MEDIEVAL
INFSO-ICT-258053

MEDIEVAL

Deliverable D6.1

Initial Testbed Status Report

Editor:	Michelle WETTERWALD, EURECOM
Deliverable nature:	Public
Due date:	June 30 , 2011
Delivery date:	June 30 , 2011
Version:	1.0
Total number of pages:	39
Reviewed by:	Carlos J. Bernardos (UC3M), Loris Marchetti (TIS)
Keywords:	MEDIEVAL, testbed demonstrator, prototyping

Abstract

The objective of this deliverable is to describe a preliminary version of the platform setup that will be used for the final demonstrator of the MEDIEVAL project. It includes an initial review of the software environment and tools that will be needed throughout its implementation phase by the project partners. In a second step, an inventory is made of the software components and simulation tools already available at the partner premises which will be later adapted to demonstrate the project results. Finally, a first version of the project development and integration methodology that will ensure a higher quality and easier integration of the project testbed is provided.

List of authors

Company	Author
EURECOM	Michelle Wetterwald, Christian Bonnet
ALBLF	Telemaco Melia
CFR	Marco Mezzavilla, Daniele Munaretto
DOCOMO	Gerald Kunzmann
IT	Daniel Corujo, Sérgio Figueiredo
LiveU	Noam Amram
PTIN	Nuno Carapeto
UC3M	María Isabel Sánchez Bueno, Carlos J. Bernardos, Antonio de la Oliva

History

Modified by	Date	Version	Comments
EURECOM	30/06/2011	1.0	Submission

Executive Summary

The purpose of this document is to describe and present the initial description of the MEDIEVAL test site. It is the initial testbed status report and provides an introduction to the implementation and deployment of MEDIEVAL framework. It also provides a first description of the platform setup including both hardware and software that will compose the testbed, their requirements, the major components that already exist and will be deployed there and enhanced by MEDIEVAL. This list is based on the resources committed to the project by the different partners.

The tasks developed on this work package are still in early stages as are the requirements and conclusions this document contains. It was, nonetheless the concern when preparing this document to push the agreement of partners involved in the various subsystems towards the definition of a work plan for proper deployment integration and test of the software components. Initially, by producing a process to guide the implementation on the platforms, environment and tool versions used by all partners in their own controlled laboratories. Then by accepting a set of software methodology rules and "good coding practices" detailed to facilitate integration and bugs fixing during integration and testing process, reducing their impact on project phases. Specifying the testbed plan at an early stage embodies the resolution to strengthen the implementation process and enable the delivery of high quality software components during the integration phase. The plan also includes the initial details for the deployment of CDN nodes and to evaluate the scalability of the MEDIEVAL project.

The document begins with a flexible list of the preliminary requirements expressed for the test site, including environment, network, hardware and accessibility requirements. It includes a testbed description listing the major hardware and software requirements for the test site extracted from the modules of the architecture which are planned to be developed, starting from existing implementations which will be extended and adapted, where necessary, to support MEDIEVAL specific functionalities . A description of these modules can be found in MEDIEVAL deliverable D1.1 [1]. Afterwards it presents the initial setting for the testbed describing some of the components and network nodes to be deployed by each work package on the test site.

The description continues with a presentation of the software components that are already available and will be reused for installation on the MEDIEVAL testbed. These are to serve as support for the MEDIEVAL architecture and to be improved within the different work packages.

- Flow Manager Software has been developed by ALBLF in the context of the IETF activities standardizing flow mobility extensions and is to be expected as the starting point to Flow Manager described in D4.1. At the moment it is mostly focused on classifying the packets traversing the LMA and for setting specific routing rules on the selected (according to operators' specific rules) IP flows.
- OpenAirInterface LTE Platform, implements the LTE protocols on the radio interface between a Mobile Node and a proprietary Base Station, using real-time extensions of Linux Operating System. It includes a middleware NAS driver, the RRC layer, the MAC layer, the RLC layer and PDPC frames support. It is completed by RF cards providing the transmission of the LTE radio signal.
- Open PMIP Protocol Stack, is a library of reusable components that compose an open-source implementation of PMIPv6 (RFC 5213) protocol with both LMA and MAG capabilities provided. When deployed it act as a router providing intra-domain mobility transparently to Mobile Nodes.
- OpenAirInterface PMIP Protocol Stack is a Linux Open Source implementation of PMIPv6 that reuses the MIPL software and implements both LMA and MAG components in the same code.
- Live Video Transmission Units, are a hardware and software solution that will be connected to cameras and to multiple clients simultaneously producing live content. It supports encoding profiles of H264 and will be improved to implement both AVC and SVC encoding types.
- ODTONE IEEE 802.21 MIH Protocol Stack is an open-source implementation of the Media Independent Handover framework from the IEEE 802.21 Media Independent Services standard. It revolves around the implementation of a MIHF function which provides the Media Independent Event Service, the Media Independent Command Service and the Media Independent Information Service along with supporting mechanisms such as Capability Discovery, MIHF Registration, Event Registration, among others.

- MRD6 Multicast Router Software, is an open-source implementation for Linux and BSD on IPv6, which supports the following main features: MLDv2, BSR and embedded-RP, MBGP (partial), native and virtual interfaces, remote configuration and automatic translation of IPv4 multicast traffic to IPv6.
- NEMO Basic Support Protocol stack is designed as an extension to MIPv6 which handles the mobility of a whole network as defined in RFC 3963 where mobility management is transparent to the nodes attached to the Mobile Router, enabling session continuity while the network moves.

