.

```
...  
HaveAuthorization disabled;  
HoAsFile "/etc/enable/hoas.txt";  
  
## Interworking IPv4  
InterworkingIPv4 enabled;  
HAddressIPv4 163.162.186.101;  
  
##  
## IPsec configuration  
  
...
```

Figure 2-47: Example of HA's configuration file with Interworking

2.1.5.2 Softwires

In the remaining sub-sections we describe the development of the Softwires-based tunnelling solution for MIPv6 IPv4 interworking.

2.1.5.2.1 Softwires overview

As explained in [ENA-D2.2], the softwires approach is a solution that proposes to get IPv6 connectivity firstly by building an IPv6 tunnel and then secondly by running mobility over that tunnel. The protocol for making the IPv6 tunnel is called softwires [draft-ietf-softwire-hs-framework-12tpv2] which standardizes the discovery, control and encapsulation methods for connecting IPv4 networks across IPv6 networks and IPv6 networks across IPv4 networks in a way that will support all the possible network scenarios.

Softwires is currently being standardized on the basis of the use of the L2TPv2 protocol [RFC2661] (L2TPv3 [RFC3931] will also be supported in the future) so the development done within the ENABLE project is based on L2TPv2. Specifically, the softwires implementation has been based on the *l2tpd_0.70-pre20031121*¹ and *ppp-2.4.3*² implementations and modified to be softwires compliant.

L2TPv2 is designed to transport L2 PPP packets through different networks: i.e. IP, FR and ATM. However Softwires only applies to IP networks, which is the main scenario which has been designed for. In the MIPv6 IPv4 interworking context the IPv6 address is provided through the PPPv6 session [RFC2472] and the final tunnel encapsulation is as depicted in figure 2-48 for both the control and data channels.

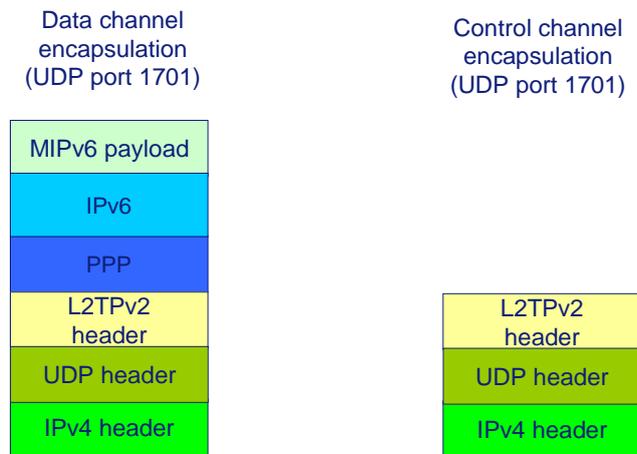


Figure 2-48: Softwires encapsulation in the MIPv6 IPv4 interworking context

As can be seen, softwires provides two channels, one for control (session establishment, maintenance and tear down) and the other one for data transportation. Once the softwires tunnel has been configured, all the MIPv6 traffic is sent through the data channel. Both the control and data channels have different L2TPv2 headers, as defined in [RFC2661].

When Softwires is used, there are two new components and interfaces in the reference architecture, as shown in figure 2-49.

¹ Available from http://ftp.debian.org/debian/pool/main/l/l2tpd/l2tpd_0.70-pre20031121.orig.tar.gz

² Available from http://sourceforge.net/project/showfiles.php?group_id=44827. The PPP implementation must be installed before installing Softwires. The PPP implementation must support IPv6.

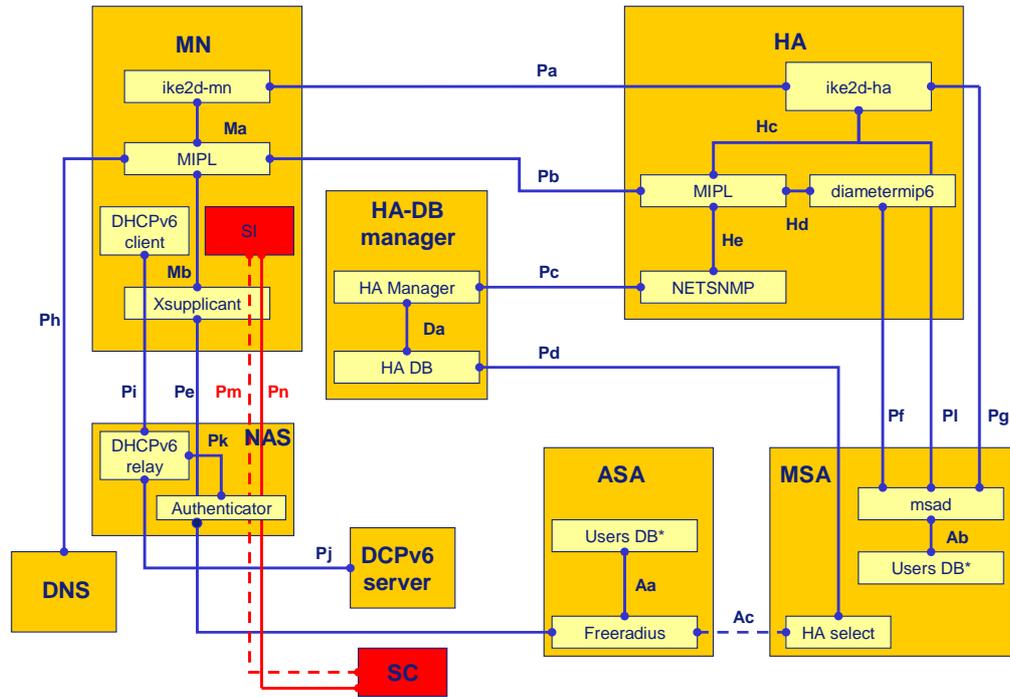


Figure 2-49: Softwires components and interfaces in the reference architecture

The Softwires modules and interfaces in the reference architecture are marked in red. The Softwires Initiator (SI) is the component responsible for starting the tunnel creation. When applying softwires in the MIPv6 IPv4 interworking context, the MN plays the SI role, i.e. IPv6 tunnel initiator. The Softwires Concentrator (SC) is the Tunnel End Point (TEP). It is in charge of ending the IPv6 tunnel, i.e. extracts the IPv6 packets from the received IPv4 packets and forwards them to the IPv6 destination. It makes the reciprocal tasks for IPv6 packets sent to the SI (MN), i.e. it gets the native IPv6 packet, encapsulates it in an IPv4 packet and forwards it to the SI. In the MIPv6 IPv4 interworking context the SC must be deployed in a dual-stack network (MSP or Third-Party provider) as stated in [ENA-D2.2].

Using Softwires as MIPv6 IPv4 interworking solution involves five phases as depicted in the following message flow:

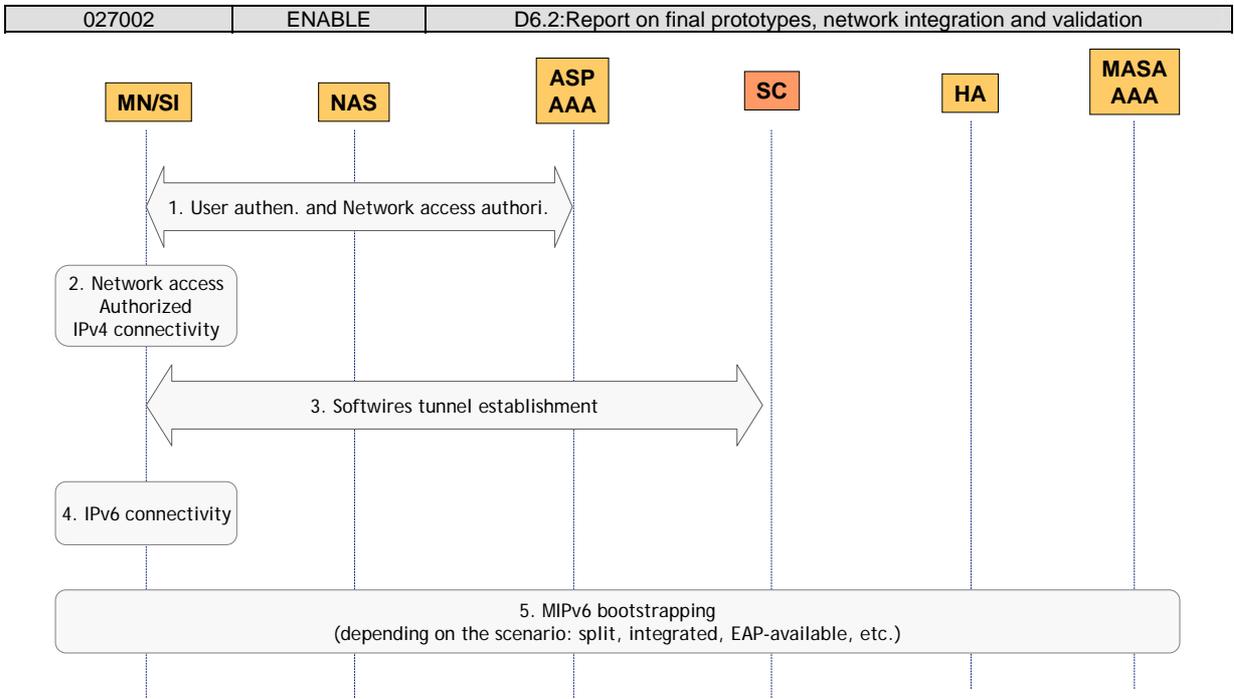


Figure 2-50: Phase required when using Softwires

First of all the MN needs to be authorized to use the access network by providing the proper user credentials to the ASP-AAA component (steps 1). When the MN is able to use the network access (steps 2) it only has IPv4 connectivity (IPv4-only network access), so it must setup the Softwires tunnel in order to get IPv6 connectivity (step 3). Because the softwires-solution development is only a prototype within the ENABLE project, the SC address is assumed to be manually configured in the MN/SI. Provisioning the SC address dynamically leads to a “Softwires bootstrapping” issue which is out of the ENABLE scope, so in this development it is assumed that the MN/SI is configured properly to contact the SC in order to address the step 3.

After the Softwires tunnel has been configured (see the following sections), the MN has IPv6 connectivity (step 4) and it is ready to bootstrap MIPv6 over the new IPv6 network interface according to the scenario where it is found (split, integrated, etc.).

2.1.5.2.2 Modules SC and SI and interfaces

Both the SC and the SI is the same piece of software (the binary is called *softwiresd*), the only difference is how they are configured to work as, so they will be described together in this section.

The *softwiresd* binary is responsible for the L2TPv2 control channel establishment, which is done through the interface *Pm* in the figure 2-49. It is also responsible for the L2TPv2 control session, maintenance and tear down session. Then after the control channel is established, in order to setup the data channel (*Pn* in the figure 2-49), the *softwiresd* daemon calls the *pppd*

daemon which will be in charge of providing the IPv6 connectivity. Finally, once the PPP session has been successfully established between the SI and the SC, the SI gets a global IPv6 address by using IPv6 auto-configuration.

Softwires tunnel configuration and establishment

The following are the configuration files used in Sofwires:

- **l2tpd.conf.** Defines how the Softwires will work: i.e. SC or SI. It is found in both the SC and SI.
- **l2tpd.secrets.** Stores information for user authentication. It is found in both the SC and SI.
- **ppp.options.** Defines specific PPP options for setting up the PPP session. The IPv6 configuration for the softwires tunnel is configured here. It is found in both the SC and SI.
- **radvd.conf.** Defines the IPv6 network information required for IPv6 auto-configuration to work through the PPP interface. It is found only in the SC.

The *softwiresd* binary has the following command syntax:

```
softwiresd [ -c configuration file ] [ -s secret file ]
           [ -p pid file ] [ -D ]
```

The command line options are:

```
-c      configuration file
        Selects a different configuration file from the default
        (typically /etc/softwiresd/l2tpd.conf).
-s      secret file
        Selects a different secret file from the default (typically
        /etc/softwiresd/l2tp-secrets).
-p      pid file
        Selects a different pid file from the default (typically
        /var/run/softwiresd.pid).
-D      Foreground mode. Messages will be sent to STDERR, and the process
        will not detach from the shell.
```

Figure 2-51: Command syntax for Softwires

The role (SC or SI) of the Softwires component is defined in the configuration file (*/etc/softwiresd/l2tpd.conf* by default). Such a file contains configuration information for the L2TP protocol. Below are provided examples of the configuration file to let the *softwiresd* daemon work as a SC (LNS in l2tp terminology) and SI (LAC in l2tp terminology).

```

[global]
listen-addr = 192.168.60.3           ; Address to bind to
port = 1701                          ; Port to bind to

[lns default]
ip range = 192.168.140.100-192.168.140.199 ; Specify the range of
                                           ; ip addresses the SC will
                                           ; assign to the connecting
                                           ; SI PPP tunnels
local ip = 192.168.140.3             ; softwired own ip address
require authentication = yes         ; ppp requires authentication
require chap = yes                  ; ppp requires CHAP auth.
ppp debug = yes
pppoptfile = /etc/ppp/ppp.options   ; ppp configuration file

```

Figure 2-52: Configuration example for Softwires SC component

The previous example shows that the *softwired* component will work as a SC. It will assign the 192.168.140.3 address to the PPP interface on the SC side and an IP address within the range 192.168.140.100-199 to the PPP interface on the SI side. It will use chap as the authentication method before setting up the PPP session. Other specific PPP options will be taken from the */etc/ppp/ppp.options* file.

```

[global]                               ; Global parameters:
port = 1701                             ; Port to bind to

[lac si1]                               ; SI configuration section
lns = 192.168.1.6                       ; SC to connect to
pppoptfile = /etc/ppp/options.l2tpd.lac ; ppp configuration file
require authentication = yes            ; ppp requires authentication
require chap = yes                      ; ppp requires CHAP auth.

[lac si2]                               ; SI configuration section
lns = 192.168.60.3                     ; SC to connect to
pppoptfile = /etc/ppp/ppp.options       ; ppp configuration file
ppp debug = yes
require authentication = no             ; authentication not required

```

Figure 2-53: Configuration example for Softwires SI component

The previous example shows that the *softwired* component has two SI configurations (one only SI configuration is possible). The first one (SI1) will connect to the SC whose IP address is

192.168.1.6. The second one will connect to the SC whose IP address is 192.168.60.3. The *softwired* daemon will run as either SI1 or SI2, depending on how the tunnel command is typed in the shell:

```
echo "c si1" > /var/run/softwires-control
    SI will connect to the SC defined in the si1 section of the
    l2tpd.conf configuration file

echo "c si2" > /var/run/softwires-control
    SI will connect to the SC defined in the si2 section of the
    l2tpd.conf configuration file
```

Figure 2-54: Syntax to establish the softwires tunnel in the SI

In order to provide IPv6 connectivity to the PPP session, both the SC and the SI must enable IPv6 in their respective *ppp.option* configuration file. The specific parameters are shown below:

```
+ipv6                #Enable IPv6 in the PPP session
ipv6 ::201,::220     #Provide IPv6 link local address in both peers
```

Figure 2-55: IPv6 parameters for the PPP session

A softwires tunnel is established through the *Pm* interface in 2 differentiated phases:

1. To establish the softwires tunnel, the SI first initiates an L2TPv2 Control Channel by using the syntax shown in figure 2-54. All the messages are sent through UDP datagrams (port 1701 by default) to the SC which accepts the request and terminates the Control Channel setup. After the L2TPv2 Control Channel is established between the SI and SC, the SI initiates an L2TPv2 Session to the SC which takes 3 RTT as shown in figure 2-56. The meaning of each message is defined in [RFC2661].

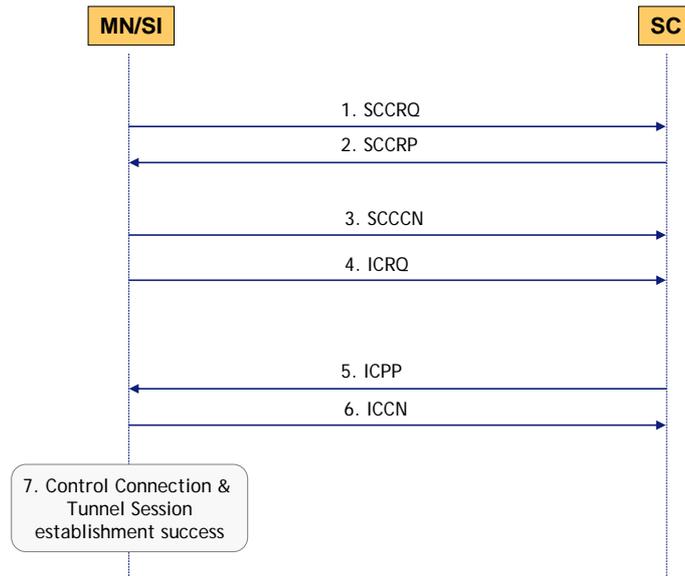


Figure 2-56: Softwires establishment step 1

2. During the second phase, a PPP link is negotiated over the L2TPv2 session between the SI and SC. In this phase user authentication might be required if either or both the SI and SC are configured to do so. In such a case, the SC might keep in touch with an AAA infrastructure (for instance with the MSA if the SC is deployed in the MSP network) which is not depicted in figure 2-57. Then after the user authentication success and the PPP link is set up, the SI is provided with an IPv6 link local address (step 21). In order to provide a global IPv6 address, the SC does the following tasks (step 22) which lets the SI get a global IPv6 address by using IPv6 auto-configuration as the last step (step 23 and 24):

- It configures the system to forward packets.
- It finds out the name of the PPP interface just created.
- It configures the *radvd.conf* file for the PPP interface just created in order to advertise the proper IPv6 network prefix.
- It runs the *radvd* daemon.

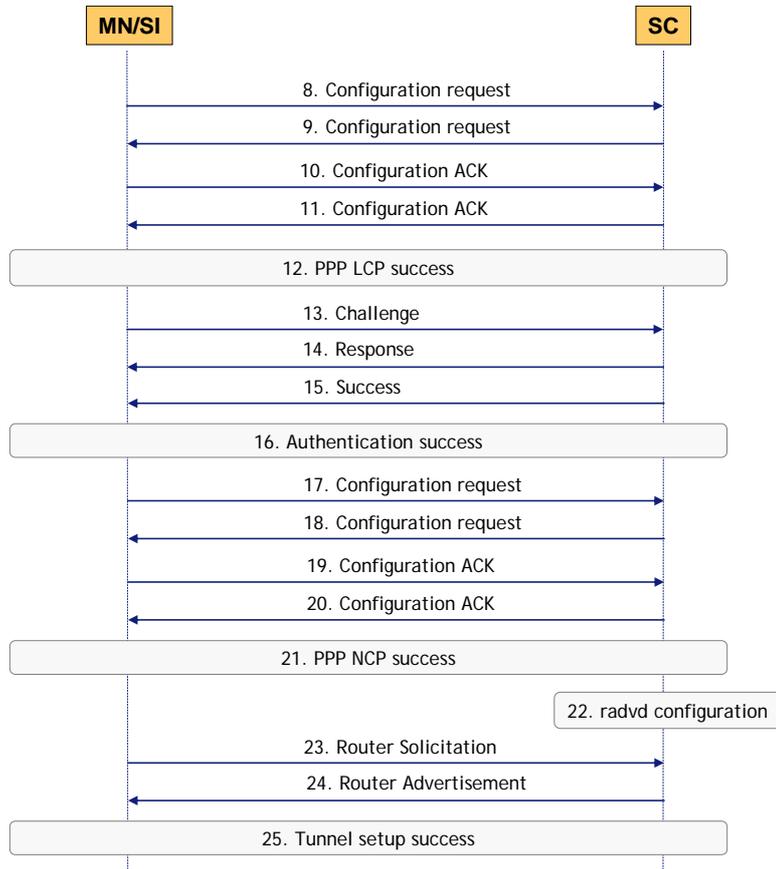


Figure 2-57: Softwires establishment step 2

After the PPP/IP link is established, it acts as the softwires tunnel between the SI and SC for tunnelling IPv6 traffic across the IPv4 access network. The MN can bootstrap in order to start the mobility service. This is done through the interface P_n by encapsulating all the MIPv6 traffic (both signalling and data) in the PPP/L2TP/UDP packets (port 1701), as depicted in figure 2-48. All the IPv6 traffic sent through the P_n interface is forwarded by the SC to the IPv6 internet. Both the control and data channels (interfaces P_m and P_n) share the same UDP port, but they have different L2TP header formats as defined in [RFC2661].

During the life of the tunnel, both the SI and SC may send L2TPv2 *keepalive HELLO* messages through the P_m interface in order to monitor the health of the softwires tunnel and to refresh the NAT/PAT translation entry at the middle-boxes in path (if any). In the event of keepalive timeout or administrative shutdown of the softwires tunnel, either the SI or the SC may tear down the L2TPv2 Control Channel and Session.

Softwires tunnel tear down

The softwires tunnel is deleted by the SI by using the following syntax:

```
echo "d sil" > /var/run/softwires-control
SI will connect to the SC defined in the sil section of the
l2tpd.conf configuration file
```

Figure 2-58: Syntax to delete the softwires tunnel in the SI

2.2 Additional Software Developments

2.2.1 Mobile IPv6 Firewall Traversal

The Mobile IPv6 firewall traversal as developed within ENABLE is described in [ENA-D2.1.2]. This section will focus on the Mobile IPv6 firewall traversal prototype implementation, written by the University of Goettingen within ENABLE.

2.2.1.1 Mobile IPv6 Firewall Traversal Reference Architecture

Figure 2-59 shows the Mobile IPv6 firewall traversal software architecture. It should be noted that this reference architecture is based on the assumption that the MSA and the MSP are co-located.

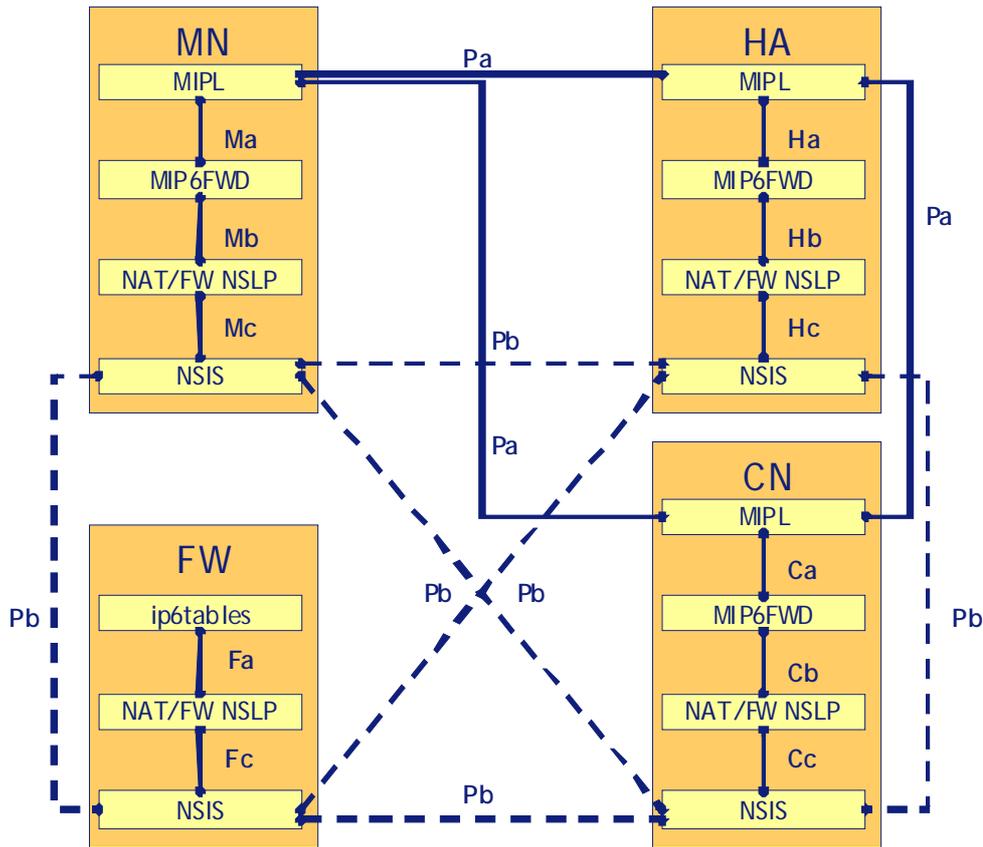


Figure 2-59: Mobile IPv6 Firewall Traversal Architecture

The functional elements (orange rectangle) that compose this architecture are:

- Mobile Mode (MN),
- Home Agent (HA),
- Corresponding Node (CN),
- Firewall (FW).

The yellow rectangles represent the software modules, namely MIPL, MIP6FWD, NAT/FW NSLP, NSIS and ip6tables. Meanwhile the main interfaces are represented with blue connectors. Both software modules and interfaces are described in the following subsections.

2.2.1.2 Software modules

2.2.1.2.1 MIPL (MN)

The MIPL module implements the Mobile Node functionalities. The release used is MIPL 2.0.2 [MIPL] developed by GO-Core in co-operation with the USAGI/WIDE Project [USAGI].

The Mobile IPv6 firewall traversal has required some extensions to the MIPL code:

- When the MIPL implementation on the MN gets a new CoA and want to send a Binding Update to the HA, the MIPL implementation is halted and trigger the MIP6FWD to trigger a reconfiguration via the interface **Mfa**. The MIP6FWD now triggers the NAT/FW NSLP via interface **Mfb**. When the reconfiguration is finished successfully and the MIPL receives a corresponding response, it triggers the MIP6FWD to install firewall pinholes for the Binding Update via interface **Mfa**, which again uses **Mfb** to trigger the NAT/FW NSLP. If the MIPL knows about the successful pinhole creation to the HA, it is resumed and sends out the Binding Update to the HA as it now can traverse the firewalls.
- If the MIPL implementation wants to send data traffic to a CN, either directly to the CN or via the HA, it is halted and triggers the MIP6FWD to install the required firewall pinholes via the interface **Mfa**, which forwards this trigger to the NAT/FW NSLP using the interface **Mfb**.
- The MIPL implementation on the MN communicates with the MIPL implementation on the HA and the CN using the interface **Pfa**, which implements the Mobile IPv6 protocol [RFC3775].

2.2.1.2.1.1 *Mfa* (MIPL – MIP6FWD)

Mfa is an inter-process interface implemented using a TCP socket. This interface is used to trigger the MIP6FW daemon in order to trigger the pinholes between the nodes when they are required. Therefore, a trigger message (of type CREATE) is used, which signals the MIP6FWD the format of the required pinhole. The format for each type of firewall pinhole is described in detail in [ENA-D2.1.2]. After triggering the MIP6FWD, the MIPL implementation is halted and waits for a response message from the MIP6FWD. With the response the MIP6FWD daemon informs the MIPL implementation about the success of the pinhole creation.

The create trigger messages exchanged between MIPL and MIP6FWD has the following format (Figure 2-60).

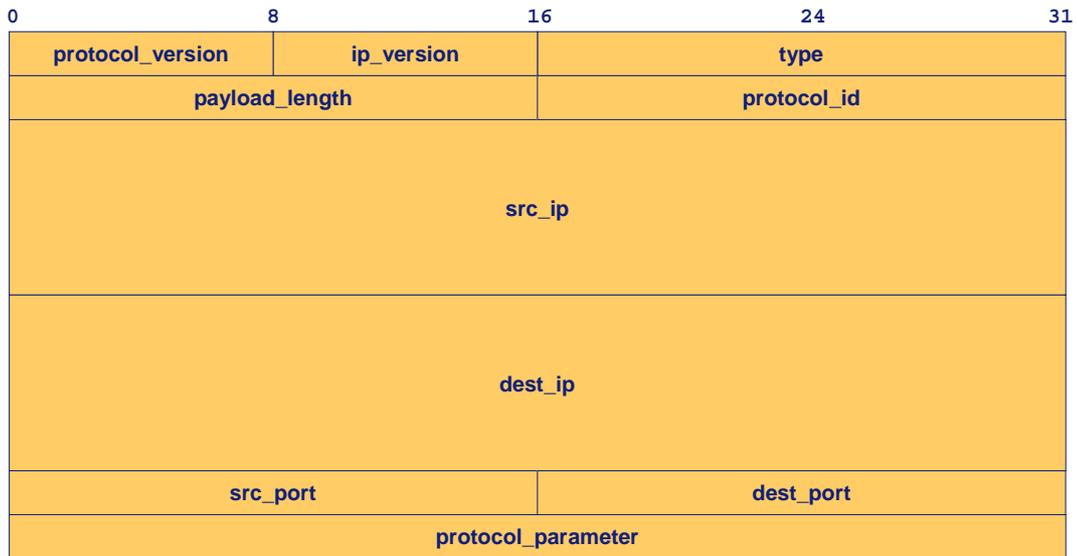


Figure 2-60: MIPL-MIP6FWD create message

- **protocol_version:** specifies the version of the interface between MIPL and MIP6FWD. Current version is “1”.
- **ip_version:** defines whether the IP version of the request is IPv4 or IPv6. In Mobile IPv6 firewall traversal case it is always set to IPv6.
- **type:** defines the type of the packet. Possible types are:
 - MIPTRIGGER_MSG_ERROR (0x00),
to inform the MIPL implementation about an occurred error.
 - MIPTRIGGER_MSG_ACK (0x01),
to inform the MIPL about the success of a request.
 - MIPTRIGGER_MSG_CREATE (0x02),
to trigger MIP6FWD for pinhole creation.
 - MIPTRIGGER_MSG_STATUS (0x03),
to exchange the status of the MIPL and the MIP6FWD implementations.
 - MIPTRIGGER_MSG_NSIS_AUTOCONF (0x04),
when the MIP6FWD receives this message, it triggers the NSIS NAT/FW NSLP implementation to re-configure its IP addresses. This is required after a handover; otherwise NSIS would not know of its new addresses.

- **payload_length**: Defines the length of the payload.
- The following fields specify the format of the requested firewall pinhole:
 - **protocol_id**
 - **src_ip**
 - **dest_ip**
 - **src_port**
 - **dest_port**
 - **protocol_parameter**: for special types for this protocol, e.g. the type of a mobility header.

Unlike the create message, the other message types, namely the error, response, status and autoconf messages have the same message format. These messages are also used for exchange between MIPL and MIP6FWD and have the format as shown in Figure 2-61.

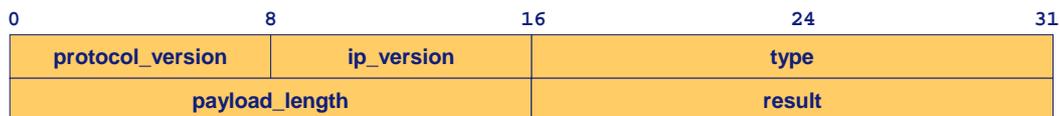


Figure 2-61: MIPL-MIP6FWD response message

The format of the **protocol_version**, **ip_version**, **type** and **payload_length** fields have the same specification as for the create message.

- **result**: Informs the MIPL specification about the result of a pinhole creation request. Possible values are:
 - MIPTRIGGER_RESULT_UNKNOWN (0x00),
 - MIPTRIGGER_RESULT_ACK (0x01),
 - MIPTRIGGER_RESULT_NACK (0x02),
 - MIPTRIGGER_RESULT_NACK_FW (0x03),
 - MIPTRIGGER_RESULT_PROTOCOL_ERROR (0x05),

- MIPTRIGGER_RESULT_QUEUE_BUSY (0x10),
- MIPTRIGGER_RESULT_QUEUE_EMPTY (0x11).

The rules that the MIPL and the MIP6FWD modules have to observe are:

- As soon as MIPL implementation knows the new care-of-address (CoA), it sends a MIPTRIGGER_MSG_NSIS_AUTOCONF to the MIP6FWD daemon.
- When the MIP6FWD receives a MIPTRIGGER_MSG_NSIS_AUTOCONF messages, it triggers the NSIS NAT/FW NSLP to re-configure its IP-addresses and informs the MIPL implementation about the success (compare section 2.2.1.2.1.2).
- After receiving a successful MIPTRIGGER_MSG_NSIS_AUTOCONF response, the MIPL implementation begins to start the pinhole creation progress for the first pinhole (the pinhole for the BU/BA messages between the MN and the HA). Therefore, it triggers the MIP6FWD daemon with a MIPTRIGGER_MSG_CREATE message to install this pinhole and waits for the response.
- When the MIP6FWD receives an MIPTRIGGER_MSG_CREATE message, it computes this request and triggers the NAT/FW NSLP to install the firewall pinhole as it is requested. It later informs the MIPL implementation about the success of the pinhole creation request.
- If the MIPL implementation receives a successful MIPTRIGGER_MSG_ACK to a MIPTRIGGER_MSG_CREATE message, it resumes the normal MIPL implementation. This could be to trigger the creation of further firewall pinholes, to trigger NSIS for a new re-configuration or to resume the normal MIPL process.

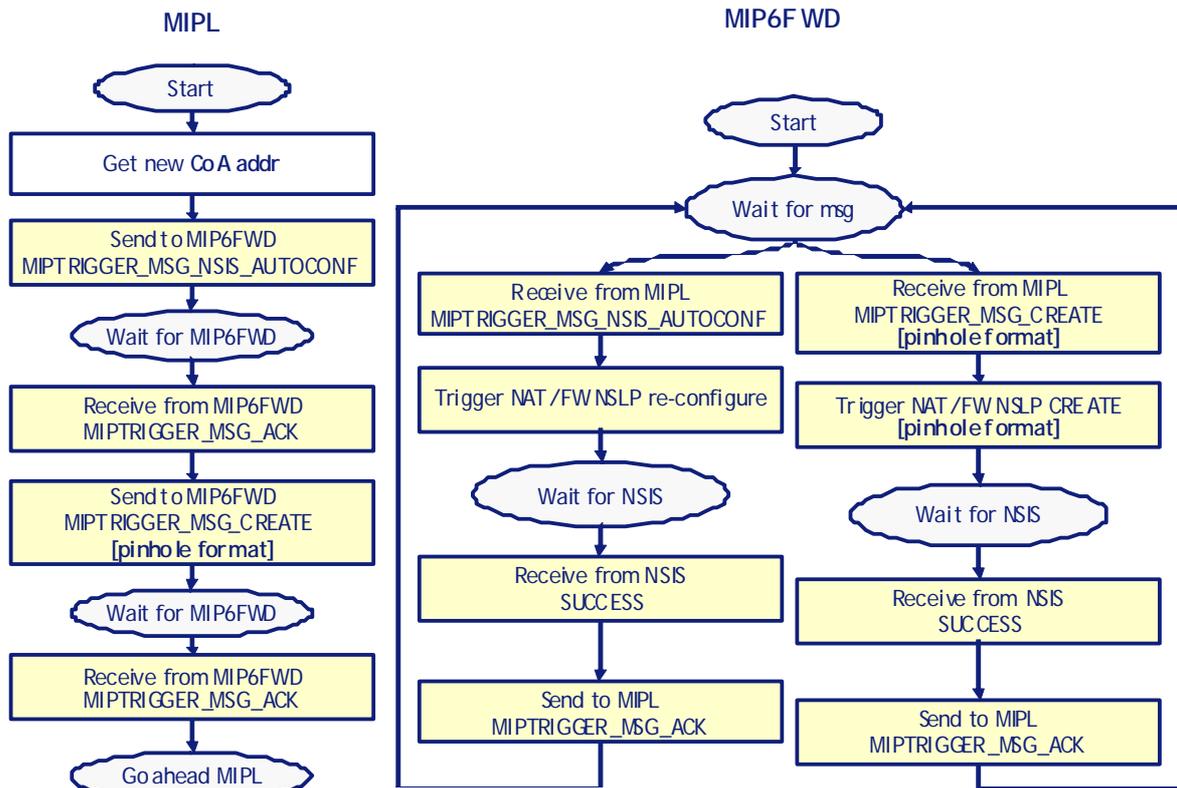


Figure 2-62: Interaction between MIPL and MIP6FWD (MN)

2.2.1.2.1.2 Mfb (MIP6FWD – NAT/FW NSLP)

Mfb is an inter-process interface implemented using a TCP socket. This interface is used to trigger the NAT/FW NSLP to trigger the pinhole creation progress upon the event of receiving a message at the MIP6FWD. Therefore, a trigger message is used, which signals to the NAT/FW NSLP implementation the type of the requested pinhole. After triggering the NAT/FW NSLP, the MIP6FWD daemon is waiting for a response from the NAT/FW NSLP implementation. With the response the NAT/FW NSLP implementation informs the MIP6FWD daemon about the success of the pinhole creation.

The trigger message exchanged between the MIP6FWD and the NAT/FW NSLP has the following format (Figure 2-63). Note: this message format is used for all kind of signalling between an application and the NAT/FW NSLP. Therefore, it also consist fields which are not necessary for the Mobile IPv6 firewall traversal case.

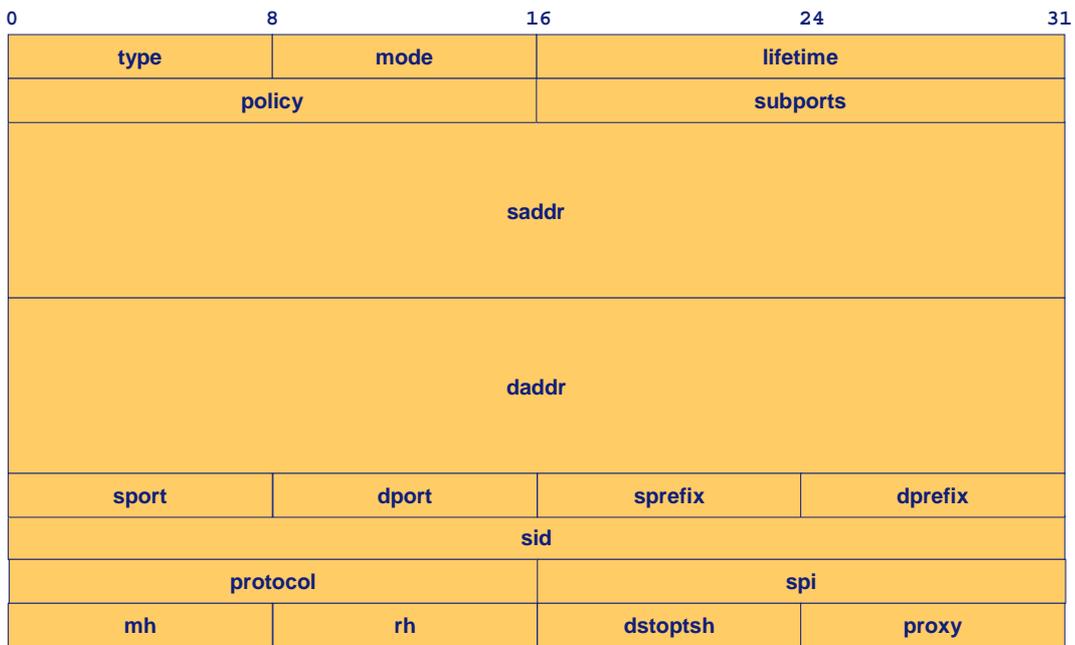


Figure 2-63: MIP6FWD-NATFW NSLP trigger message

- **type:** defines the type of the packet. Possible types are:
 - NATFW_TYPE_REA (0x01),
to signal for a EXT message (not required for MIPv6 firewall traversal).
 - NATFW_TYPE_CREATE (0x02),
to signal for a CREATE message.
 - NATFW_TYPE_TRACE (0x03),
to signal for a TRACE message (not required for MIPv6 firewall traversal).
 - NATFW_REREADROUTINGTABLE (0x04),
when the NAT/FW NSLP receives this kind of messages, it triggers the NSIS implementation to re-configure its IP addresses. This is required after a handover; otherwise NSIS would not know about its new addresses.
- **mode:** specifies the mode of the message. Possible values are NATFW_MODE_CMODE (0x01) and NATFW_MODE_DMODE (0x02).
- The following fields specify the format of the requested firewall pinhole:
 - **lifetime:** specifies the lifetime of a firewall pinhole in seconds. If the pinhole is not refreshed before the lifetime has expired, it will be automatically removed.

- **policy:** defines the policy of the pinhole. Possible values are ALLOW (0x01) and DENY (0x02).
- **subports:** defines a range of port (if the pinhole requires them).
- **saddr, sport, sprefix:** source address, source port and source prefix.
- **daddr, dport, dprefix:** destination address, destination port and destination prefix.
- **protocol:** defines the protocol the pinhole matches to.
- **spi:** specifies a SPI (if the pinhole requires).
- **mh, rh, dstoptsh:** to signal for a special type of Mobility Header, Routing Header or Destination Options Header.
- **sid:** The session ID of this request. Used to identify the request, e.g. in the response message.
- **proxy:** flag to trigger the NAT/FW NSLP to use the proxy mode (not required for MIP6 firewall traversal).

The response message format, sent from the NAT/FW NSLP to the MIP6FWD to inform it about the result of a request is shown in Figure 2-64.

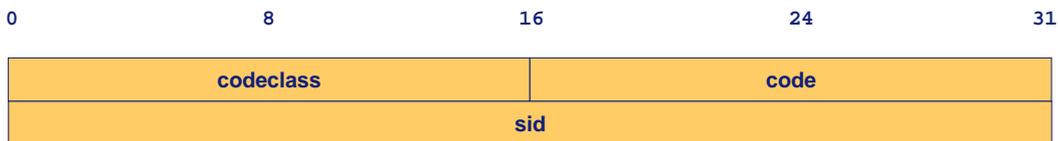


Figure 2-64: MIP6FWD-NATFW NSLP response message

- **codeclass:** specifies the codeclass of the response message.
- **code:** The result of the pinhole creation request. Possible values are SUCCESS (0x01) or FAILURE (0x02).
- **sid:** The session ID of the pinhole creation request. Used to identify the primarily request.

2.2.1.2.1.3 *Mfc* (NAT/FW NSLP – NSIS)

The interface **Mfc** implements the NAT/FW NSLP protocol [draft-ietf-nsis-nslp-natfw]. It communicates with the other NSIS implementation via the interface **Pfb**. The interface **Pfb** implements the NSIS protocol [draft-ietf-nsis-ntlp]. For more details on the NAT/FW NSLP see section 2.2.1.2.5 or [ENA-D2.1.2].

2.2.1.2.2 MIPL (HA)

This MIPL module implements the Home Agent functionalities. The same release used for the MN (MIPL 2.0.2) has been used as a starting point for the development on the HA.

The Mobile IPv6 firewall traversal has required some extensions to the MIPL code:

- When the MIPL implementation on the HA receives a specific message (e.g. BA, HoTI or bidirectional data traffic) it triggers the MIP6FWD via the interface **Hfa** to install the corresponding pinhole and waits for the response. The MIP6FWD itself again triggers the NAT/FW NSLP to trigger this pinhole via the interface **Hfb**. When the MIPL implementation receives a successful response it resumes the normal MIPL progress.
- The MIPL implementation on the HA communicates with the MIPL implementation on the MN and the CN using the interface **Pfa**, which implements the Mobile IPv6 protocol [RFC3775].

2.2.1.2.2.1 *Hfa* (MIPL – MIP6FWD)

Hfa is an inter-process interface implemented using a TCP socket. This interface is very similar to the interface **Mfa**, but reacts on different events. In contrast to the **Mfa** interface, which reacts on a new CoA, the **Hfa** interface interacts on the event of receiving a BU message, a HoTI message or on receiving bidirectional data traffic from the MN.

If this happens, it triggers the MIP6FW daemon to open pinholes between the nodes (e.g. for the HoTI message between the HA and the CN). Therefore, a trigger message (of type CREATE) is used, which signals to the MIP6FWD, the format of the required firewall pinhole. After triggering the MIP6FWD, the MIPL implementation is halted and waits for a response message from the MIP6FWD. With the response the MIP6FWD daemon informs the MIPL implementation about the success of the pinhole creation. These messages are similar to the messages as explained in section 2.2.1.2.1.1.

The rules that the MIPL and the MIP6FWD modules must observe include:

- As soon as the MIPL implementation receives a specific message (e.g. BA, HoTI or bidirectional data traffic), it sends a MIPTRIGGER_MSG_NSIS_CREATE message to the MIP6FWD daemon to install the corresponding pinhole and waits for the response. For the example, in receiving a HoTI, it triggers a firewall pinhole for the HoTI between HA and CN, as specified in [ENA-D2.1.2], section 3.4.3.
- When the MIP6FWD receives an MIPTRIGGER_MSG_CREATE message, it computes this request and triggers the NAT/FW NSLP to install the firewall pinhole as it is requested. It later informs the MIPL implementation about the success of the pinhole creation request.
- If the MIPL implementation receives a successful MIPTRIGGER_MSG_ACK to a MIPTRIGGER_MSG_CREATE message, it resumes the normal MIPL implementation. This could be to trigger the creation of further firewall pinholes, to trigger NSIS for a new re-configuration or the normal MIPL process. In our example, this means to send the HoTI to the CN, as it can now traverse the firewalls.

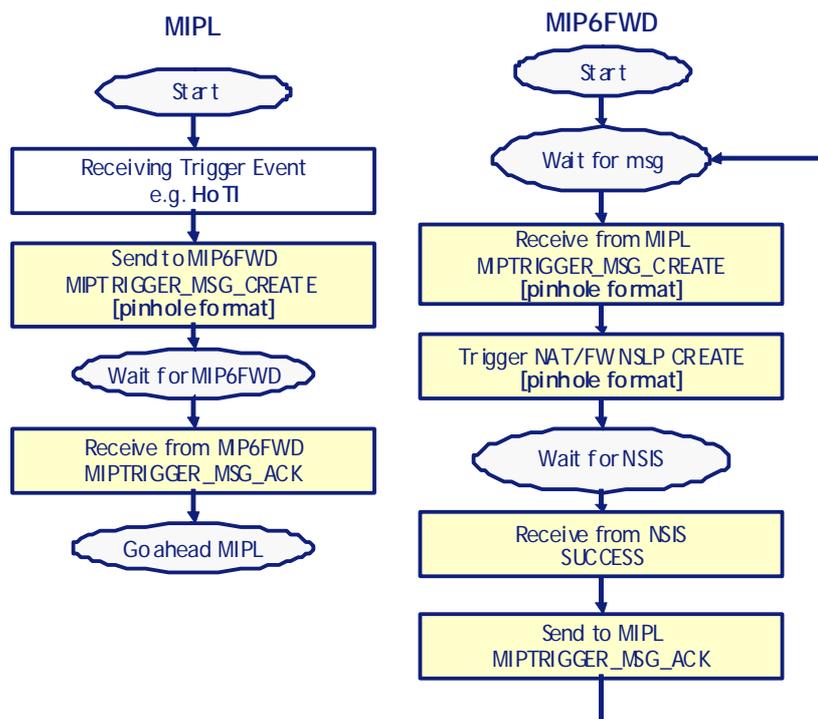


Figure 2-65: Interaction between MIPL and MIP6FWD (HA)

2.2.1.2.2.2 Hfb (MIP6FWD – NAT/FW NSLP)

Hfb represents the same interface as **Mfb**, described in section 2.2.1.2.1.2.

2.2.1.2.2.3 *Hfc* (NAT/FW NSLP – NSIS)

Hfc is the same interface as **Mfc**, which is described in detail in section 2.2.1.2.1.3.

2.2.1.2.3 MIPL (CN)

This MIPL module implements the Corresponding Node functionalities. The same release used for the MN (MIPL 2.0.2) has been used as a starting point for the development on the CN.

The Mobile IPv6 firewall traversal has required some extensions to the MIPL code:

- When the MIPL implementation on the CN receives a specific message (e.g. HoTI, CoTI or data traffic), it triggers the MIP6FWD via the interface **Cfa** to install the corresponding pinhole and waits for the response. The MIP6FWD itself again triggers the NAT/FW NSLP to trigger this pinhole via the interface **Cfb**. When the MIPL implementation receives a successful response it resumes the normal MIPL progress.

2.2.1.2.3.1 *Cfa* (MIPL – MIP6FWD)

Cfa is an inter-process interface implemented using a TCP socket. This interface is very similar to the interface **Mfa**, but reacts on different events. In contrast to the **Mfa** interface, which reacts on a new CoA, the **Cfa** interface interacts on the event of receiving a HoTI message, a CoTI message or on receiving/sending bidirectional/route optimized data traffic from/to the MN.

If this happens, it triggers the MIP6FW daemon in order to trigger a firewall pinhole between the nodes (e.g. for the HoT message between the CN and the HA). Therefore, a trigger message (of type CREATE) is used, which signals the MIP6FWD the format of the required firewall pinhole. After triggering the MIP6FWD, the MIPL implementation is halted and waits for a response message from the MIP6FWD. With the response the MIP6FWD daemon informs the MIPL implementation about the success of the pinhole creation. These messages are similar to the messages as explained in section 2.2.1.2.1.1.

The rules that the MIPL and the MIP6FWD modules have to observe are:

- As soon as MIPL implementation receives a specific message (e.g. HoTI, CoTI or data traffic), it sends a MIPTRIGGER_MSG_NSIS_CREATE message to the MIP6FWD daemon to install the corresponding pinhole and waits for the response. For the example of receiving a HoTI, it triggers a pinhole for the upcoming HoT between CN and HA, as specified in [ENA-D2.1.2], section 3.4.4.
- When the MIP6FWD receives an MIPTRIGGER_MSG_CREATE message, it computes this request and triggers the NAT/FW NSLP to install the firewall pinhole as it is

requested. It later informs the MIPL implementation about the success of the pinhole creation request.

- If the MIPL implementation receives a successful MIPTRIGGER_MSG_ACK to a MIPTRIGGER_MSG_CREATE message, it resumes the normal MIPL implementation. This could be the installation of further firewall pinholes, or the normal MIPL process. In our example, this means to send the HoT to the HA, as it now can traverse the firewalls.

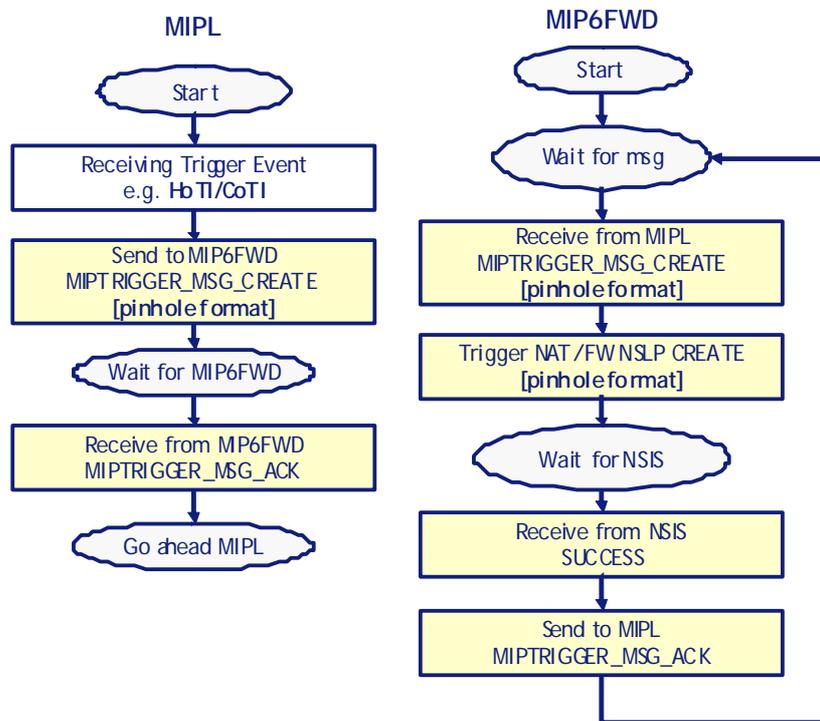


Figure 2-66: Interaction between MIPL and MIP6FWD (CN)

2.2.1.2.3.2 Cfb (MIP6FWD – NAT/FW NSLP)

Cfb represents the same interface as **Mfb**, described in section 2.2.1.2.1.2.

2.2.1.2.3.3 Cfc (NAT/FW NSLP – NSIS)

Cfc is the same interface as **Mfc**, which is described in detail in section 2.2.1.2.1.3.

2.2.1.2.4 MIP6FWD

The MIP6FWD is implemented by the University of Göttingen within ENABLE. It is used to translate and forward triggers from the MIPL implementation arrival via the interfaces **Mfa**, **Hfa** or **Cfa** to the NAT/FW NSLP via the interface **Mfb**, **Hfb** or **Cfb**. The NAT/FW NSLP uses the

interface **Mfc**, **Hfc** or **Cfc** to communicate with the NSIS NTLP. It also waits for the response of the requests and forwards it back via the arrival interface.

2.2.1.2.5 NSIS

The University of Göttingen has implemented the GIST protocol (which is the implementation of the NSIS NTLP protocol) in C++, using Linux 2.6 kernel. The implementation is fully conformant to the GIST protocol and it's API [draft-ietf-nsis-ntlp]. The code is publicly available in [http://user.informatik.uni-goettingen.de/~nsis].

The implementation architecture is shown in Figure 2-67. It is currently based on a single process approach using a main event loop based on [XORP] library, which is used to implement socket maintenance and callbacks as well as timer callbacks. This design has no additional overhead for maintaining and synchronizing multiple threads, which results in a high throughput and a rather simple implementation.

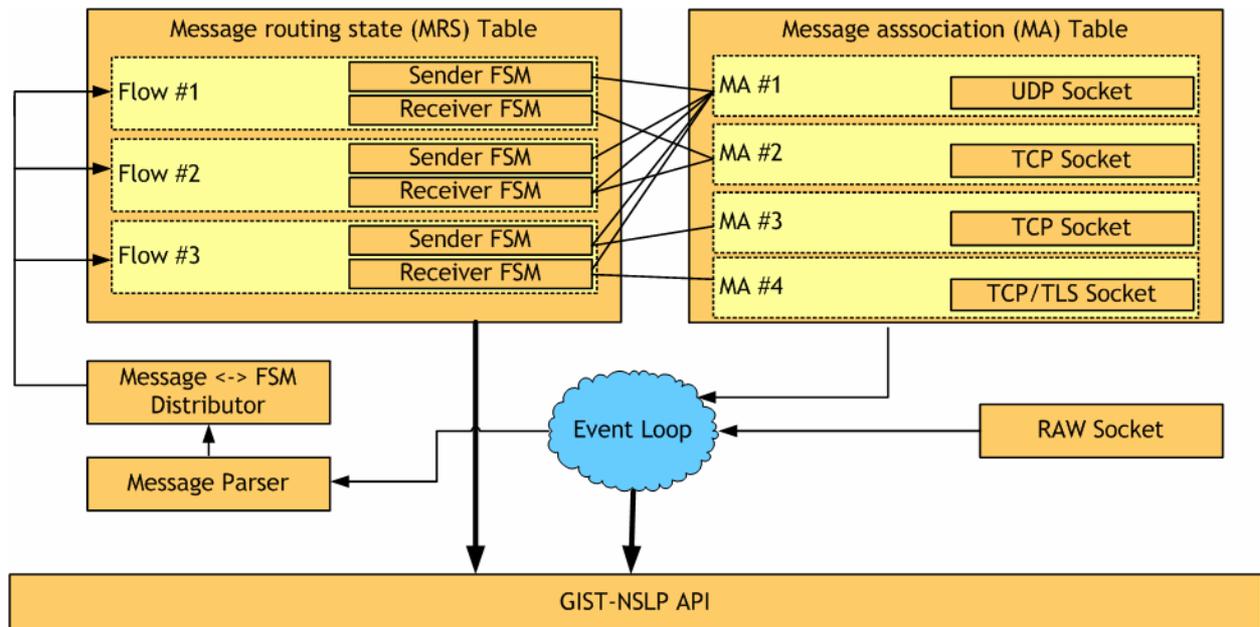


Figure 2-67 NSIS Implementation Architecture

Besides the event loop, a key component in the GIST implementation is state management. In order to support tens of thousands of signalling sessions efficiently, we used a hash table to manage the MRSs, associated with linked lists to resolve conflicts. A standard lookup takes a constant time, however in the worst case, all table entries would be compared to find a given MRS.

To search the MRS table, one needs to know the associated key information, namely the session ID, the NSLP ID and message routing information (MRI). This is nevertheless subject to some

limitations, e.g., it is not possible to search for all MRSs using a specific MRI. Such a search feature may be useful to find MRSs that are affected by a detected link failure. A possible solution is to maintain specialized hash tables for link failures, which would allow for quick searches. However, this approach would add maintenance overhead to every MRS table (which usually comprise a number of tables) operation.

In addition to managing MRSs, a GIST implementation has to manage MAs for C-mode operations. If two peers already have a MA and a new session is being established on the same path, the MA should be reused to minimize resource usage. This feature implies that there should be a way to search the MA table for an MA that can be reused for a certain session. Our implementation uses a second hash table to accomplish that goal. The upstream peer information (PI) serves as the key information. The UDP socket is treated as a “virtual” MA for the convenience of unifying the socket interface module.

Another important component of the GIST implementation is the finite state machine (FSM) to maintain states for each session. The GIST finite state machine [draft-ietf-nsis-ntlp-statemachine] is implemented based on a combination of the XORP timer class and an FSM framework that was originally written for the Linux ISDN device driver.

Every MRS is associated with two FSMs, one for the upstream peer and another one for the downstream peer. There is no need for a global table of FSMs, because every MRS provides pointers to the associated FSMs. In addition, every MA has a list of FSMs which it is associated with, so that the state machines can be informed e.g., when a loss of connectivity with its current peer takes place.

2.2.1.2.6 NAT/FW NSLP

The NAT/FW NSLP [draft-ietf-nsis-nsfp-natfw] daemon is implemented in userspace using C++. The code builds upon a GIST daemon that was developed at the University of Göttingen, both implementations are freely available in a single release [NSIS_UGOE] for Linux. The GIST daemon offers an API for NSLPs to use its generic transport services via UNIX sockets. NAT/FW NSLP daemon itself also offers an API to upper layers to allow applications to trigger signalling flows, such as accepting inbound connections at an edge firewall. As depicted in Figure 2-68, the implementation consists of six main parts:

- server core connecting to GIST-API and delegating callbacks to the other components,
- NAT/FW engine API,
- protocol behaviour defined in a finite state machine,

- message parsing and construction,
- security policy table and
- APIs to firewall and NAT.
- NAT/FW engines, (e.g. MIP6FWD) which enforce the NAT bindings and FW policy rules for corresponding data traffic, are connected to the NAT/FW NSLP daemon via the NAT/FW engine-API.

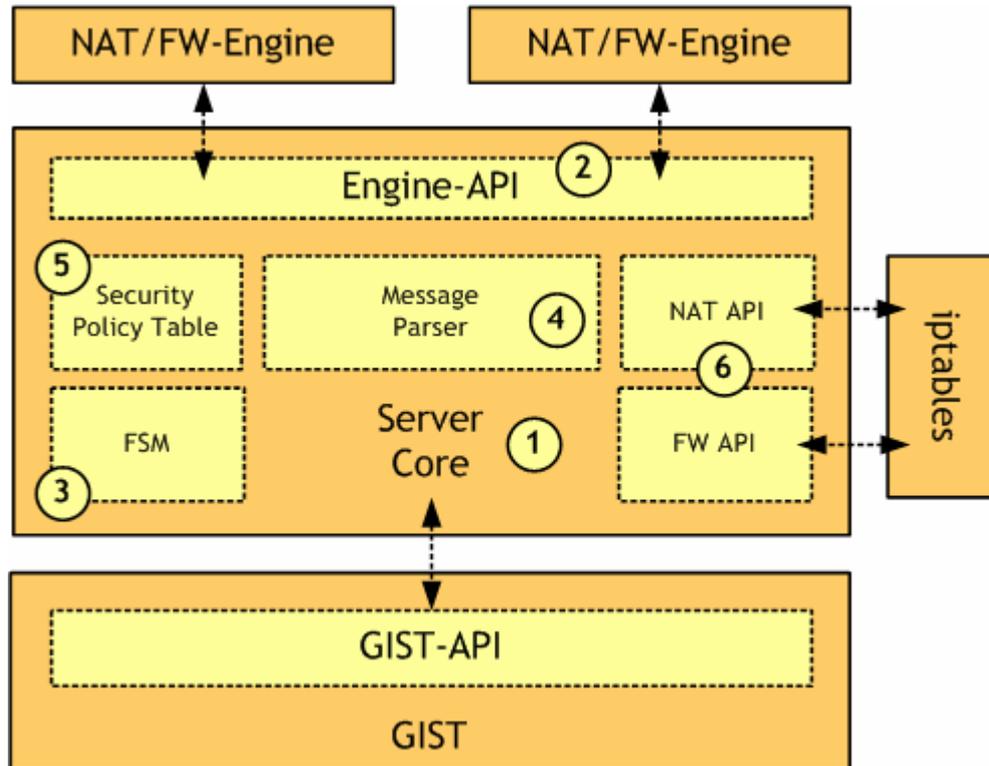


Figure 2-68: NAT/FW NSLP Architecture

We chose to use the Linux kernel netfilter [NETFILTER] module and its iptables front end as the NAT and firewall, because of its availability and complete coverage of required features. The use of the low-level iptables API *libiptc* is still discouraged by its developers because lack of robustness and missing documentation. To avoid problems and incompatibility with different iptables versions we chose to use a *system()* call to invoke an iptables process with according parameters, although this approach is known to be inefficient. NAT/FW NSLP imposes only a small set of requirements on the used firewall and NAT, as NAT/FW NSLP supports only adds a small subset of functionality to any possible firewall or NAT implementation and thus replacing the currently used firewall and NAT can be done easily.

It was shown during the GIST development that having an efficient finite state machine in source code that represents similar sets of states, transitions and actions as in the state machine specification, simplifies the understanding of the code without sacrificing performance. A C++ template was written to allow reusability among GIST and NSLP daemon development, enabling a mapping between the definition of a finite state machine, including states, transitions and actions to corresponding variables, function pointers and executable code.

The NAT/FW NSLP state machine [draft-werner-nsis-natfw-nslp-statemachine] lists three possible initial states, a host being in an initiator, a forwarder or a receiver idle state. The decision whether a message has to be forwarded or delivered can not be made solely on the destination address in GIST Message Routing Information (MRI) as in the NAT case; it is being rewritten, similarly as IP headers in NATs. Moreover, it depends on the current NAT configuration (or alternatively, often called reservation) status at the host where a message is received. The idea of the state machine giving a high-level overview for protocol understanding misses some aspects, such as locator rewrite and reservation dependency, that were fundamental aspects during implementation. Incoming message are processed by GIST and delivered to an NSLP if the NSLP is supported on that node. The NSLP decides whether to accept the message or to forward it. In the NAT/FW NSLP there are messages that are meaningful either just for NAT or just for firewall. The daemon configuration allows for the setting of flags to indicate whether a NAT/FW NSLP host is running a firewall, a NAT or both. After a message is accepted, basic validity checks are performed and the daemon will try to associate an existing state with the incoming message based on the session ID carried in NSLP payload. If there is no state installed yet, a new state machine object with a new session ID needs to be created. As mentioned above, it must be distinguished whether the host is the NR of the signalling path or whether it is a NF. Depending on the initial state, the protocol behaviour for this session is different. In contrast, if the action is triggered via the API the host is always the NI and a new state machine object with a new session ID is created. Now, as the state machine object is created and the initial state is determined, the transition is applied on it. Transitions are modelled as a set of states, events and function pointers on the state machine object. According to a given state and an incoming event, the corresponding function in the state machine is called. This keeps the function call overhead very small. The function bodies contain all relevant code that defines the protocol behaviour, such as state manipulation, NAT and firewall interaction, message parsing and construction.

Fa is the interface between the NAT/FW NSLP on the firewall and ip6tables. This interface translates the parameters of the firewall pinhole used inside the NAT/FW NSLP to a format adoptable by ip6tables. Furthermore, it builds a command line string which can install or delete the pinhole into/from the local firewall implementation and triggers this by using a *system()* system call.

Fc is the interface between the NAT/FW NSLP and NSIS and similar to **Mfc**, which is described in detail in section 2.2.1.2.1.3.

2.2.1.2.7 ip6tables

The ip6tables module implements the firewall functionalities. The NETFILTER iptables [NETFILTER] release 1.3.8 (or above) can be used. The Mobile IPv6 firewall traversal implementation requires the following patches/extensions to be installed on all firewalls:

- **mh:** This module add Mobility Header matches for IPv6.
- **rt:** This module add Routing Header IPv6 support.
- **dst:** This module allows to match the parameters in Destination Options Header.

2.2.2 Mobility optimisations

The goals of this FMIP6 system prototyping is to provide an implementation of the FMIPv6 protocol which is fully compliant with [RFC4068], which provides improved handover latency in MIP6 and to also demonstrate the developed FMIP6 and GSABA optimized mobility service as designed and studied in WP4. Given below is a reference software architecture along with the required modules and interfaces.

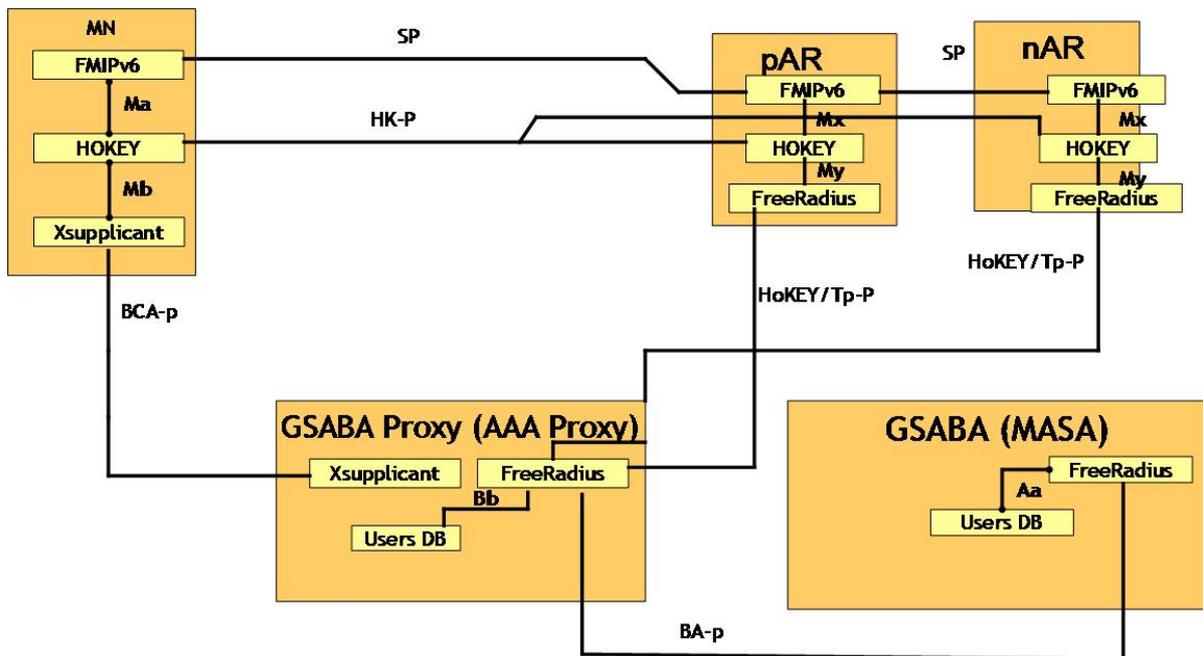


Figure 2-69 Modules and interfaces for fmip6 integrated with GSABA

The software architecture of the FMIPv6 / GSABA prototype contains the following components:-

- PAR
- NAR
- MN
- HA
- CN

The rectangles represent the software modules, namely FMIPv6, HOKEY, Xsupplicant, FreeRadius. Meanwhile the main interfaces are represented with blue connectors. For simplicity reasons the GSABA server and GSABA proxy have been co-located in the testbed development. Both software modules and interfaces are described in the following subsections.

2.2.2.1 Overview of FMIPv6 applicability to the GSABA architecture

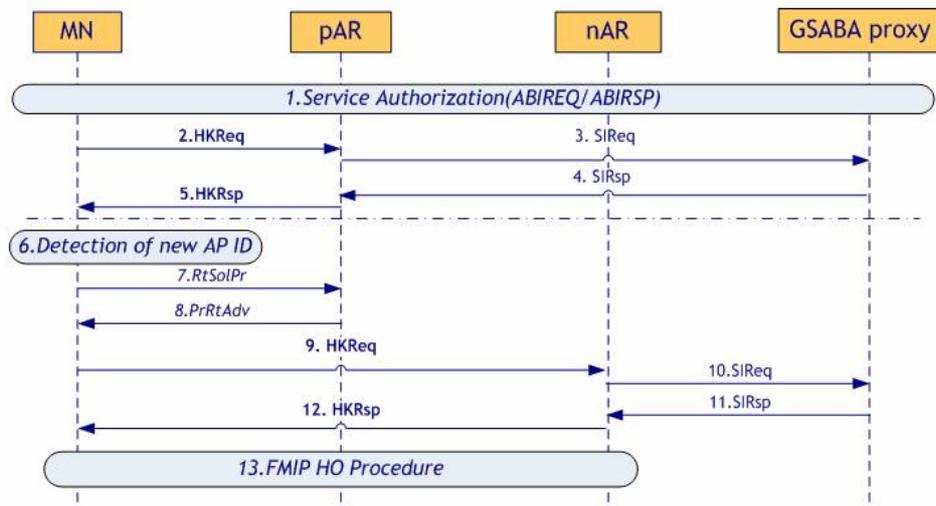


Figure 2-70 FMIPv6 integrated with GSABA message flow

Detailed description of the message for FMIPv6 applicability to the GSABA architecture is provided in [ENA-D4.2]. In Figure 2-70, only the relevant message flows (i.e. implemented interfaces) are shown. Functionality of the network nodes involved are discussed below.

2.2.2.1.1 MN

In Figure 2-70, after MN bootstrapping, the MN requests FMIP services by sending an ABIREQ (step 1) message to the GSABA proxy. The MN must specify its identifier, which is the BCID, and must indicate that it is requesting Fast Mobile IPv6 support. After MN gets fmip_{v6} service authorisation from GSABA Proxy/Server via the ABIRSP (step 1), it will request the handover keys from the current attached AR and the neighbour ARs which will be attached during Handover. Initially the MN sends the Handover Key Request (HKReq) message to the serving access router (pAR)(step 2). The MN creates the HKReq message including an NAI-like identifier (that was derived possibly during the time of HMK derivation), a message ID, and the care-of-address (CoA).

The MN also generates a nonce and includes it in the HKReq message. Further, the MN indicates the PRF algorithm that it chooses to use for key generation in the HKReq message. After neighbour AR detection and before fmip_{v6} handover, the HOKEY request message to nARs is sent by MN (step 9).

After successful HOKEY procedures, the handover keys will be used during the FMIPv6 protocol handover procedure. Please refer to the message flow above. A detailed description of how the FMIPv6 protocol utilises handover keys will be discussed in section 2.2.2.1.3.1 (i.e. the SP interface).

2.2.2.1.2 AR

After receiving the HOKEY request message from the MN and doing the relative processing of the message, the AR will send the SIRsp message to GSABA Proxy/Server.

If the AR receives a successful SIRsp message from the GSABA Proxy/Server, it MUST store the handover key received from the AAA server along with the CoA and MN ID and index it additionally with an SPI. The AR MUST send the SPI, AAA Nonce and lifetime received at the RADIUS message by the HKRsp message to the MN.

2.2.2.1.3 GSABA Proxy/Server

After received the ABIREQ message from the MN, the GSABA Proxy will verify it and do some judgments according to the fields in that message. Then the GSABA Proxy sends the ABIRSP message to the MN.

2.2.2.2 Implemented Software Modules

2.2.2.2.1 MN

2.2.2.2.1.1 Openssl

The result of openssl module at the MN side leads to the following interface:-

2.2.2.2.1.1.1 BCA-P

The BCA-p interface is used for the communication between MN and GSABA proxy. This interface is intended to serve two purposes. Firstly, the configuration parameters needed for a specific service can be requested from the MN. Secondly, the authorisation decision taken by the network is conveyed to the MN.

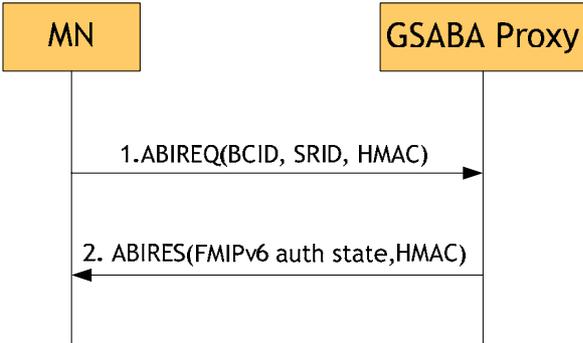


Figure 2-71 Message flows for FMIPv6 service Authorisation

After the GSABA key has been established, the MN and the GSABA proxy can exchange service configuration and authorisation information carried out by the ABIREQ and ABIRES messages via the BCA-p interface. The MN requests FMIP services by sending an ABIREQ message to the GSABA proxy, after checking the MN authorisation state against the previously downloaded user profile, the GSABA Proxy returns an ABIRES message to the MN, which includes the authorisation results.

During development, we used HTTP/TLS to realise this function. The software Openssl-0.9.8e is installed and configured at both MN side and GSABA Proxy side; the Apache-2.2.4 server is installed and configured at GSABA Proxy side.

2.2.2.2.1.2 HOKEY (MN)

Huawei has fully implemented the HOKEY software module using C programming language in Linux ‘Ubuntu’ distribution with kernel version 2.6.16. This software module is in complete compliance with the [HOKEY] IETF draft. The result of HOKEY module implementation is the creation of the HK-p interface and Ma internal interface. Given below is the description of each of these interfaces.

2.2.2.2.1.2.1 HK-p

The HK-p is an external/network interface between the MN and AR. Through this interface, the MN can retrieve the Handover Key (HK) to protect the relative messages between the MN and AR. With respect to the FMIPv6 application, the HK-p interface includes two aspects: one is the interface between the MN and pAR. The other is the interface between the MN and nAR.

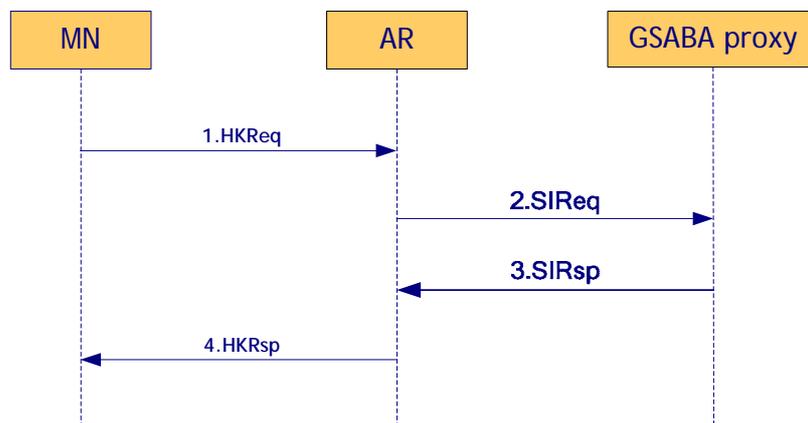


Figure 2-72 Message Flow for HOKEY

After bootstrapping and service authorisation, the MN sends the Handover Key Request (HKReq) message to the serving access router (pAR). The MN creates the HKReq message including an NAI-like identifier (which was derived possibly during the time of HMK derivation), a message ID, and the care-of-address (CoA). The MN also generates a nonce and includes it in the HKReq message. Further, the MN indicates the PRF algorithm that it chooses to use for key generation in the HKReq message. The MN includes a MAC of the message fields in an MN-AAA MAC Mobility sub-option. In order to obtain replay protection, the MN SHOULD use the Timestamp mobility option. Upon successful delivery of HKReq, the HOKEY module in the module waits for HKRsp message from the pAR.

Upon receiving a successful HKRsp message from the AR, the MN MUST ensure that the message contains the MN-AR MAC mobility option. If not, it MUST silently discard the

message. The MN MUST ensure that the Message ID matches with that of the corresponding HKReq. If there is a mismatch, it MUST drop the packet. The MN MUST compute the handover key using the keying material contained in the HKRsp message. The MN MUST verify the AUTH Value in the MN-AR MAC mobility option using the HK derived. The MAC algorithm used is the one specified in the Algorithm Type field of the MN-AR MAC mobility sub-option. If the MAC algorithm is not supported by the MN, it MUST discard the message. If the AUTH Value verification fails, the MN MUST silently discard the message.

Upon successful processing of the HKRsp and derivation of the valid HK, the MN MUST store the SPI and lifetime associated with the key, as sent in the HKRsp.

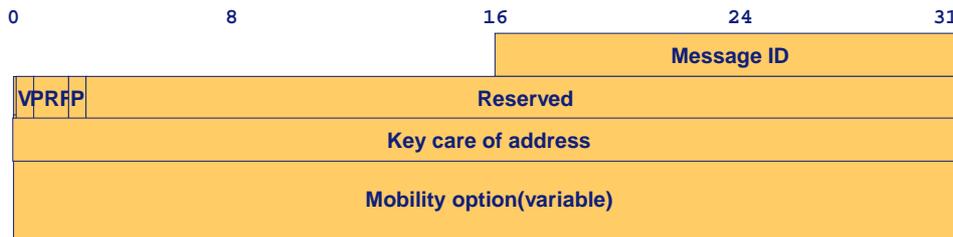


Figure 2-73 HKReq mobility header

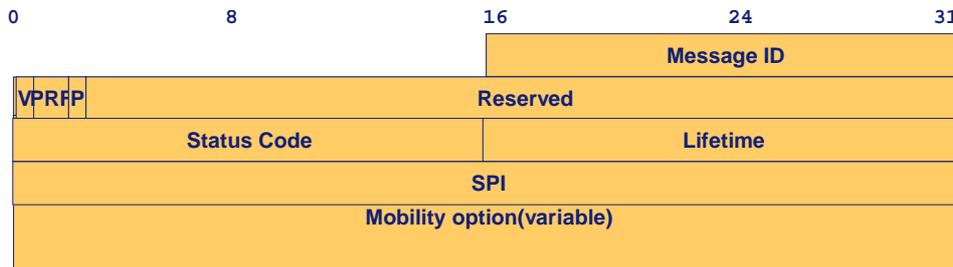


Figure 2-74 HKRsp mobility header

2.2.2.2.1.2.2 Ma/Mb

After the HK processing with pAR, the MN will store the HK (between MN and pAR) and the relative parameters (such as SPI and lifetime). This should be taken place between step5 and step7 in Figure 2-70.

After the MN finishes the scanning and before handover processing, the MN will do the HK processing with the recommend nAR (Maybe there are more than one nAR, but at present, we suppose there is only one nAR), after the HK processing with nAR, the MN will store the HK (between the MN and nAR) and the relative parameters (such as SPI and lifetime).

So, there are at least 2 sets of HK parameters existing sometime in MN. We designed a structure to store the HK relative parameters. The structure is:

```
struct hk_para{
    uint16_t lifetime;
    uint32_t spi;
    uint8_t hk[12]; /*96 bits*/
    struct in6_addr *ar_addr; /*to distinguish which AR the HK belong to*/
};
```

Then we define the global variable: `struct hk_para ar_hk_para;`

After the MN handles the received successful HKRsp message from the pAR, the MN will store the HK parameters in the ar_hk_para structure.

After the MN handles the received successful HKRsp message from the nAR, the MN will store the HK parameters in the ar_hk_para structure also.

There are 2 other global variables defined:

```
struct hk_para mn_par_hk_para;
struct hk_para mn_nar_hk_para;
```

While the MN does the scanning procedure, we can copy the ar_hk_para variable to mn_par_hk_para. And during the MN handover procedure, we copy the ar_hk_para to the mn_nar_hk_para.

It is now that the fmip module can get the HK (between MN and pAR) from mn_par_hk_para variable to protect the FBU/FBack message, and can get the HK (between MN and nAR) from mn_nar_hk_para variable to protect the FNA message.

Notice: the code for more than one nAR is for future study.

2.2.2.2.1.3 FMIPv6 (MN)

The FMIPv6 module in the MN uses the open source code developed by [FMIPv6]. Brunel University has modified the source code in a Linux ‘Ubuntu’ distribution under the 2.6.16 kernel version. The FMIPv6 at the MN is used for all the FMIPv6 signalling which is compliant with

[RFC4068]. The modifications/extensions made by Brunel contribute to the creation of the SP interface.

2.2.2.2.1.3.1 SP

The SP interface is service specific and is the interface between the MN and ARs. The protocol used to implement this interface is [FMIPv6] which is available as open source and distributed under the GNU General Public License.

It is very important to note here that source code in the MN needs to be modified to have a new mobility (authentication) option. This allows the Fast Binding Update (FBU) to be securely sent (i.e. in terms of integrity of the message, and authentication) from the MN to the pAR. The Fast Binding Acknowledgement (FBack) **sent by the pAR in response to the FBU needs to be secured in the same way.** Also, the FNA sent by the MN to the nAR needs to be secured using the MAC mobility option as well. The new authentication option should be in the form of a MAC (Message Authentication Code) and described in [RFC 4285]. A cryptographic **message authentication code (MAC)** is a short piece of information used to authenticate a message. A MAC algorithm accepts as input a secret key and an arbitrary-length message to be authenticated, and outputs a MAC (sometimes known as a tag). The MAC value protects both a message's integrity as well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the message content.

The format of the MN-pAR and MN-nAR (i.e.FBU, FBack and FNA) mobility message authentication option is defined in Figure 2-75. This option uses the subtype value of 1. The MN-pAR mobility message authentication option is used to authenticate the FBU and FBack messages based on the shared-key-based security association between the MN and pAR. Similarly, the MN-nAR mobility option is used to authenticate the FNA message. The shared-key-based mobility security association between the MN and the pAR used within this specification consists of a mobility SPI, a key, an authentication algorithm, and the replay protection mechanism in use. The mobility SPI is a number in the range [0-4294967296], where the range [0-255] is reserved. The key consists of an arbitrary value and is 16 octets in length. The authentication algorithm is HMAC_SHA1. The replay protection mechanism may use the Sequence number as specified in [RFC3775] or the Timestamp option as defined in Section 6. If the Timestamp option is used for replay protection, the mobility security association includes a "close enough" field to account for clock drift. A default value of 7 seconds *should* be used. This value *should* be greater than 3 seconds.

The mobility message authentication option in the FBU and FBack *must* be the last option in a message with a mobility header if it is the only mobility message authentication option in the message.

The encryption algorithms for HMAC available in Linux (Ubuntu distribution, kernel 2.6.16) are available in a package called **libdigest-hmac-perl (1.01-1)**. The encryption algorithms included in the packages are MD5 and SH1.

The shared secret key (in this case, the HK key as shown in figure 1-4) is derived from the Handover Master Key (HMK).

The authentication data is calculated on the message starting from the mobility header up to and including the mobility SPI value of this option.

Authentication Data = First (96, HMAC_SHA1(MN-pAR Shared key, Mobility Data))

Mobility Data = care-of address | home address | Mobility Header (MH) Data

MH Data is the content of the Mobility Header up to and including the mobility SPI field of this option. The Checksum field in the Mobility Header *must* be set to 0 to calculate the Mobility Data. The first 96 bits from the Message Authentication Code (MAC) result are used as the Authentication Data field.



Figure 2-75 MAC mobility option

Option Type:

AUTH-OPTION-TYPE value 9 has been defined by IANA. An 8-bit identifier of the type mobility option.

Option Length:

8-bit unsigned integer, representing the length in octets of the Subtype, mobility Security Parameter Index (SPI) and Authentication Data fields.

Subtype:

A number assigned to identify the entity and/or mechanism to be used to authenticate the message.

Mobility SPI:

Mobility Security Parameter Index

Authentication Data:

This field has the information to authenticate the relevant mobility entity. This protects the message beginning at the Mobility Header up to and including the mobility SPI field.

2.2.2.2.2 ARs (pAR & nAR)

The ARs also use the open source code developed by [FMIPv6]. Brunel University has modified the source code in a Linux 'Ubuntu' distribution under the 2.6.16 kernel version. The FMIPv6 at the AR is used of all the FMIPv6 signalling which is compliant with [RFC4068]. With the ENABLE architecture requiring secure FMIPv6 handover, the following interfaces were created.

2.2.2.2.2.1.1 Hokey/Tp-p

Tp interface is used for communication between Access Router and GSABA Server. SIREQ and SIREs messages are exchanged on this interface. The SIREQ message contains CoA, MN ID, message ID, life time and SPI. There are no existing AVPs for sending all the parameters. So we have used two different protocols for this purpose. MN ID is sent via standard radius protocol and remaining parameters are sent via standard SQL query which directly updates the user Database. SQL updates are only allowed from specified IP addresses of ARs on the database server to prevent un-authorised access.

The Radius packet has the following format.

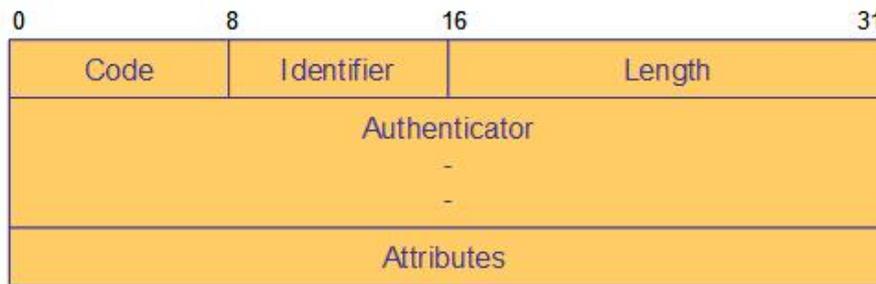


Figure 2-76 Tp Interface Message format

- Code: specifies type of RADIUS packet
- Identifier: specifies the RADIUS response with the correct outstanding request
- Length: specifies length of packet
- Authenticator: sixteen octets long and contains the information that the RADIUS client and server use to authenticate each other
- Attributes: is a section where an arbitrary number of attributes are stored.

The attribute section of the packet is used to send MN ID to GSABA server, if it is verified correct, GSABA would reply with AAA-Nonce, GSABA time-stamp and a handover key.

2.2.2.2.2.1.2 Mx/My

If the AR receives a successful AAA response message from the AAA server, *it must* store the handover key received from the AAA server along with the CoA and MN ID and index it additionally with an SPI.

The AR **MUST** send the SPI, AAA Nonce and lifetime in the RADIUS message in the HKRsp message to the MN. The AR **MUST** include a MAC of the message created using the HK in the MN-AR MAC Mobility Sub-Option carried in the HKRsp message.

When AR sends the HKRsp message to the MN, it should use the HK to calculate the MAC to protect the HKRsp message; and during the handover procedure, the AR should use the HK to protect the FBack message.

So the Mx interface functionality is how the HOKEY module to get the HK when calculating the MAC.

So we can define a global structure variable to store the received HK from AAA at the AR side. The structure is:

```
struct hokey_para{
    uint32_t spi;
    uint8_t hk[12]; /*96 bits*/
    struct in6_addr *coa;
    struct in6_addr *mn_id;
    uint16_t status_code,
    uint16_t lifetime,
    uint8_t ho_nonce[16]
    uint8_t timestamp[8]
};
```

Then we define the global variable: struct hokey_para mn_ar_hokey_para

After receiving a successful AAA response (SIRsp), the AR stores the HK parameters to the mn_ar_hokey_para global variable, and when the HK is needed at the AR side, we can get the HK from this variable.

The following parameters:

```
uint16_t status_code,
uint16_t lifetime,
uint32_t spi,
uint8_t ho_nonce[16]
uint8_t timestamp[8]
```

will be used through the hokey_ar_send_hkrsp() function to be sent to the MN. They can be read from the global structure variable: mn_ar_hokey_para.

2.2.2.2.2.2 HOKEY (AR)

After receiving the HKReq message, the AR *must* first determine if it has a pending request from the MN with the same message id. If so and if the AR has already received the AAA response corresponding to the HKReq, the AR *should* retransmit the HKRsp to the MN. For further protection from replays, the rate of retransmissions of responses to MN *must* not be more than a preconfigured RETRANS_RATE. If the AR already forwarded this message to the home GSABA Proxy but has not yet received a response from GSABA Proxy, the AR *must* silently discard the retransmitted request from the MN. If the HKReq message is the new request from MN, pAR *must* first ensure that it has a valid neighbour cache entry for the CoA claimed by the MN. The pAR further verifies the validity and uniqueness of the CoA claimed by the MN. After verifying the validity of the CoA of the MN, pAR forwards the HKReq message to the GSABA Proxy (using the SReq message).

After the AR receives a successful response message from the GSABA Proxy, it *must* store the handover key received from the AAA server along with the CoA and MN ID and index it additionally with an SPI. The AR **MUST** send the SPI, AAA Nonce and lifetime in the RADIUS message in the HKRsp message to the MN. The AR **MUST** include a MAC of the message created using the HK in the MN-AR MAC Mobility Sub-Option carried in the HKRsp message.

2.2.2.2.2.3 FMIPv6 (AR)

The FMIPv6 module in the pAR and nAR uses the open source code of [FMIPv6]. Like the FMIPv6 module at the MN, the FMIPv6 in the ARs (both the pAR and nAR) have been modified/extended by Brunel University. The FMIPv6 protocol at the pAR is responsible for sending the FBack upon receipt of the FBU from the MN. The following interfaces are created and used by FMIPv6 module at ARs (i.e. pAR and nAR)

2.2.2.2.2.3.1 SP

The pAR and nAR will use the SP interface to securely send the FBACK and FNA respectively to the MN. For this reason, the source code [FMIPv6] in the ARs has been modified to have a new mobility authentication option as described in section 2.2.2.2.1.3.1.

The format of the MN-pAR and MN-nAR (i.e.FBU, FBack and FNA) mobility message authentication option is defined in Figure 2-75.

2.2.2.2.3 GSABA Server

Bb/Aa is an internal interface of the GSABA server which is used between the user database and FreeRadius. FreeRadius supports different database servers including MySQL which is used in our case. Simple SQL queries are run over this interface.

3. FINAL TEST-BED DESIGN

The following sections illustrate the different test-beds used for the three major developments undertaken in the ENABLE projects.

3.1 Test-bed for Integrated Software

The test-bed for integrated software components is illustrated in Figure 3-1. This test-bed has been used to test and demonstrate the following ENABLE functionalities:

- EAP-based MIPv6 bootstrapping for integrated scenarios.
- DHCPv6 MIPv6 bootstrapping for integrated scenarios.
- DNS-based bootstrapping for split scenarios.
- MIPv6 authentication based on IKEv2 (and HoA provisioning).
- Diameter interface between HA and MASA AAA server for service authorisation.
- HA load-sharing.
- IPv4 extensions for MIPv6 including support for roaming in IPv4-only (AN6 and AN7) and dual stack access network (AN1 and AN2).

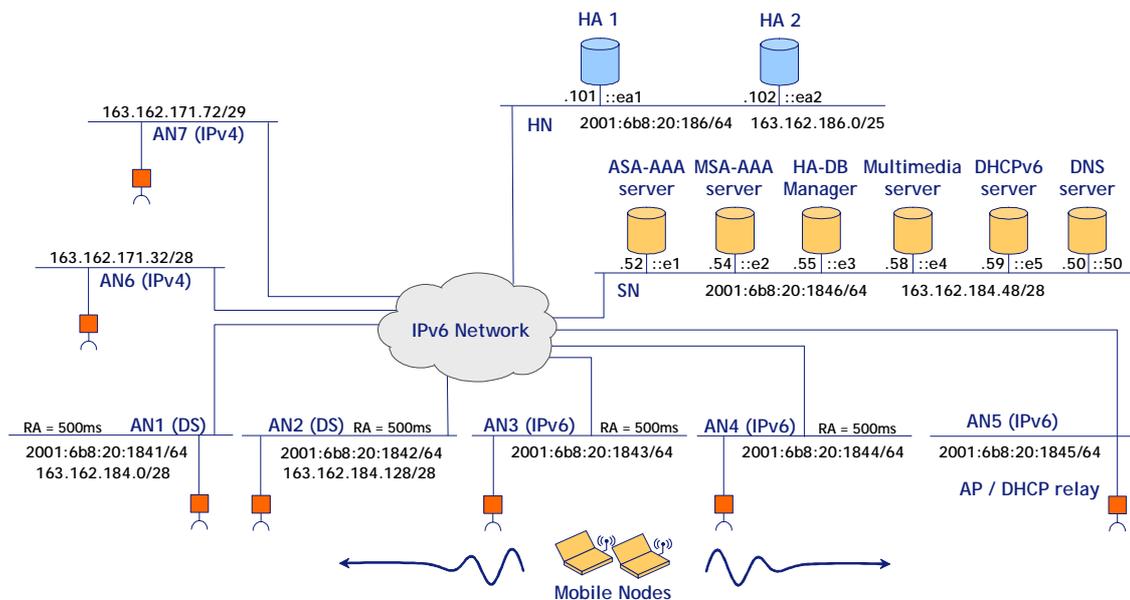


Figure 3-1 Final Test-bed for integrated components

The Mobile Node is not supposed to enter the Home Network. This assumption is reasonable since it is likely that the HN will be implemented as a virtual network by mobile operators. The Home Network has been populated with two Home Agents to demonstrate the HA load sharing functionalities.

All the machines, apart from mobile nodes and the DNS server, have been implemented as virtual machines installed on a powerful VMware ESX 3.0 server. Basically, the starting point for all of them was a cloning operation from a clean installation of an Ubuntu 6.0.6 (Dapper) server. As MNs, two Compaq nc8000 laptops and a Nokia 770 PDA are used.

For demonstration purposes all the access networks were accessible both as open and through EAP authentication. For example, access network 1 (AN1) was freely accessible using as essid “ENABLE_AN1” or through EAP authentication using the “ENABLE_EAP_AN1” essid. An exception to this rule is access network 5 which is only accessible through EAP authentication and it is the only one supporting DHCPv6-based bootstrapping. Since DHCPv6 support required modification on the AP behaviour, for access network 5 a Linksys WRT54GL using Linux OpenWRT (KAMIKAZE bleeding edge, r5974) was used. For other access networks two unmodified Cisco Aironets 1200 (IOS IOS C1200-K9W7-M, 12.3(7)JA3) were used. They were configured in virtual ap mode to support multiple essid in order to avoid the necessity of an access point for every single access network.

3.2 Test bed for NSIS / Mobile IPv6 firewall traversal

The test-bed for integrated software components is illustrated in Figure 3-2. This test-bed will be used to test and demonstrate the ENABLE Mobile IPv6 firewall traversal functionalities.

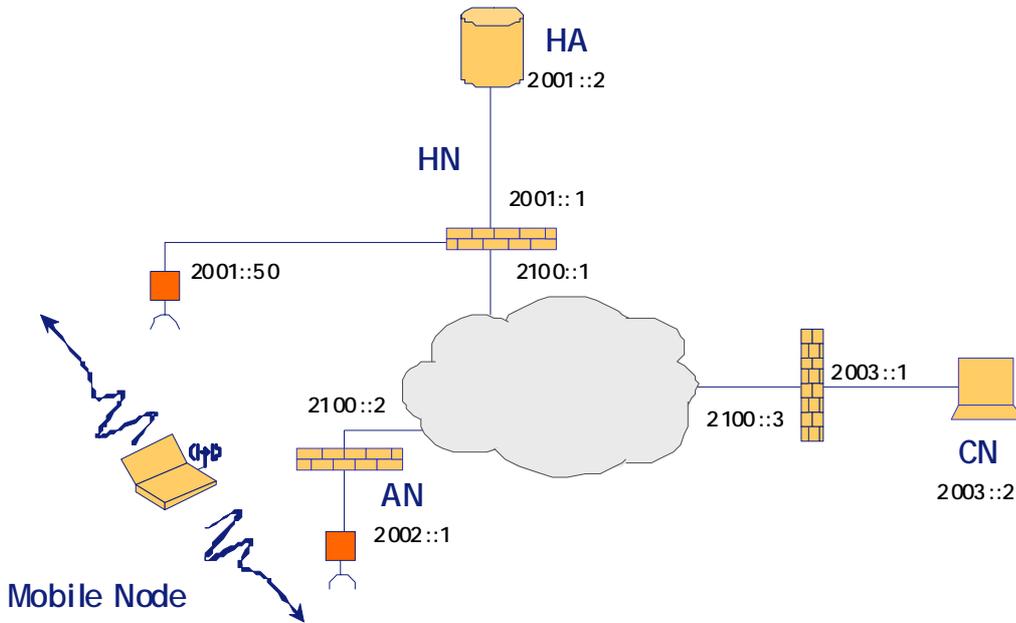


Figure 3-2 Final Test-bed for Mobile IPv6 Firewall Traversal

The Mobile Node will start in the Home Network and later enter a foreign network. During this handover, the Mobile IPv6 firewall traversal process is performed automatically.

The HA, the CN and the firewall at the edge of the CN network have been implemented as virtual machines installed on a VMware ESX 3.0 server. Basically, the starting point for all of them was a cloning operation from a clean installation of an Ubuntu 6.0.6 (Dapper) server. For the firewall on the access network side, it is integrated into a wireless access router which is based on a small form factor mini-itx based computer which can also run an installation of the Ubuntu 6.0.6 (Dapper) OS. The MN for the demonstration is a Sony laptop.

3.3 Test bed for Mobility Optimisations

The test bed for Mobility Optimisation is illustrated in Figure 3-3. This test bed was used to test and demonstrate the optimised mobility based on FMIPv6.

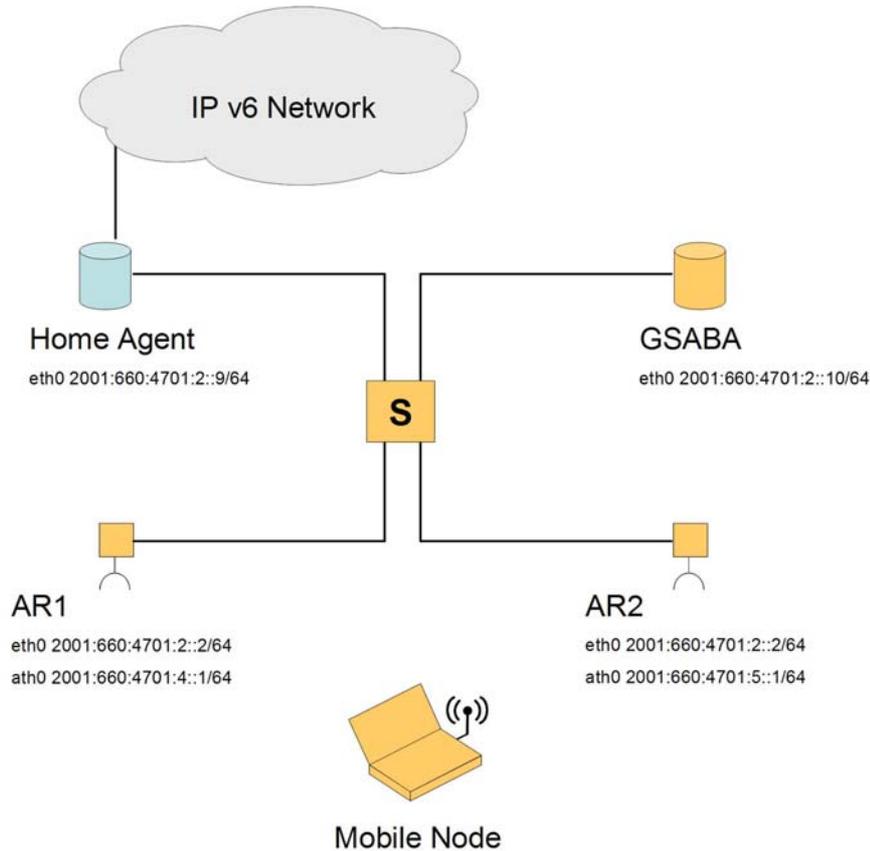


Figure 3-3 Test-bed for Optimised Mobility

The MN will start connecting to one of the access points and later on it will switch to the other one, while receiving data on its HoA. During this handover, the modified FMIPv6 procedures would be followed. At startup, the FMIPv6 service will be authorised by the GSABA server, then the MN will send a HKreq to the pAR. In response to the HKreq, the pAR will contact the GSABA for HK and will reply with a HKres. The same procedure would be followed for contacting nAR. On receiving a HKrsp from the nAR, the MN will send a FBU to the pAR and then the pAR will send a HI to the nAR. The nAR will reply with a HAck. Once the HAck is received a FBack will be sent to the MN, which will send a FNA to the nAR.

The GSABA is implemented in VMware for portability and rest of the elements are implemented on actual machines.

4. INSTANTIATION OF THE APPLICATION SCENARIO

In [ENA-D6.1] a realisation of the ENABLE MIPv6 service environment was described using a search and rescue scenario. The scenario described a number of scenes where multiple networks, actors and devices interoperate with each other. A search and rescue scenario is ideal for the MIPv6 service environment as there is an abundance of technologies interoperating with each other. In addition, due to the emergency aspect of the scenario the quality of the connection between the actors and the ASPs must be both reliable and seamless due to the risk involved of a dropped data exchange.

This section shall present the link between the application scenario as described in [ENA-D6.1], and the instantiation of the ENABLE demonstration. This section shall show how the main actor in these search and rescue scenes comes in and goes through the ENABLE MIPv6 service environment test bed, accessing the different networks (EAP/non-EAP, IPv6-only, IPv4-only and dual-stack). It shall also describe the visualisation software and shall show an example of how the ENABLE MIPv6 service environment is demonstrated to interested third parties.

4.1 Definition of the ENABLE Demonstration

In order to define the ENABLE demonstration, mappings have to be done between the two trial scenes in the search and rescue scenario, specifically Scene 3 (not enough assets on site, volunteers called in) & Scene 6 (ambulance transports the victim from rescue scene to hospital) and the test case scenarios as detailed in Section 5.1, and the screenshots of the test bed as shown in Appendix A.

4.1.1 SAR Scene 3 Demonstration

Scene 3 is described in [ENA-D6.1] as follows:

“Once all vehicles and people have been deployed around the search location area, further assets may be required onsite if the location of the search area grows, e.g. there are not enough SAR practitioners to cover an area. In this scenario extra volunteers may be called in to the site to aid in the searching. In addition to people, extra equipment may also be brought to the site. These additional personnel called into the search will be one of the main actors played in the rescue scenario as they may move between IPv6, IPv4-only, and dual stacked networks “

In the first action of events in scene 3 John represents an example of a SAR volunteer, called by the search teams whenever there is a lack of resources at the search site. John has a MN with three network interfaces including WLAN/WMAN, 3G and LAN. In this scene John bootstraps

his mobile node in an access network that is currently open with no authentication; it supports a DS of IPv4 and IPv6 and requests his home agent through a DNS query. This sequence of events is represented using the following test cases and graphical test bed references.

Table 4-1 Mapping of Scene 3 to Test Cases & Demonstration of Split Bootstrapping

Test Case References	Test Bed Graphical References
ENA-TCS-001	split_bs_dnsquery_2001-6b8-20-186—eaX_163-162-186-10X
ENA-TCS-005	split_bs_ike_michele_anY_2001-6b8-20-186—Z-eaX
ENA-TCS-009	bs_autz_michele_anY_ha1

In the next sequence of events, once John has successfully bootstrapped and is communicating with his selected HA a video call is established to the SAR base. This call will be continued throughout his journey on the way to be collected by the search and rescue personnel. The handover between various IPv6 and IPv4 networks is represented by the following test cases and graphical test bed references.

Table 4-2 Mapping of Scene 3 to Test Cases & Demonstration of Handovers

Test Case References	Test Bed Graphical References
ENA-TCS-009	handover_michele_haY_anX
ENA-TCS-007	
ENA-TCS-008	

4.1.2 SAR Scene 6 Demonstration

Scene 6 is described in [ENA-D6.1] as follows:

“The main content of Scene 6’s descriptive text involves an ambulance picking up the rescue victim and transporting him in the Ambulance to the hospital location. This scene includes specific mobility issues, challenges and domain issues, taking into account IPv4 interworking and IPv6 Middlebox traversal to mention but some. Environmental Issues and assumptions must also be taken into consideration when completing a more detailed work flow of this scene in order to incorporate and deal with all aspects of the scene.”

The first action in Scene 6 is where John switches back on his MN as he is on the way back to the hospital. He is out of range of the incident control vehicle and only has a signal to his home network. Johns MN needs to bootstrap in an EAP, dual-stack network, which requires EAP-based authentication, using the integrated mechanism with the IKE-PSK optimisation. However, on John’s home network his operator has a number of ENABLE functions being carried out in the background. Firstly the HA-DB Manager is at periodic intervals continuing to collect

information about the state of the Home Agents in the network. The parameters including bandwidth, number of registrations and Region IDs, which are stored in the HA-DB manager. The MSA-AAA server is then periodically querying the HA-DB Manager to identify which HA is currently the one with the lowest bandwidth or registrations etc. Both the query to the HAs and the SQL query to the HA-DB manager can be found in the test cases and test bed graphical scenarios as described in the following table.

Table 4-3 Mapping of Scene 3 to Test Cases & Demonstration of HA Load Sharing

Test Case References	Test Bed Graphical References
ENA-TCS-002	integ_hal_snmp
	integ_hal_sql

Before leaving the scene John switches on his MN, again bootstrapping of John's mobile node will take place. He then starts a video call to make sure vital information regarding the patient reaches the hospital. The victim is loaded onto the ambulance and his vitals are continuously been sent from the ambulance on the journey to the hospital. On his way to the hospital John is constantly handing over from one network to another including IPv6 and IPv4 networks. These are primarily his home network which is protected by EAP and also other networks which use the DHCPv6 mechanism to allow John send information from his PDA. The sequence of events for the EAP based network and DHCPv6 based scenarios can be found in the following test case scenarios and Test Bed Graphical references.

Table 4-4 Mapping of Scene 3 to Test Cases & Demonstration of Integrated Bootstrapping with DCHPv6

Test Case References	Test Bed Graphical References
ENA-TCS-003	integ_psk_auth_alex_haX_2001-6b8-20-186—Z-eaX
ENA-TCS-004	bs_autz_alex_anX_haX
ENA-TCS-005	integ_bs_dhcp_an5_ha1_2001-6b8-20-186--ea1
ENA-TCS-008	integ_bs_dhcp_an5
ENA-TCS-009	integ_bs_dhcp_ha1_2001-6b8-20-186--ea1
	integ_psk_auth_ivano_haX_2001-6b8-20-186—Y-X
	bs_autz_ivano_anX_haX

4.2 Trial of the ENABLE Demonstration

Bringing the ENABLE demonstration to life comprises of a number of different software components that work in a serial fashion to demonstrate both visually (from a command line) and graphically (a flash application) exactly how all the different aspects of the project come together in ENABLE MIPv6 service environment. The demonstration is comprised of the following architecture.

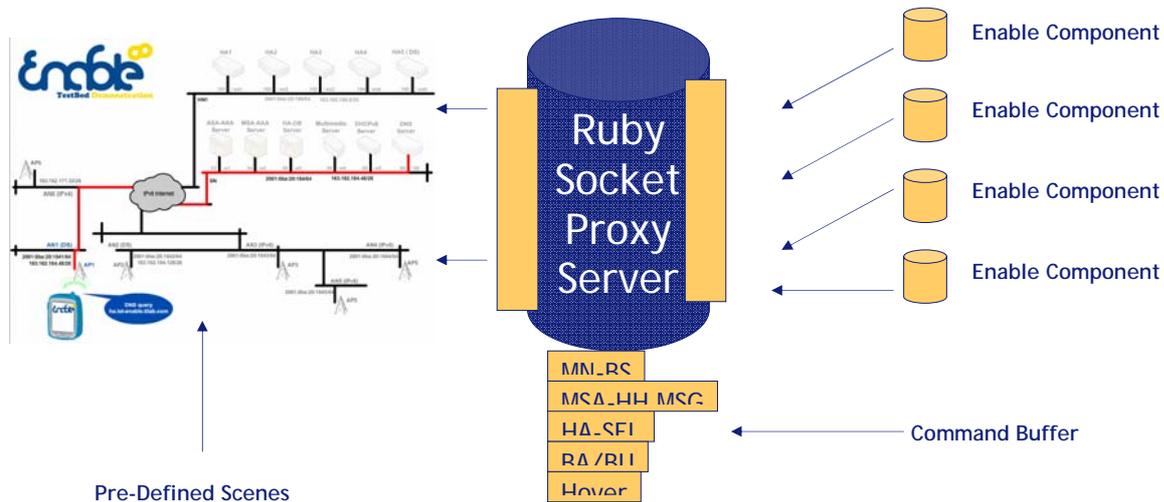


Figure 4-1 Flash, Ruby Server and Enable Component/Module Interaction

The trial demonstration comprises of four main parts, the Test bed, ENABLE developed software components, a Flash application and a Ruby Server.

4.2.1 Trial Test bed

The test bed contains all of the hardware and virtual servers that are part of the demonstration. These include the DNS, DHCP, MSA-AAA, ASA-AAA, HA's etc. These components are represented on the graphical end of the demonstration and are described in detail in Section 3.

4.2.2 ENABLE Software Components

The ENABLE software components have been slightly enhanced to contain code trigger mechanisms that send information to the demonstration ruby socket server which will in turn trigger an event to the Flash Application.

4.2.3 Demonstration Visualisation Application

A Flash [FLASH] application is the graphical end of the demonstration. It will display exactly how the ENABLE components are interworking together and what steps are required to carry out

specific actions. For example, the HA selection process has a number of steps including SNMP data collection. These will be represented by a graphical animation on the flash application. The flash application was created using Adobe Flash CS3 using action script 2.0.

The Flash application connects to the Ruby socket server on port 7777 listening for commands. Once a command and its parameters are received the Flash application will play and dynamically insert information into the required visualisation clip. This information is currently parameterised with specific data. For example on the HA relocation visualisation clip the parameters may contain specific information such as HA Bandwidth or number of connected nodes. This allows the demonstration to graphically display real time information coming from the test bed and the ENABLE software modules.

4.2.4 Ruby Server

The ruby server is a socket based IPv6/IPv4 server written in Ruby on Rails [RUBY]. It contains a dual port server listening on port 7777 and port 8888. The server listens for commands on port 7777 from the ENABLE software components. The flash application connects to the server on port 8888 and listens for commands to play specific movie clips that relate to the incoming enable module request. If the request to play a visualisation clip comes before the previous one has been completed they get added to a buffer. The buffer will keep adding requests for visualisation clips onto the server.

```
# start the infrastructure server to monitor message from infrastructure
nodes
iserver = InfrastructureServer.new(7777, host_v4, ticker)
iserver.start(-1)

# start the flash server to manage flash clients
fserver = FlashServer.new(12346, host_v6, ticker)
fserver.start(-1)
```

Figure 4-2 Example of configuration code for Ruby Server

A ruby server was installed on an Ubuntu 7.04 machine running inside a virtual machine which is available from the code repository. Once unpacked it can be run by firstly configuring which addresses it must bind to in “Application.rb”. Once the addresses have been defined other ports and parameters can be found in the configuration file enable_server.rb.

4.2.5 Demonstration Example

A typical scenario of how the test bed, ENABLE components, flash demo and ruby server would be instantiated would be described as follows.

In the split scenario a mobile node must bootstrap against its MSA-AAA server. Firstly, the MN is turned on by the user. A DNS request is sent to the DNS server in the test bed. The DNS server is continuously monitoring for DNS requests and notices a request for ha.ist-enable.tilabs.com. Code on the DNS server now sends a request to the ruby server to start the visualisation “split_bs_dnsquery_2001-6b8-20-186—eaX_163-162-186-10X”. The request is received by the Ruby server and added to the buffer. Since it’s the only command in the buffer it will automatically be passed onto the flash application. The flash application now plays the requested visualisation simulating exactly how the operations are being carried out by the real devices on the network. However, this visualisation plays for a running time of 5 seconds. In the meantime the mobile node is continuing to authenticate against the HA and the HA is verifying information with the MSA-AAA. This potentially means there is an overlap of visualisation clips. To prevent this happening, as the DNS visualisation is playing the next visualisation clip will be buffered on the Ruby server. Once finished the next logical clip in the sequence will play.

5. TEST MANAGEMENT AND METHODOLOGY

The objective of the testing phase is to validate the functionality and correctness of the overall MIPv6 system as defined by the ENABLE project and to detect the differences between the specified behaviour of the system as per the defined ENABLE architecture [ENA-D1.1] and the observed behaviour during the test execution. These tests should be conducted the way a normal user would interact with the system, providing to the user an experience of the full system. This system testing can be classified as black box testing, since the team does not need to be aware of how the system is internally built.

Therefore for ENABLE the testing methodology includes:

- Definition of the Tests Cases: Contains a description of the different test cases needed to validate the system. Tests cases must be described according to the template defined in Appendix B.
- Set up of the test environment: For conducting the test plan, an adequate testing environment must be set up. The ENABLE testing environment is fully described in Section 3.
- Tests execution, according to test specification: Test descriptions and results should be kept in files to allow easy handling and analysis at different times. However, in some cases it might be unavoidable to perform tests manually.
- Document test results: A document describing the results of tests (test passed/not passed) execution must be produced according to a specific template (see Appendix B). Besides, errors, and bugs must be reported using a ticket tracking tool as described in [ENA-D6.1]
- Analysis of identified weaknesses in the system: From this analysis, suggestions for improvement must be derived.

It is obvious though, that ENABLE does not provide any direct user interface as it is an infrastructural based technology, therefore functionality of ENABLE developed components are difficult to illustrate and test due to the fact that the components are not visible on an actual user interface. In this way visualisation tools can provide a means to convey the sequence of operations occurring behind the scene, and can also prove useful during the initial integration and then testing phases. The visualisation tool as described in Section 4 was used:

- For all system test cases where there will be an expected “output” from the activity of an ENABLE component.

- However a number of test cases (“notifications”) will have to be initiated through a manual or automated/simulated intervention at some point in the network.

This section is intended to describe the System Test Methodology which would be followed in ENABLE and to provide a full list of test cases that would be used during the System Test conformance phase.

5.1 Test Cases

There are different types of testing that can be carried out, from White Box Testing, mainly intended to unit tests, where the software development team are looking to find errors in low level operations of an ENABLE component. More high level tests than this can be described as:

- **Black Box Testing:** In this granularity of behavioural testing, the testing team is looking to find errors in high-level operations, at the level of features, operational profiles and end – user scenarios. The type of tests defined here are functional tests based on ‘what’ the system should do. These tests are designed to exercise the extremes, interfaces, boundaries and error conditions of the system. Simply playing around with the average conditions of the system is not an effective technique for behavioural testing, a structured, methodical and repeatable sequence of tests and test conditions that probe the suspected system weaknesses is a must. It is the intention of this section of the document to describe the ENABLE approach to this task.
- **Live Demonstration Tests:** A demonstration of the ENABLE architecture involves putting end users, content experts and early adopters in front of the system, encouraging them to use and explore the key concepts of the system. This type of testing will be provided by the official demo site.

ENABLE system tests will focus on Black Box Testing, and the following section contains detailed descriptions of the different system test cases needed to validate ENABLE architecture.

5.1.1 Summary of Test Case Scenarios

Table 5-1 Test Case Summary



Case Test ID	Test Case Name:
ENA-TCS-001	Split bootstrapping
ENA-TCS-002	HA Load Sharing
ENA-TCS-003	Integrated Bootstrapping EAP
ENA-TCS-004	Integrated Bootstrapping DHCPv6
ENA-TCS-005	Handover IPv4-Interworking
ENA-TCS-006	Split bootstrapping PDA
ENA-TCS-007	Handover PDA
ENA-TCS-008	VoIP Call with Streaming Audio
ENA-TCS-009	Streaming Video with Handover
ENA-TCS-010	Pinhole creation for BU/BA after handover
ENA-TCS-011	Pinhole creation for BT/RO data traffic
ENA-TCS-012	Pinhole deletion
ENA-TCS-013	Mobility Optimised Handover
ENA-TCS-014	Streaming with Mobility Optimised Handover

5.1.2 Integrated Software Test Cases

5.1.2.1 Enable Test Case Scenario 001: Split bootstrapping

Table 5-2 ENA-TCS-001

	<h2>Enable Test Case Scenario 001</h2>
Test Case ID:	ENA-TCS-001
Test Case Name:	Split bootstrapping

Test Case Description:	<p>The MN1 bootstraps in AN1 (open, dual-stack) using the split mechanism. The MN1 discovers HA1 (or HA2) IPv4/v6 addresses through a DNS query "ha.ist-enable.org", with DNS round-robin load sharing being used.</p>		
Work Package Reference:	WP1, WP4		
		Software Module	
Components under test:	MN1	MIPL	ike2d-mn
	HA1	MIPL	diametermip6
		ike2d-ha	
	MSA	msad	Users DB
Testbed Nodes involved:	DNS		
	AN1	Open DS (IPv4/IPv6) network	
Protocols/lfs involved:	MN1 -> DNS	Ph	
	MN1 -> HA1	Pb	
	HA1	Hc, Hd	
	HA1 -> MSA	Pf, Pg	
	MSA	Ab	
Steps Taken:	Step ENA-TCS-001-01	MN makes a DNS query for ha.ist-enable.tilab.com	
	Step ENA-TCS-001-02	IKEv2 (HoA Assignment) is done and SA between MN & HA is created	
	Step ENA-TCS-001-03	MN sends BU to HA. The HA sends a Diameter MIP6-Authorisation-Request (MAR) to the MSA AAA server. MSA-AAA checks MN's profile, and replies with a Diameter MIP6-Authorisation-Answer MN receives BA from HA	
Expected Results:	MN1 with the identity <michele@ist-enable.tilab.com > is authorised to use the mobility service through HA1 on AN1.		

Actual Results:	
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No
Test Conducted by:	

5.1.2.2 Enable Test Case Scenario 002: HA Load Sharing

Table 5-3 ENA-TCS-002

		Enable Test Case Scenario 002	
Test Case ID:	ENA-TCS-002		
Test Case Name:	HA Load Sharing		
Test Case Description:	HA Loading Sharing parameters are collected via SNMP from each HA, saved into HA-DB (interval HA_Ptime) and load parameters of all HAs are retrieved from the HA-DB Manager via SQL (interval HA-DB_Ptime) by the MSA.		
Work Package Reference:	WP3		
		Software Module	
Components under test:	HA1	MIPL	NETSNMP
	HA2	MIPL	NETSNMP
	HA-DB Manager	HA Manager	HA DB
Testbed Nodes involved:	MSA	HA select	
	HN	Home network (IPv4/IPv6)	

Protocols/lfs involved:	HA1, HA2	He
	HA1, HA2 -> HA-DB Mgr	Pc
	HA-DB Mgr	Da
	HA-DB Mgr-> MSA	Pd
Steps Taken:	Step ENA-TCS-002-01	The HA-DB Mgr initiates a periodic SNMP query to collect HA loading sharing parameters from each HA, retrieving them every interval HA_Ptime.
	Step ENA-TCS-002-02	For each HA, HA loading sharing parameters are stored in the HA DB.
	Step ENA-TCS-002-02	The MSA initiates a periodic SQL query for load parameters of all Has, retrieving them from the HA-DB Manager every interval HA-DB_Ptime.
Expected Results:	The MSP-AAA can evaluate which HA to assign to the next registering MN, based on load sharing parameters.	
Actual Results:		
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive	
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low	
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Test Conducted by:		

5.1.2.3 Enable Test Case Scenario 003: Integrated Bootstrapping EAP

Table 5-4 ENA-TCS-003

	<h2>Enable Test Case Scenario 003</h2>
---	--

Test Case ID:	ENA-TCS-003		
Test Case Name:	Integrated Bootstrapping EAP		
Test Case Description:	MN2 bootstraps in AN2 (EAP, dual-stack network), which requires EAP-based authentication, using the integrated mechanism with the IKE-PSK optimisation.		
Work Package Reference:	WP1, WP3		
		Software Module	
Components under test:	MN2	MIPL	Xsupplicant
		ike2d-mn	
	ASA	Freeradius	Users DB
	HA2	MIPL	diametermip6
		ike2d-ha	
Testbed Nodes involved:	AN2	EAP DS (IPv4/IPv6) network	
Protocols/lfs involved:	MN2	Ma, Mb	
	MN2->ASA	Pe	
	ASA	Aa	
	ASA->MSA	Ac	
	MN2 -> HA2	Pa, Pb	
	HA2	Hc, Hd	
	HA2->MSA	Pf, Pg	
Steps Taken:	Step ENA-TCS-003-01	MN2 initiates EAP-based bootstrapping	
	Step ENA-TCS-003-02	ASA makes a HA selection request towards the MSA with the weighted load sharing parameters being No. of Registrations.	

	Step ENA-TCS-003-03	MSA-AAA calculates HA selection reply by looking at No. of Registrations for each HA. Load HA1: 1 MN registered Load HA2: 0 MN registered HA2 is selected
	Step ENA-TCS-003-04	ASA returns EAP-based bootstrapping Success to MN2 with HA2 IPv6 Address returned. MN2 is authorised to use the AN2 network
	Step ENA-TCS-003-05	IKEv2 (HoA Assignment) is done and SA between MN2 & HA2 is created
	Step ENA-TCS-003-06	MN2 sends BU to HA2. The HA2 sends a Diameter MIP6-Authorisation-Request (MAR) to the MSA AAA server. MSA-AAA checks MN2's profile, and replies with a Diameter MIP6-Authorisation-Answer MN2 receives BA from HA2
Expected Results:		The load sharing assigns MN2 to HA2 since HA1 is already serving MN1. MN2 with the identity <alex@ist-enable.tilab.com > is authorised to use the mobility service through HA2 on AN2.
Actual Results:		
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive	
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low	
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Test Conducted by:		

5.1.2.4 Enable Test Case Scenario 004: Integrated Bootstrapping DHCPv6

Table 5-5 ENA-TCS-004

		Enable Test Case Scenario 004		
Test Case ID:		ENA-TCS-004		
Test Case Name:		Integrated Bootstrapping DHCPv6		
Test Case Description:		MN4 bootstraps in AN5, requiring EAP-based authentication, but with DHCPv6 bootstrapping used.		
Work Package Reference:		WP1		
			Software Modules	
Components under test:		MN4	MIPL	Xsupplicant
			ike2d-mn	DHCPv6 Client
		ASA	Freeradius	Users DB
		HA1	MIPL	diametermip6
			ike2d-ha	
		MSA	HA select	
Testbed Nodes Involved		DCPv6 server		
		AN5	EAP IPv6 only network	
Protocols/lfs involved:		MN4	Ma, Mb	
		MN4->ASA	Pe	
		ASA	Aa	
		ASA->MSA	Ac	
		MN4 -> HA1	Pa, Pb	
		HA1	Hc, Hd	

	HA1->MSA	Pf, Pg
	NAS	Pi, Pk
Steps Taken:	Step ENA-TCS-004-01	MN4 initiates EAP-based bootstrapping
	Step ENA-TCS-004-02	ASA makes a HA selection request towards the MSA with the weighted load sharing parameters being No. of Registrations.
	Step ENA-TCS-004-03	MSA-AAA calculates HA selection reply by looking at No. of Registrations for each HA. Load HA1: 1 MN registered Load HA2: 2 MN registered HA1 is selected
	Step ENA-TCS-004-04	ASA returns EAP-based bootstrapping Success, passing through NAS in AN5 HA1 IPv6 Address returned. MN4 is authorised to use the AN5 network.
	Step ENA-TCS-004-05	MN4 initiates DHCPv6 Information Request. DHCPv6 Server returns HA1 IPv6 Address.
	Step ENA-TCS-004-06	IKEv2 (HoA Assignment) is done and SA between MN4 & HA1 is created
	Step ENA-TCS-004-07	MN4 sends BU to HA1. The HA1 sends a Diameter MIP6-Authorisation-Request (MAR) to the MSA AAA server. MSA-AAA checks MN4's profile, and replies with a Diameter MIP6-Authorisation-Answer MN4 receives BA from HA1
Expected Results:	MN4 with the identity <ivano@ist-enable.tilab.com > is authorised to use the mobility service through HA1 on AN5.	
Actual Results:		
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive	

027002	ENABLE	D6.2:Report on final prototypes, network integration and validation	
Priority:	<input type="checkbox"/> Immediate	<input type="checkbox"/> High	<input type="checkbox"/> Medium <input type="checkbox"/> Low
Severe Defect:	<input type="checkbox"/> Yes	<input type="checkbox"/> No	
Test Conducted by:			

5.1.2.5 Enable Test Case Scenario 005: Handover IPv4Interworking

Table 5-6 ENA-TCS-005

		Enable Test Case Scenario 005	
Test Case ID:	ENA-TCS-005		
Test Case Name:	Handover IPv4Interworking		
Test Case Description:	MN1 moves on to an IPv4-only network (AN6)		
Work Package Reference:	WP1, WP2		
Components under test:	MN1	MIPL	
	HA1	MIPL	
Testbed Nodes involved:	AN1	IPv6 only network	
	AN6	IPv4 only network	
Protocols/lfs involved:	MN1 -> HA1	Pb	
Steps Taken:	Step ENA-TCS-005-01	MN1 moves into range of AN6 & out of range of AN1	
	Step ENA-TCS-005-02	MN1 sends BU to HA1. MN1 receives BA from HA1	
Expected Results:	MN1 with the identity <michele@ist-enable.tilab.com > is authorised to use the mobility service through HA1 on AN6.		

Actual Results:	
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No
Test Conducted by:	

5.1.2.6 Enable Test Case Scenario 006: Split bootstrapping PDA

Table 5-7 ENA-TCS-006

		Enable Test Case Scenario 006	
Test Case ID:	ENA-TCS-006		
Test Case Name:	Split bootstrapping PDA		
Test Case Description:	<p>The MN3 (PDA) bootstraps in AN3 (open, v6-only) using the split mechanism. The MN3 discovers HA1 (or HA2) IPv4/v6 addresses through a DNS query “ha.ist-enable.org”, with DNS round-robin load sharing being used.</p>		
Work Package Reference:	WP1, WP4, WP6		
Components under test:		Software Module	
	MN3	MIPL	ike2d-mn
	HA1	MIPL	diametermip6
		ike2d-ha	
	MSA	msad	Users DB
Testbed Nodes involved:	DNS		

	AN3	Open IPv6 network	
Protocols/lfs involved:	MN3 -> DNS	Ph	
	MN3 ->HA1	Pb	
	HA1	Hc, Hd	
	HA1 -> MSA	Pf, Pg	
	MSA	Ab	
Steps Taken:	Step ENA-TCS-006-01	MN3 makes a DNS query for ha.ist-enable.tilab.com	
	Step ENA-TCS-006-02	IKEv2 (HoA Assignment) is done and SA between MN3 & HA1 is created	
	Step ENA-TCS-006-03	MN3 sends BU to HA1. The HA1 sends a Diameter MIP6-Authorisation-Request (MAR) to the MSA AAA server. MSA-AAA checks MN's profile, and replies with a Diameter MIP6-Authorisation-Answer MN3 receives BA from HA1	
Expected Results:	MN3 with the identity <luca@ist-enable.tilab.com > is authorised to use the mobility service through HA1 on AN3.		
Actual Results:			
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive		
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low		
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No		
Test Conducted by:			

5.1.2.7 Enable Test Case Scenario 007: Handover PDA

Table 5-8 ENA-TCS-007

	<h2>Enable Test Case Scenario 007</h2>
---	--

Test Case ID:				ENA-TCS-007				
Test Case Name:				Handover PDA				
Test Case Description:				MN3 moves on to an IPv4-only network (AN7), and then onto IPv6-only network (AN4)				
Work Package Reference:				WP1, WP2, WP4, WP6				
				Software Module				
Components under test:				MN1		MIPL		
				HA1		MIPL		
Testbed Nodes involved:				AN3		Open IPv6 only network		
				AN7		Open IPv4 only network		
				AN4		EAP IPv6 only network		
Protocols/lfs involved:				MN1 -> HA1		Pb		
Steps Taken:				Step ENA-TCS-007-01		MN3 moves into range of AN7 & out of range of AN3		
				Step ENA-TCS-007-02		MN3 sends BU to HA1. MN3 receives BA from HA1		
				Step ENA-TCS-007-03		MN3 moves into range of AN4 & out of range of AN7		
				Step ENA-TCS-007-03		MN3 sends BU to HA1. MN3 receives BA from HA1		
Expected Results:				MN3 with the identity <luca@ist-enable.tilab.com > is authorised to use the mobility service through HA1 on AN3, AN7 & AN4 without significant interruption to its mobility service.				
Actual Results:								
Test Case Status:				<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive				
Priority:				<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low				

Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No
Test Conducted by:	

5.1.2.8 Enable Test Case Scenario 008: VoIP Call with Streaming Audio

Table 5-9 ENA-TCS-008

		Enable Test Case Scenario 008	
Test Case ID:	ENA-TCS-008		
Test Case Name:	VoIP Call with Streaming Audio		
Test Case Description:	Upon request MN2 and MN4 start a sip-call session between them, then MN2 is to perform a handovers among EAP-networks.		
Work Package Reference:	WP1, WP2, WP4, WP6		
Components under test:	MN2	MIPL	SIP Client App
	MN4	MIPL	SIP Client App
	HA2	MIPL	
Testbed Nodes involved:	Multimedia Server	SIP Proxy	
	AN2	IPv6 only network	
	AN4	IPv6 only network	
Protocols/ifs involved:	MN2 -> HA2	Pb	
	MN2 -> MN4	SIP	
Steps Taken:	Step ENA-TCS-008-01	MN2 and MN4 run their SIP Client application which registers with the ENABLE SIP proxy	

	Step ENA-TCS-008-02	MN2 initiates a SIP call session towards MN4. MN4 answers the call and live streaming audio is present
	Step ENA-TCS-008-03	MN2 moves into range of AN4 & out of range of AN2
	Step ENA-TCS-008-04	MN2 sends BU to HA2. MN2 receives BA from HA2
	Step ENA-TCS-008-05	Check the presents of audio streaming between MN2 and MN4
Expected Results:	During the MN2 network access transition the SIP call session remains active and once MN2 is attached to its new network, audio conversation can be exchanged again between MN2 and MN4	
Actual Results:		
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive	
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low	
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Test Conducted by:		

5.1.2.9 Enable Test Case Scenario 009: Streaming Video with Handover

Table 5-10 ENA-TCS-009

	Enable Test Case Scenario 009
Test Case ID:	ENA-TCS-009
Test Case Name:	Streaming Video with Handover
Test Case Description:	Upon request MN4 starts a video-streaming session with the ENABLE Multimedia server. Then MN4 is to perform handovers among EAP-networks and Open networks.
Work Package Reference:	WP1, WP2, WP4, WP6

		Software Modules	
Components under test:	MN4	MIPL	Video Client App
	HA2	MIPL	
Testbed Nodes involved:	Multimedia Server	Video Streamer	
	AN4	IPv6 only network	
	AN5	IPv6 only network	
	AN1	DS (IPv6/IPv4) network	
Protocols/lfs involved:	MN4 -> HA1	Pb	
	MN2 -> MN4	SIP	
Steps Taken:	Step ENA-TCS-009-01	MN4 initiates a video streaming session towards the multimedia server and live streaming video is presented on MN4	
	Step ENA-TCS-009-02	MN4 moves into range of AN4 & out of range of AN5	
	Step ENA-TCS-009-03	MN4 sends BU to HA1. MN4 receives BA from HA1	
	Step ENA-TCS-009-04	Check the presents of video streaming on MN4	
	Step ENA-TCS-009-05	MN4 moves into range of AN1 & out of range of AN4	
	Step ENA-TCS-009-06	MN4 sends BU to HA1. MN4 receives BA from HA1	
	Step ENA-TCS-009-07	Check the presents of video streaming on MN4	
Expected Results:	During the MN4 network access transitions the streaming video session remains active and once MN4 is attached to its new network, the video remains playing.		
Actual Results:			
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive		

Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No
Test Conducted by:	

5.1.3 Mobile IPv6 Firewall Traversal Test Cases

5.1.3.1 Enable Test Case Scenario 010: Pinhole creation for BU/BA after handover

Table 5-11 ENA-TCS-010

	Enable Test Case Scenario 010		
Test Case ID:	ENA-TCS-010		
Test Case Name:	Pinhole creation for BU/BA after handover		
Test Case Description:	MN performs handover and then the firewall pinholes for BU/BA should be installed		
Work Package Reference:	WP1, WP2, WP6		
		Software Modules	
Components under test:	MN	MIPL	MIP6FWD,NSIS
	HA	MIPL	MIP6FWD,NSIS
Testbed Nodes involved:	MN-FW	NSIS	
	HA-FW	NSIS	
Protocols/lfs involved:	MN<->HA	Pfa	
	MN<->FW<->HA	Pfb	

Steps Taken:	Step ENA-TCS-010-01	MN starts in home net and then perform handover
	Step ENA-TCS-010-02	Before sending BU through HA, MN performs firewall pinhole creation for BU and BA
	Step ENA-TCS-010-03	MN sends BU to HA MN receives BA from HA
Expected Results:	The firewall pinholes should be installed at the MN-FW and the HA-FW as described in D2.1.2.	
Actual Results:		
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive	
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low	
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Test Conducted by:		

5.1.3.2 Enable Test Case Scenario 011: Pinhole creation for BT/RO data traffic

Table 5-12 ENA-TCS-011

	Enable Test Case Scenario 011		
Test Case ID:	ENA-TCS-011		
Test Case Name:	Pinhole creation for BT/RO data traffic		
Test Case Description:	MN wants to communicate with CN and the firewall pinholes for BT/RO data traffic should be installed		
Work Package Reference:	WP1, WP2, WP6		
Components under test:		Software Modules	
	MN	MIPL	MIP6FWD,NSIS
	HA	MIPL	MIP6FWD,NSIS

	CN	MIPL	MIP6FWD,NSIS
Testbed Nodes involved:	MN-FW	NSIS	
	HA-FW	NSIS	
	CN-FW	NSIS	
Protocols/lfs involved:	MN<->HA	Pfa	
	HA<->CN	Pfa	
	MN<->CN	Pfa	
	MN<->FW<->HA	Pfb	
	HA<->FW<->CN	Pfb	
	MN<->FW<->CN	Pfb	
Steps Taken:	Step ENA-TCS-011-01	MN wants to start communication with CN	
	Step ENA-TCS-011-02	Before sending data packets (either BT or RO), MN performs firewall pinhole creation for BT and RO data traffic	
	Step ENA-TCS-011-03	MN sends data traffic to CN	
Expected Results:	The firewall pinholes should be installed at the MN-FW, HA-FW and the CN-FW as described in D2.1.2.		
Actual Results:			
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive		
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low		
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No		
Test Conducted by:			

5.1.3.3 Enable Test Case Scenario 012: Pinhole deletion

Table 5-13 ENA-TCS-012

	<h2>Enable Test Case Scenario 012</h2>		
Test Case ID:	ENA-TCS-012		
Test Case Name:	Pinhole deletion		
Test Case Description:	MN performs test cases ENA-TCS-010 and ENA-TCS-011 and move back to home net. Firewall pinholes which are no longer needed should be removed automatically after a specific timeout.		
Work Package Reference:	WP1, WP2, WP6		
		Software Modules	
Components under test:	MN-FW	NSIS	
	HA-FW	NSIS	
	CN-FW	NSIS	
Testbed Nodes involved:			
Protocols/lfs involved:			
Steps Taken:	Step ENA-TCS-012-01	MN performs test cases ENA-TCS-010 and ENA-TCS-011 and moves back to home net.	
	Step ENA-TCS-012-02	Firewall pinholes should be removed automatically after a specific timeout.	
Expected Results:	The firewall pinholes which are no longer needed should be deleted automatically at the MN-FW, HA-FW and the CN-FW after a specific timeout.		
Actual Results:			
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive		

Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No
Test Conducted by:	

5.1.4 Mobility Optimisation Test Cases

5.1.4.1 Enable Test Case Scenario 013: Mobility Optimised Handover

Table 5-14 ENA-TCS-013

	Enable Test Case Scenario 013		
Test Case ID:	ENA-TCS-013		
Test Case Name:	Mobility Optimised Handover		
Test Case Description:	MN starts a network initiated handover for AP1 to AP2 (figure 3.3)		
Work Package Reference:	WP1, WP2, WP6		
		Software Modules	
Components under test:	MN	MIPL	GSABA
	FMIPv6-AR	FMIPv6-MN	
Testbed Nodes involved:	AP1	IPv6 only network	
	AP2	IPv6 only network	
Protocols/lfs involved:			

Steps Taken:	Step ENA-TCS-013-01	MN request service authorisation
	Step ENA-TCS-013-02	Force MN to do handover (network initiated)
	Step ENA-TCS-013-03	Behaviour and performance
Expected Results:	A smooth handover should be observed	
Actual Results:		
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive	
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low	
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No	
Test Conducted by:		

5.1.4.2 Enable Test Case Scenario 014: Streaming with Mobility Optimised Handover

Table 5-15 ENA-TCS-014

	Enable Test Case Scenario 014
Test Case ID:	ENA-TCS-014
Test Case Name:	Streaming Video with Mobility Optimised Handover
Test Case Description:	MN should start streaming video and then it should initiate a handover

Work Package Reference:	WP1, WP2, WP4, WP6		
		Software Modules	
Components under test:	MN	MIPL	Video Client App
	FMIPv6-AR	FMIPv6-MN	
Testbed Nodes involved:	Multimedia Server	Video Streamer	
	AP1	IPv6 only network	
	AP2	IPv6 only network	
Protocols/lfs involved:			
Steps Taken:	Step ENA-TCS-014-01	MN should start video streaming and then perform handover	
	Step ENA-TCS-014-02	MN moves into range of AN4 & out of range of AN5	
	Step ENA-TCS-014-03	MN sends BU to HA1. MN receives BA from HA1	
Expected Results:	There should a minor disruption in video streaming as the codes are not properly optimised but it should be able to resume.		
Actual Results:			
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive		
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low		
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No		
Test Conducted by:			

5.1.5 Test Case Evaluation

This section shall address the issue of how to determine when Black Box testing has been completed and to assigning a verdict to the execution of a test case. One of the primary criterions in this step is that all planned test cases have been executed.

Once a test case has been exercised the easiest exit criterion to evaluate is the fact that the test case has been successfully executed and the verdict is that it has passed. However when the verdict is that a test case has been executed, but unsuccessfully, a number of factors have to be considered in order to fully evaluate the test.

Three types of verdict will be used to qualify the results of a test case execution: pass, fail and inconclusive.

- Pass means that the test case results prove that the observed test outcome gives evidence of conformance to the ENABLE usage scenario.
- Fail means that the observed test outcome either demonstrates non-conformance to the ENABLE architecture or contains at least one invalid test event (according to the test specification).
- Inconclusive means that an event, not related to the component under test, did not occur or lead to a wrong result. In this case, the test case result has no significance and can be discarded.

In addition to the verdict described above the test cases will also be assigned a priority on the level of attention a particular test case result will require. This section of the test case will only be completed if the test case status has been set to failed. There are four priorities an ENABLE test case may have.

- Immediate: This test case needs to have immediate attention as it's a vital part of a components function and may affect the Enable test bed or individual components.
- High: High means the test case that failed will need priority over other cases that are Medium and low. It is however, a critical flaw that will need attention.
- Medium: Medium priorities are test cases that have the same critical flaw as a high priority test case but just fall under a lower priority.

- Low: Low priority is where it may not affect the test bed or components but may be fixed due to cosmetic or non critical reasons. Low priorities will always be treated last if there are a number of failed test cases.

A severe defect will override any priorities due to a major flaw in an enable test bed component or modules. Server defects are uncommon in the test procedure and if found should be given urgent attention.

A test result template, described in Appendix C, will be used to document test case execution results.

5.2 S/W Development Management

As described in [ENA-D6.1] the ENABLE software development teams continued to use Subversion as the version controlled archive for the source code and TRAC for bugtracking.

The subversion server hosted on [<http://repos.ist-enable.org/repos>] was used to store all versions (stable and unstable) of the software modules developed in ENABLE, pre-compiled binaries of associated libraries and configuration files for the test bed components. The subversion repository structure was divided up as follows:

- <http://repos.ist-enable.org/repos/binaries/>
 - pre-compiled binaries of kernels, associated libraries and applications to the ENABLE code base
- <http://repos.ist-enable.org/repos/coderepos/>
 - Here the live, ever changing source code, build files for ENABLE developments were stored
- <http://repos.ist-enable.org/repos/configurations/>
 - Here all stable configuration and scripts files for the test bed set up were stored in this folder

5.3 Test Reporting/Debugging Tools Description

As unit testing was carried by the individual software development teams, there were a varying number of debugging tools used within the project, however there were three common tools used which are worth noting, the standard GNU debugger (gdb), valgrind and callgrind.

The GNU debugger (gdb) allows a developer to view what is going on `inside' their program while it executes on the Linux OS.

The gdb can carry out four main operations:

- Start a program, specifying anything that might affect its behaviour.
- Make a program stop on specified conditions.
- Examine what has happened, when the program has stopped.
- Change things in a program, so the developer can experiment with correcting the effects of one bug and go on to learn about another.

In ENABLE at times all four operations were implemented.

The Valgrind programme provides a suite of debugging and profiling tools, however on ENABLE it was primarily used as a memory checking tool to detect many common memory errors such as:

- Identifying improper uses of memory such as the overrunning of the heap block boundaries, or the reading/writing to freed memory.
- Using values before they have been initialised.
- Incorrect freeing of memory, such as double-freeing of heap blocks.
- Memory leaks.

Finally Callgrind is a programme that allows a developer to run an application under supervision to generate profiling data. This data is used to measure how much CPU it being consumed by an application and to help optimise the functions and hence the speed of an application.

During the course of this type of testing the Trac “defect tracking” system installed during year one of the project was used.

<http://repos.ist-enable.org/>

Trac allowed the developers to gain quick access to the code repository (Repository Browse) and to keep track of current faults on their prototypes (Ticket System) while providing testers with a co-ordinated approach to reporting faults.

6. COLLABORATION WITH IST PROJECTS (TI)

Throughout the entire duration of the ENABLE project possible collaborations with other IST Projects have been sought and evaluated. It was decided that the most valuable opportunity was an agreement with Anemone Project [ANEMONE] since its primary goal is to provide a playground test-bed supporting mobile users, devices and enhanced services by integrating cutting edge IPv6 mobility and multihoming initiatives. An official agreement between ENABLE and ANEMONE has been signed to describe the technical content of the collaboration and a few basic rules to regulate it.

The key technical points of the agreement were:

- to test and evaluate the ENABLE software and its interoperability in the ANEMONE test-bed and to provide the ENABLE partners with a detailed test report thereof;
- to provide the ENABLE partners with a reproducible copy (in source code form) of any modification or derivative work of the ENABLE software developed by ANEMONE partners and to license the same, on a non-exclusive, free of charge basis.

The basic idea of the agreement was to provide ANEMONE partners with a copy of binaries and source code of selected module of the ENABLE architecture on a non-exclusive, free of charge basis subject to the Open Source Licenses of the original packages. The components that have been selected for the collaboration were those involved in the split scenario since they make up the core of the ENABLE bootstrapping architecture.

Another expected outcome of the collaboration is the validation of the ENABLE software on additional platforms, such as handheld devices.

7. CONCLUSION

The scope of this document was to provide a description of the prototype software developments undertaken within the ENABLE project and then to proceed with a description of the network integration and trials of these prototypes and finally to highlight the mechanisms towards the validation and verification of the new ENABLE MIPv6 service environment.

In Section 2 the document picks up from [ENA-D6.1] and takes the initial prototypes described in [ENA-D6.1] and updates the software design and implementation as seen after year two of the project.

In some cases, such as the EAP-based MIPv6 bootstrapping, AAA for MIPv6 bootstrapping, DSMIP interworking with IPv4 networks, and HA load sharing, only the differences in the design are highlighted in this deliverable. In others cases such as for the “DHCPv6 extensions on access router” and “Softwires as tunnelling solution for IPv4 interworking” whole new designs are described. All but Softwires of the developed components from WP1, WP2 and WP3 have integrated seamlessly, and are called the “*Integrated Software of ENABLE*” throughout this document. The reason behind not integrating the Softwire components is that though it solves the interworking issue, it doesn’t currently support handovers and thus it can’t be considered a complete mobility solution.

The other two ENABLE prototypes components, the MIPv6 firewall traversal and Fast Mobile IPv6 (FMIPv6), were developed as separate mobility extensions and are also described in Section 2. There are some significant updates to these components as compared to their descriptions in [ENA-D6.1].

The document moves on to Section 3 which describes the research trial infrastructure for the integrated software components which was hosted at TI, the MIPv6 firewall traversal software components hosted at UGOE and Fast Mobile IPv6 (FMIPv6) software components hosted in Brunel. These reference trial infrastructures were initially used to check the compatibility and the functionality of the software modules being created by ENABLE. However once the software was stable, the trial infrastructures were amalgamated into one test bed which was then used for component integration and then the final demonstration, all hosted by TI.

Section 4 presents the realisation of the ENABLE application scenario, which from [ENA-D6.1] was the Scene 3 & Scene 6 of the search & rescue management scenario. It shows the association between the search & rescue scenes and its instantiation as a demonstration in the ENABLE MIPv6 service environment.

Section 5 captures the methodology used to validate and verify the ENABLE prototype developments. It also outlines the system tests carried out in the component integration and validation phase.

In conclusion this document is a comprehensive summary of software development carried out on specific functional components from the ENABLE work packages 1 to 4, the integration of these elements, entities and components, the conformance testing of these components to the ENABLE architecture and finally the demonstration of an efficient and operational mobility service in large heterogeneous IP networks.

8. REFERENCES

- [DIPv4] B. Aboba, J. Carlson, S. Cheshire: Detecting Network Attachment in IPv4 (DIPv4), IETF RFC4436, March 2006
- [draft-ietf-mip6-hiopt-02.txt] Hee Jin Jang, Alper Yegin, Kuntal Chowdhury, JinHyeock Choi: “DHCP Option for Home Information Discovery in MIPv6”, draft-ietf-mip6-hiopt-02 (work in progress), February 2007
- [draft-ietf-nsis-nsip-natfw] M. Stiemerling, H. Tschofenig, C. Aoun, E. Davies “NAT/Firewall NSIS Signaling Layer Protocol (NSIP), draft-ietf-mip6-hiopt-02.txt (work in progress), July 2007
- [draft-ietf-nsis-ntlp] H. Schulzrinne, R. Hancock „GIMPS: General Internet Messaging Protocol for Signaling”, draft-ietf-nsis-ntlp (work in progress), July 2007
- [draft-ietf-nsis-ntlp-statemachine] T. Tsenov, H. Tschofenig, X. Fu, C. Aoun, E. Davies “GIST State Machine”, draft-ietf-nsis-ntlp-statemachine (work in progress), July 2007
- [draft-werner-nsis-natfw-nsip- statemachine] C. Werner, N. Steinleitner, X. Fu, H. Tschofenig, C. Aoun NAT/FW NSIP State Machine“ draft-werner-nsis- natfw-nsip- statemachine-06.txt (work in progress), November 2007
- [draft-ietf-softwire-hs-Softwires-l2tpv2] B. Storer, C. Pignataro, M. Dos Santos, B. Stevant, J. Tremblay: “Hub & Spoke Deployment Framework with L2TPv2”, draft-ietf-softwire-hs-framework-l2tpv2-07, (work in progress) September 2007
- [EAP-AKA] J. Arkko, H. Haverinen, “Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA)”, IETF RFC 4187, January 2006
- [ENA-D1.1] Project IST-ENABLE Deliverable D1.1 “Requirements, scenarios and initial architecture”, June 2006.
- [ENA-D1.2] Project IST-ENABLE Deliverable D1.2 “Solutions for Mobile IPv6 bootstrapping and load sharing across Home Agents”, December 2006.
- [ENA-D2.1.2] Project IST-ENABLE Deliverable D2.1.2 “Final results in middlebox traversal”, June 2007
- [ENA-D2.2] Project IST-ENABLE Deliverable D2.2 “Solutions for IPv4 interworking with Mobile IPv6”, March 2007

- [ENA-D4.2] Project IST-ENABLE Deliverable D4.2 “Service authorization and control for QoS and multi-homing”, October 2007
- [ENA-D6.1] Project IST-ENABLE Deliverable D6.1 “Report on case studies and initial prototypes”, December 2006
- [FMIPv6] www.fmipv6.org
- [FreeRADIUS] Free Radius, www.freeradius.org
- [MIPL] MIPL Mobile IPv6 for Linux, <http://www.mobile-ipv6.org/>
- [NETFILTER] Netfilter, <http://www.netfilter.org>
- [NSIS_UGOE] <http://user.informatik.uni-goettingen.de/~nsis/download.html>, “Next Steps in Signaling (NSIS) Implementation by University of Goettingen”, (work in progress), June 2006
- [OpenWRT] Open WRT, <http://openwrt.org/>
- [PEAPv2] A. Palekar et al., "Protected EAP Protocol (PEAP) Version 2", draft-josefsson-pppext-eap-tls-eap-10 (work in progress), October 2004.
- [RFC2131] R. Droms: Dynamic Host Configuration Protocol, IETF RFC2131, March 1997
- [RFC2138] C. Rigney, A. Rubens, W. Simpson, S. Willens: Remote Authentication Dial In User Service (RADIUS), IETF RFC2138, April 1997.
- [RFC2461] T. Narten, E. Nordmark, W. Simpson: Neighbor Discovery for IP Version 6 (IPv6), IETF RFC2461, December 1998
- [RFC2462] S. Thomson, T. Narten: IPv6 Stateless Address Autoconfiguration, IETF RFC2462, December 1998
- [RFC2472] D. Haskin, E. Allen: IP Version 6 over PPP, IETF RFC2472, December 1998
- [RFC2661] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, B. Palter: Layer Two Tunneling Protocol "L2TP", IETF RFC2661, August 1999
- [RFC3775] D. Johnson, C. Perkins, J. Arkko: Mobility Support in IPv6, IETF RFC3775, June 2004
- [RFC3931] J. Lau, M. Townsley, I. Goyret: Layer Two Tunneling Protocol - Version 3 (L2TPv3), IETF RFC3931, March 2005
- [RFC4068] R.Koodli, Ed.: Fast Handovers for Mobile IPv6, IETF RFC4068, July 2005

- [RFC4285] A. Patel, M. Khalil, H. Akhtar, Authentication Protocol for Mobile IPv6, IETF RFC4285, January 2006
- [RUBY] Ruby on Rails, <http://www.rubyonrails.org/>
- [USAGI] UniverSAI playGround for Ipv6, <http://www.linux-ipv6.org/>
- [WIDE] Wide Project, <http://www.wide.ad.jp>
- [WIDE-DHCPV6] Wide DHCPv6 <http://sourceforge.net/projects/wide-dhcpv6>
- [XORP] XORP Router, www.xorp.org
- [XSUPP] Xsupplicant v 1.0, <http://open1x.sourceforge.net/>
- [ANEMONE] http://www.ist-anemone.eu/index.php/Home_Page

APPENDIX A.

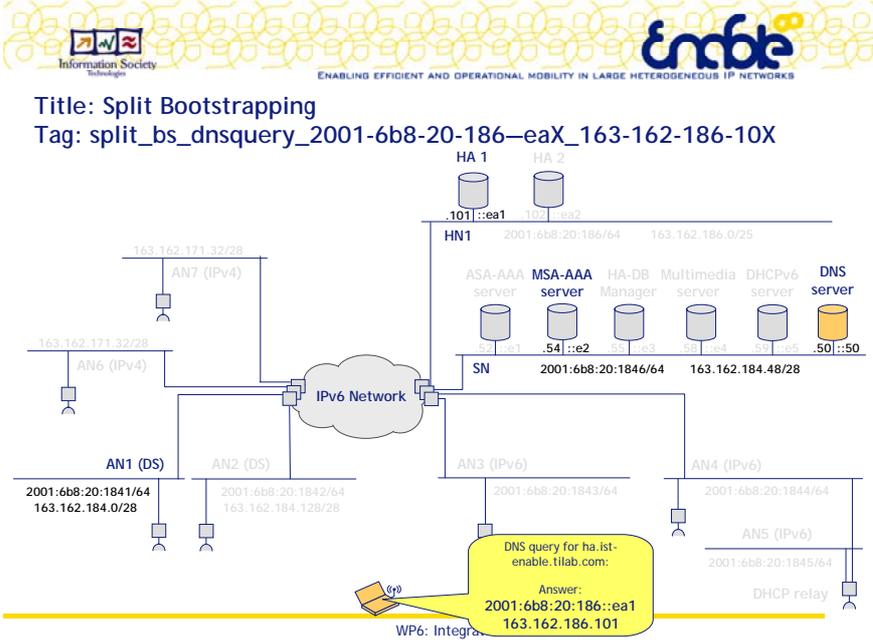


Figure 8-1 split_bs_dnsquery_2001-6b8-20-186—eaX_163-162-186-10X

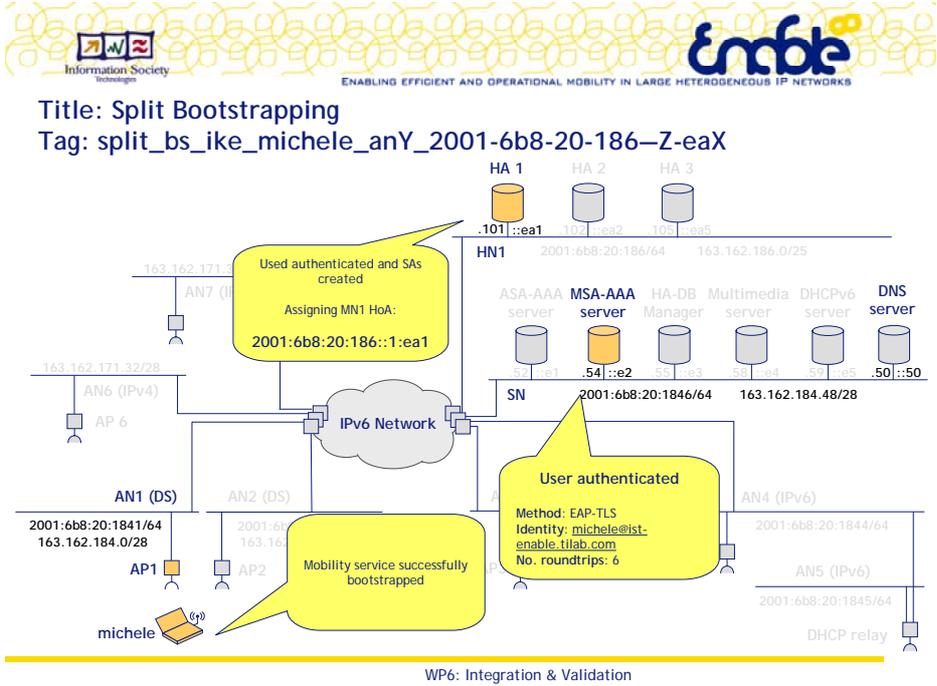
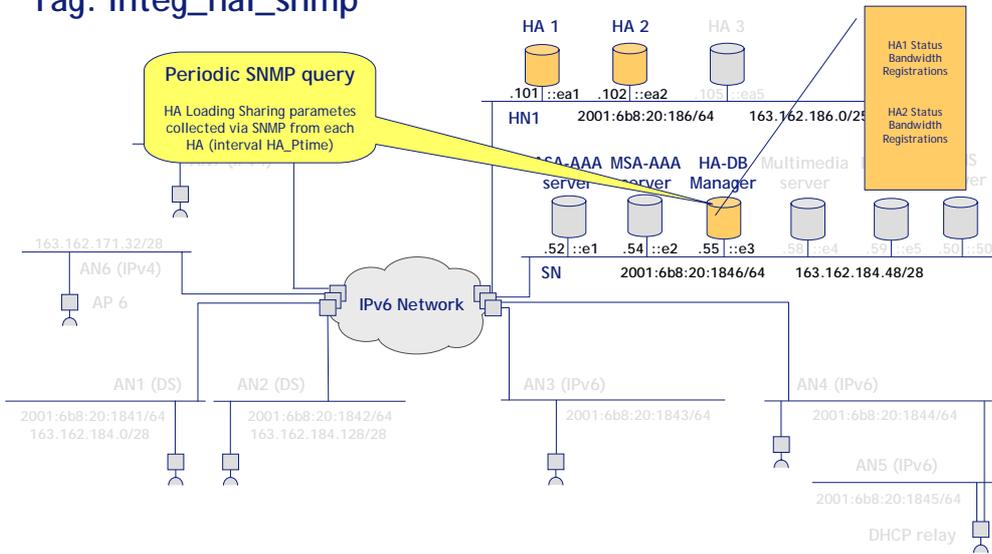


Figure 8-2 split_bs_ike_michele_anY_2001-6b8-20-186—Z-eaX



Title: HA Load Sharing
Tag: integ_hal_snmp

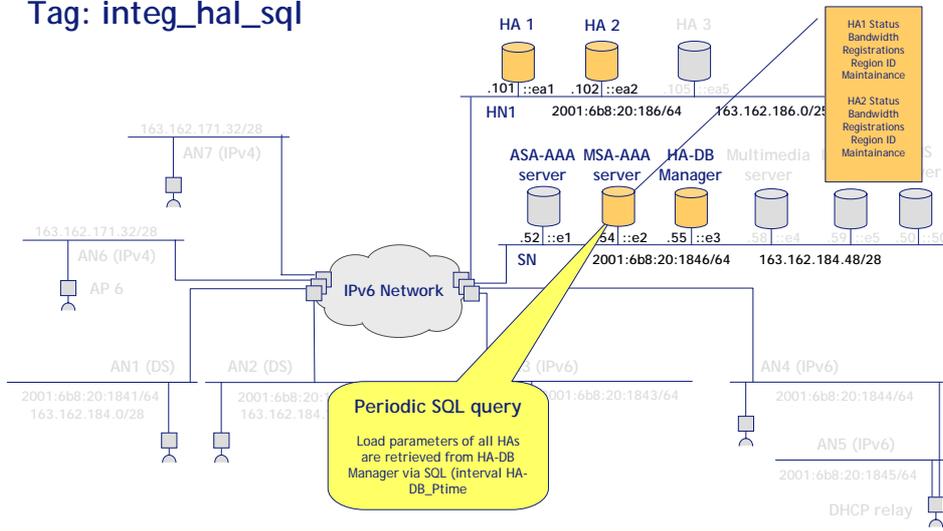


WP6: Integration & Validation

Figure 8-3 integ_hal_snmp



Title: HA Load Sharing
Tag: integ_hal_sql



WP6: Integration & Validation

Figure 8-4 integ_ha_sql

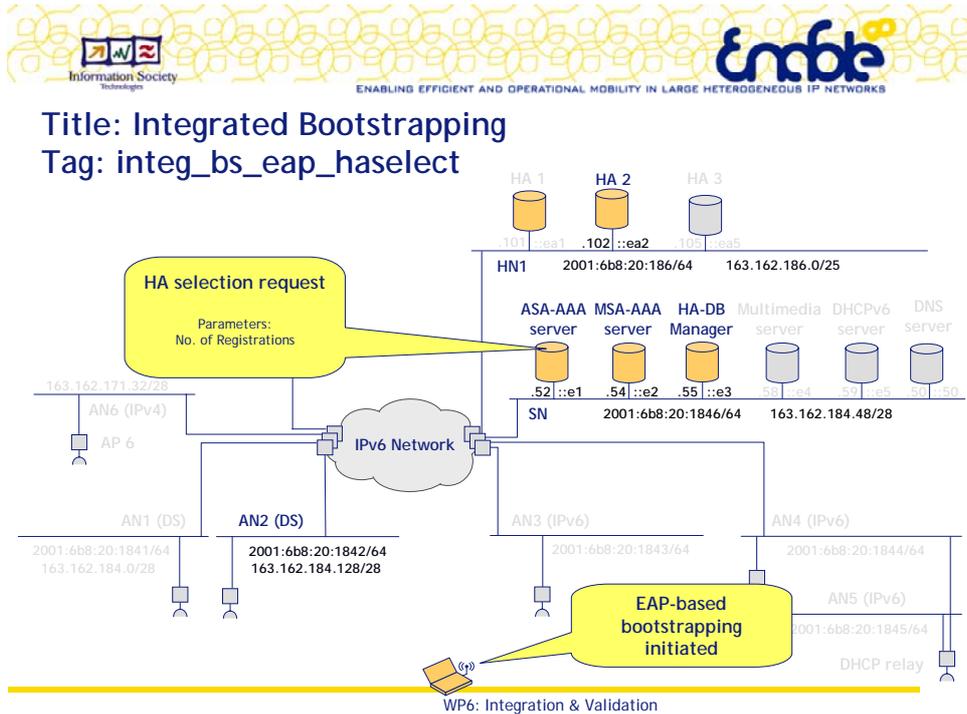


Figure 8-5 integ_bs_eap_haselect

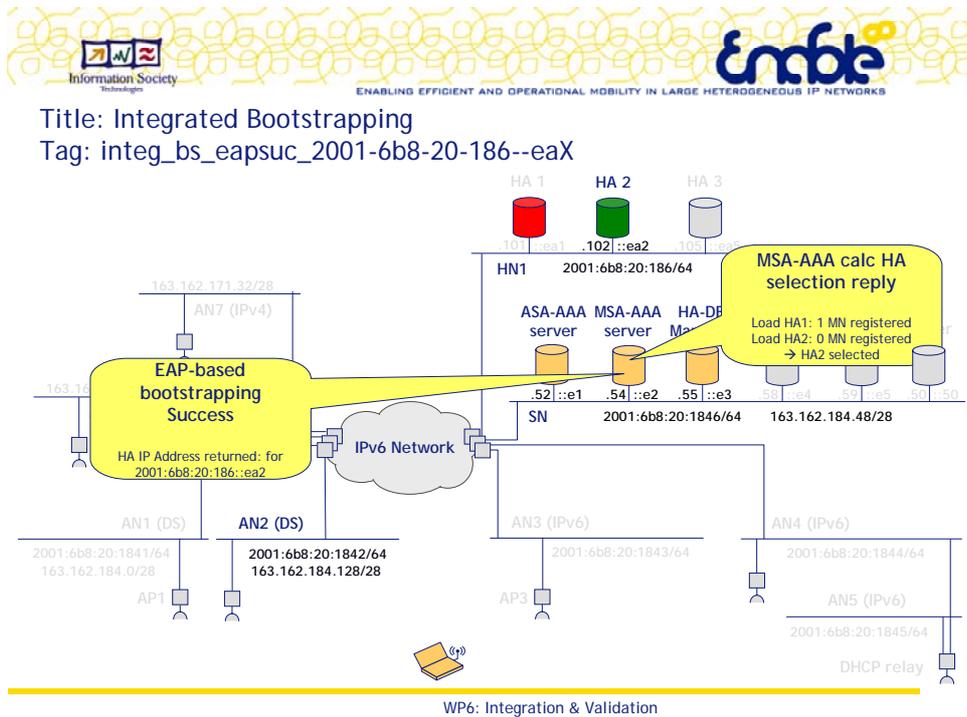


Figure 8-6 integ_bs_eapsuc_2001-6b8-20-186--eaX

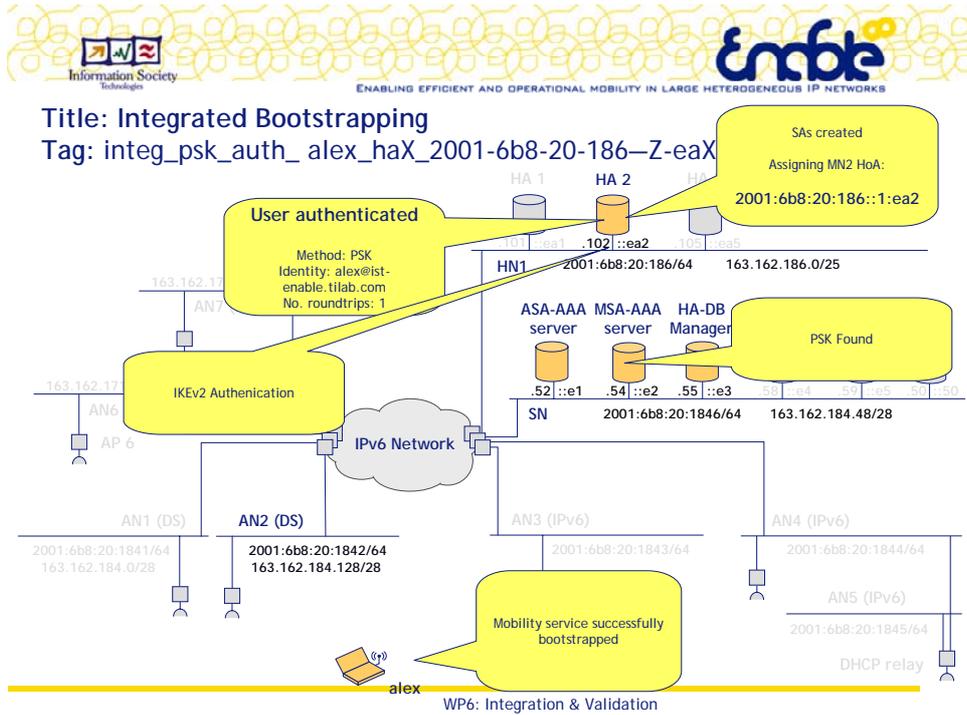


Figure 8-7 integ_psk_auth_alex_haX_2001-6b8-20-186-Z-eaX

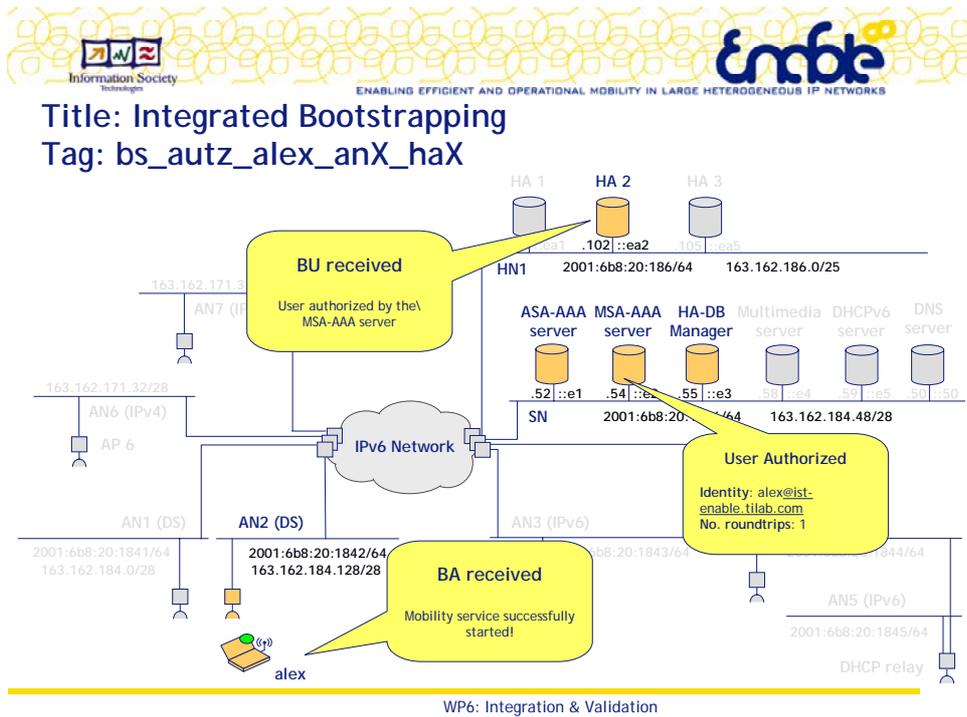


Figure 8-8 bs_autz_alex_anX_haX

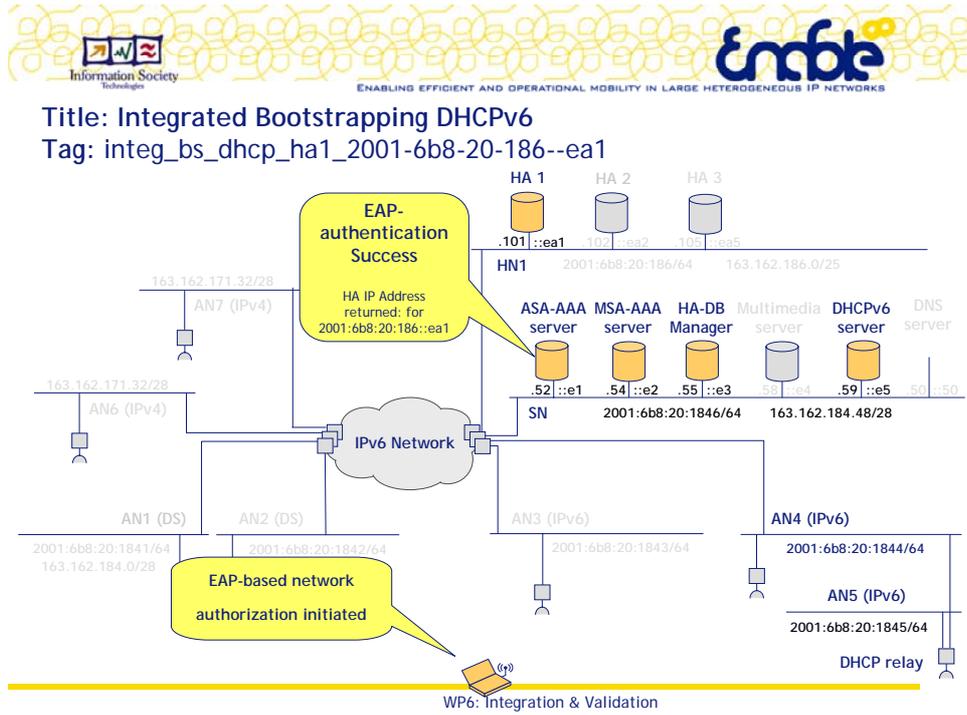


Figure 8-9 integ_bs_dhcp_ha1_2001-6b8-20-186--ea1

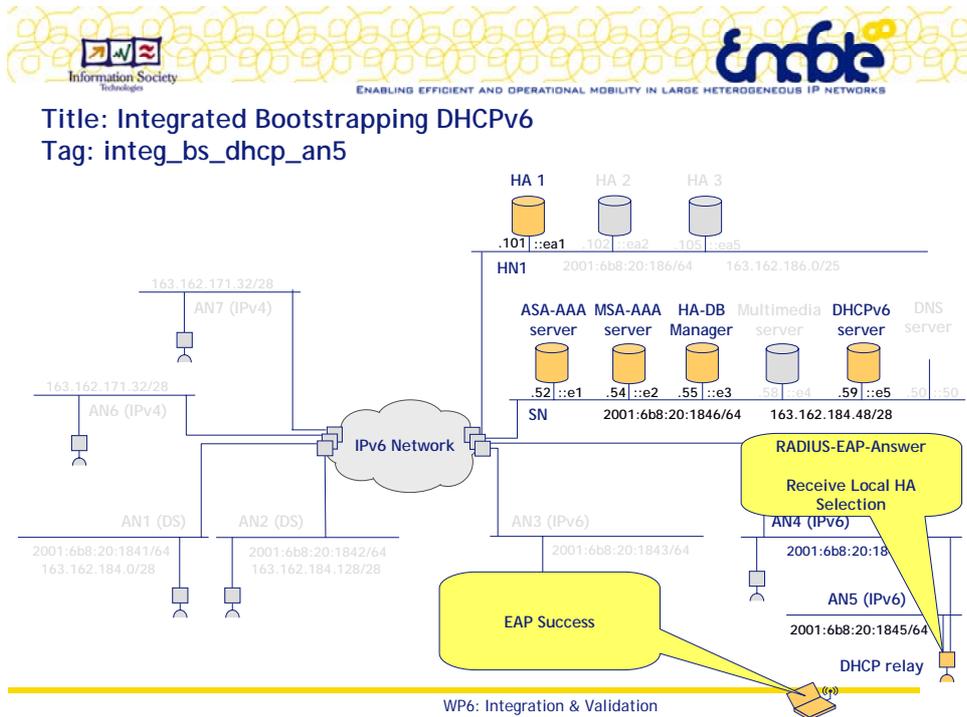


Figure 8-10 integ_bs_dhcp_an5

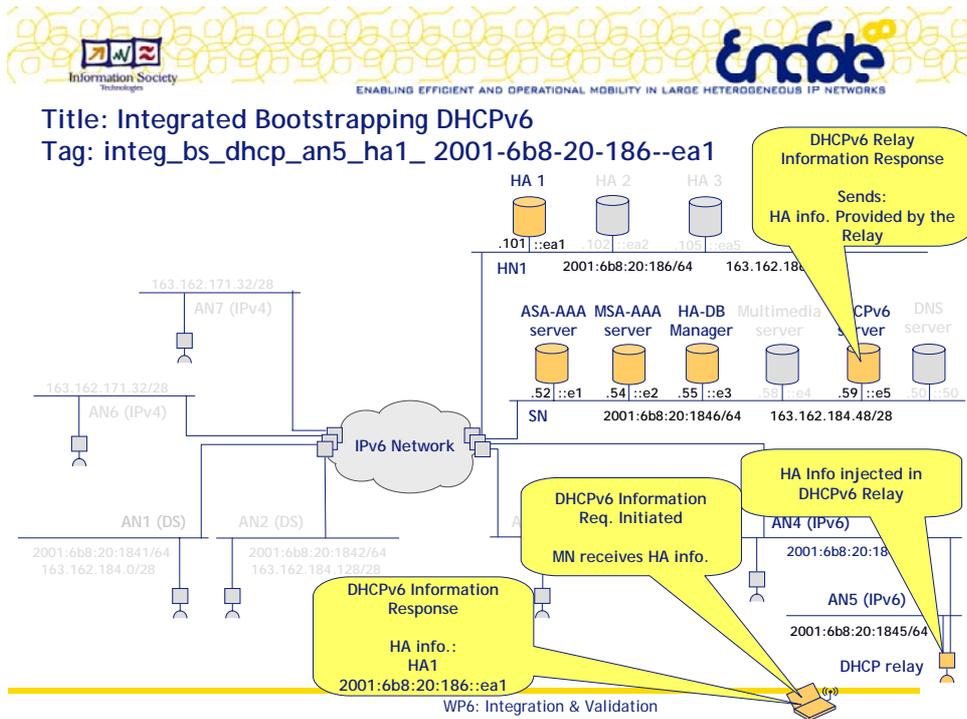


Figure 8-11 integ_bs_dhcp_an5_ha1_2001-6b8-20-186--ea1

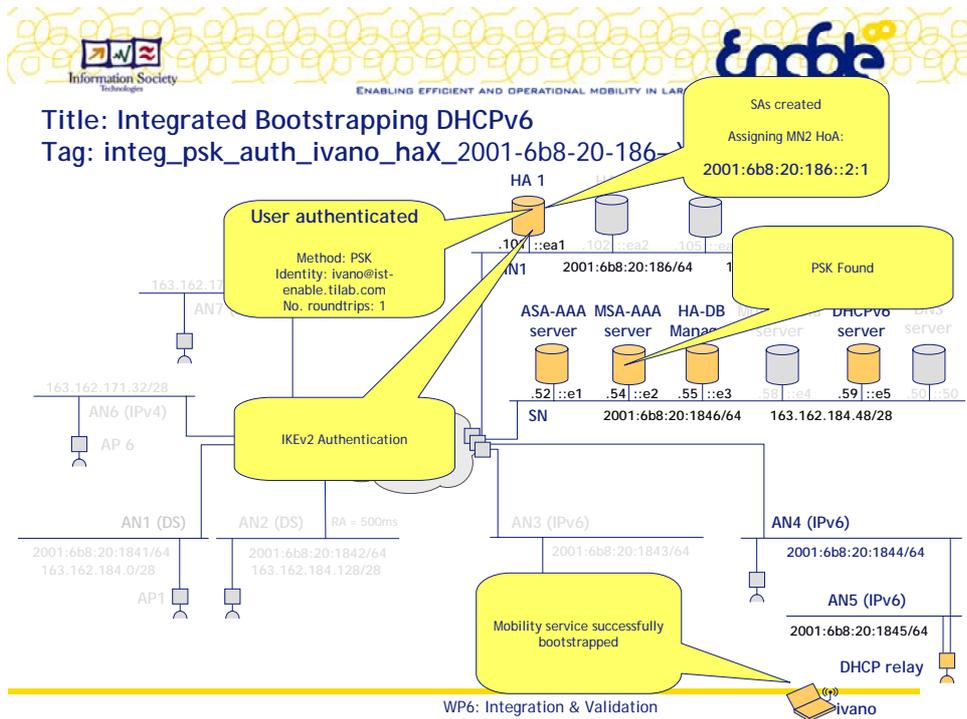


Figure 8-12 integ_psk_auth_ivano_haX_2001-6b8-20-186—Y-X

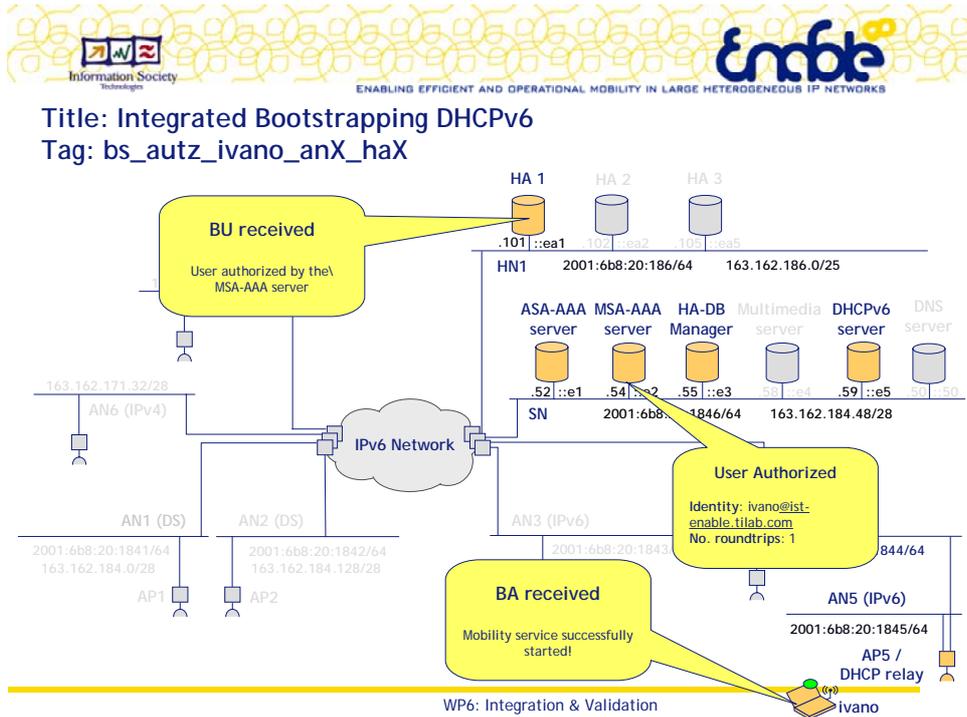


Figure 8-13 bs_autz_ivano_anX_haX

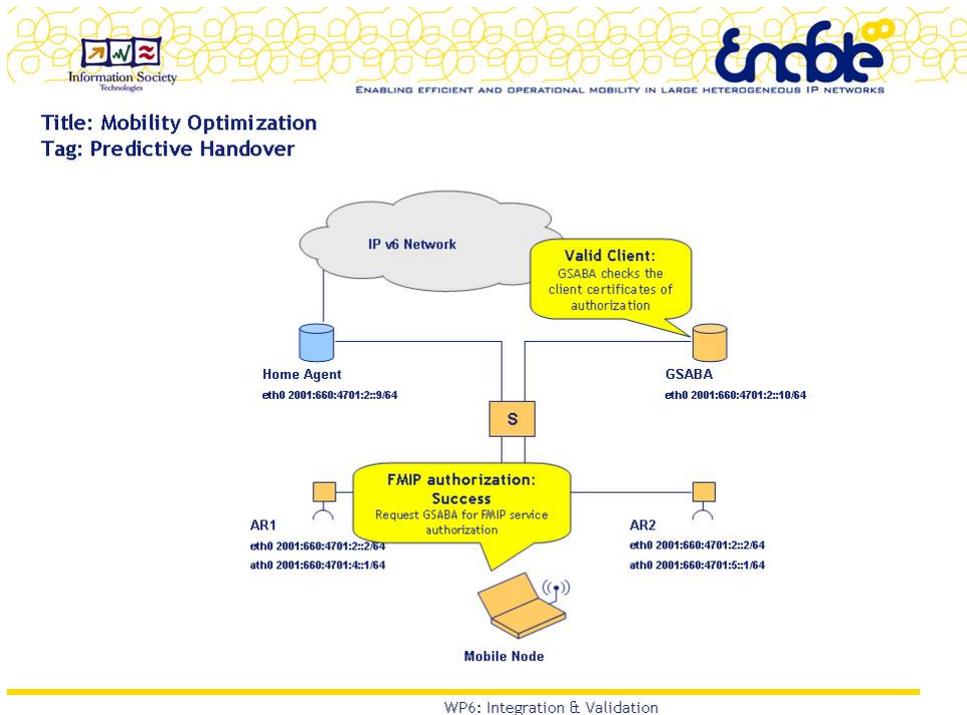
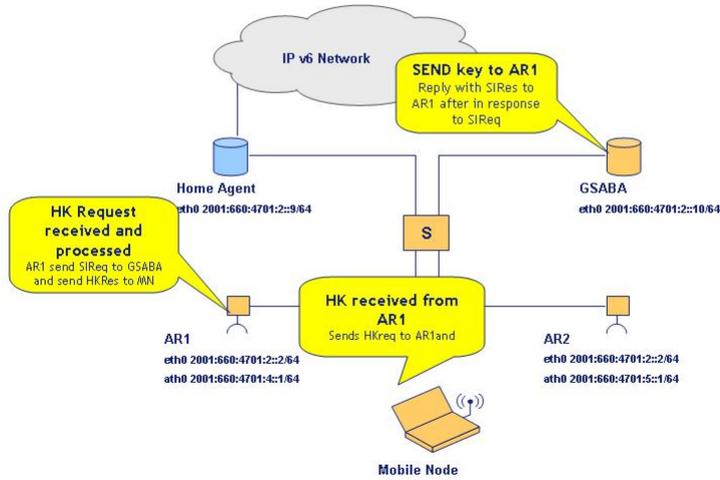


Figure 8-14 Predictive Handover



Title: Mobility Optimization
Tag: Predictive Handover

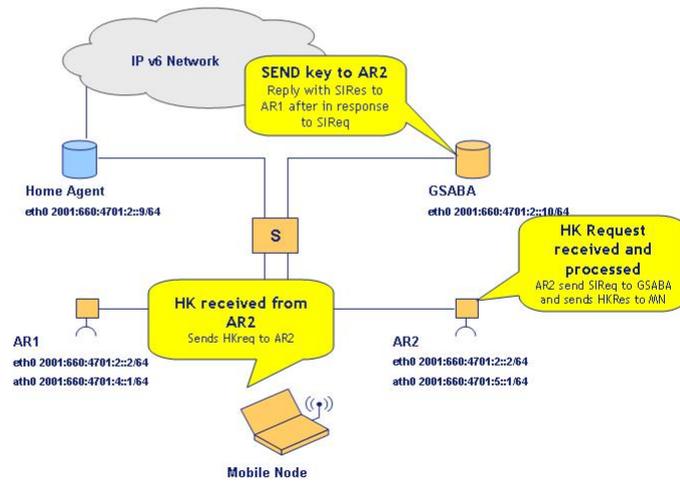


WP6: Integration & Validation

Figure 8-15 Predictive Handover



Title: Mobility Optimization
Tag: Predictive Handover

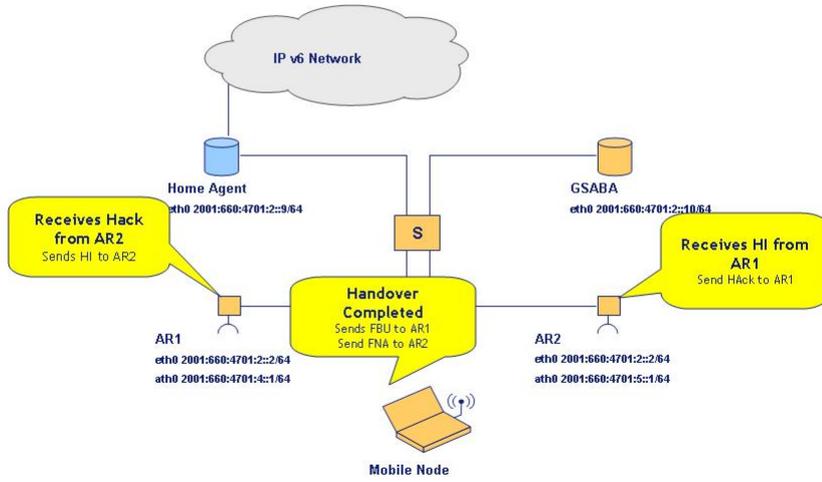


WP6: Integration & Validation

Figure 8-16 Predictive Handover



Title: Mobility Optimization
Tag: Predictive Handover



WP6: Integration & Validation

Figure 8-17 Predictive Handover

APPENDIX B.

	Enable Test Case Scenario 000		
Test Case ID:	ENA-TCS-000		
Test Case Name:			
Test Case Description:			
Work Package Reference:			
Components under test:			
Testbed Nodes involved:			
Protocols/lfs involved:			
Steps Taken:	Step ENA-TCS-000-01		
	Step ENA-TCS-000-02		
	Step ENA-TCS-000-03		

Expected Results:	
Actual Results:	
Test Case Status:	<input type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Inconclusive
Priority:	<input type="checkbox"/> Immediate <input type="checkbox"/> High <input type="checkbox"/> Medium <input type="checkbox"/> Low
Severe Defect:	<input type="checkbox"/> Yes <input type="checkbox"/> No
Test Conducted by:	

APPENDIX C.

 Test Case Scenarios Summary Sheet					
Case Test ID	Test Case Name:	Priority:	Severe Defect:	Results	Comment
ENA-TCS-001	Split bootstrapping				
ENA-TCS-002	HA Load Sharing				
ENA-TCS-003	Integrated Bootstrapping EAP				
ENA-TCS-004	Integrated Bootstrapping DHCPv6				
ENA-TCS-005	Handover IPv4Interworking				
ENA-TCS-006	Split bootstrapping PDA				
ENA-TCS-007	Handover PDA				
ENA-TCS-008	VoIP Call with Streaming Audio				
ENA-TCS-009	Streaming Video with Handover				
ENA-TCS-010	Pinhole creation for BU/BA after handover				
ENA-TCS-011	Pinhole creation for BT/RO data traffic				
ENA-TCS-012	Pinhole deletion				
ENA-TCS-013	Mobility Optimized Handover				
ENA-TCS-014	Streaming with Mobility Optimized Handover				