Additional simulation tools and platforms will be used by the MEDIEVAL project, to investigate with more details some of the challenges for the LTE wireless components. It was initially developed is ns-3 and tackles the issue of multimedia traffic optimization in LTE networks. It targets both the analysis and design of novel communication techniques and the implementation and extension of the network simulation tool to test and validate the proposed approaches.

Lastly the document includes a preliminary version of the methodology for the development and integration inside the project which adopts an incremental approach, scheduling an iteration of design and prototyping phases, while maintaining an always-on testing platform starting from the end of the first year of the project. It plans a design phase, including architecture and initial interface specification for the first year; first implementation with selected integration and refinement of the specifications for the second year and revised implementation with full integration and final specification for the last year. This incremental approach will ensure the adequate quality of the produced systems that will be documented in the future deliverables D6.2 providing a final report of the MEDIEVAL testbed and D6.3 and D6.4 which will present the testing reports.

Table of Contents

List of Figures

List of tables

Abbreviations

ALTO	Application-Layer Traffic Optimization
AM	Acknowledged Mode
AR	Access Router
ARCEP	Autorité de Régulation des Communications Électroniques et des Postes (France)
AVC	Advanced Video Coding
BC	Binding Cache
BCE	Binding Cache Entry
BCH	Broadcast Channel
BLER	Block Error Rate
BSR	Bootstrap Router
BU	Binding Update
CCCH	Common Control Channel
CDN	Content Delivery Network
CoA	Care-of Address
COTS	Commercial of the Shelf
CQI	Channel Quality Indicator
DLSCH	Downlink Shared Channel
FM	Flow Manager
GSOC	Google Summer of Code
HA	Home Agent
HARQ	Hybrid Automatic Repeat request
HDMI	High Definition Multimedia Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
ISP	Internet Service Provider
L2S	Link to System
LLR	Log-Likelihood Ratio
LMA	Local Mobility Anchor
LTE	Long Term Evolution
MAC	Media Access Control
MAG	Mobile Access Gateway
MAR	Mobile Access Router
MBGP	Multiprotocol Extensions for BGP
MEDIEVAL	MultimEDia transport for mobIlE Video AppLications
MCS	Modulation and Coding Scheme
MICS	Media Independent Command Service

MIES	Media Independent Event Service
MIH	Media Independent Handovers
MIHF	Media Independent Handover Function
MIIS	Media Independent Information Service
MLDv2	Multicast Listener Discovery
MMIB	Mean Mutual Information per coded Bit
MN	Mobile Node
MNP	Mobile Network Prefix
MR	Mobile Router
MRD6	Multicast Routing Daemon for IPv6
NAS	Network Attached Storage
NEMO B.S.	Network Mobility Basic Support
ns-3	Network simulator 3
ODTONE	Open Dot Twenty One
OPMIP	Open Proxy Mobile IP
OTT	Over The Top
PBA	Proxy Binding Acknowledgement
PBU	Proxy Binding Update
PDCP	Packet Data Convergence Protocol
PMIPv6	Proxy Mobile IPv6
PoA	Point of Attachment
RA	Router Advertisement
RACH	Random Access CHannel
RDF	Resource Description Framework
RFC	Request for Comments
RLC	Radio Link Control
RNTI	Radio Network Temporary Identifier
RRC	Radio Resource Control
RTAI	Real-Time Application Interface
RTSP	Real-Time Streaming Protocol
SAP	Service Access Point
SDI	Serial Digital Interface
SI	System Information
SINR	Signal to Interference plus Noise Ratio
SVC	Scalable Video Codec
UDP	User Datagram Protocol
ULSCH	Uplink Shared Channel
UE	User Equipment
UM	Unacknowledged Mode

VSP	Video Service Portal
WLAN	Wireless Local Area Network

1 Introduction

The goal of the MEDIEVAL project is to provide efficient mobile system architecture to support video services. The proposed architecture is composed of four main subsystems: Video Services Control, Transport Optimization, Wireless Access and Mobility, as described in deliverable D1.1 [1]. In D1.1, each subsystem has been broken into components and modules, further mapped on a network topology made of various types of network nodes: servers, routers, mobile nodes equipped with wireless access modules, etc...

One of the key objectives defined in the MEDIEVAL work plan is to develop the technology designed that serves as a proof of concept of the project result as well as a basis for future commercial deployments. In order to validate the performance of the MEDIEVAL architecture, a demonstrator has to be designed over the lifetime of the project, taking into account the requirements of the main subsystems and also the implementation constraints of the various software elements.

The present document prepares the elaboration of this demonstrator and investigates several aspects related to the setup of the initial MEDIEVAL testbed. The WP6 work package started only at M8, this deliverable is thus a very preliminary result of its initial studies and coordinated activities.

The deliverable begins by proposing the environment that will be setup for the testbed. Since each software component may run on a different environment (in terms of Operating Systems, kernel versions, library versions …), a specific attention is paid to highlight the corresponding requirements in order to ease the final choice of a common platform.

The deliverable then continues with the inventory of various functionalities available at the partners' premises and made at the disposal of the project. These software components will be enhanced or directly reused to serve the MEDIEVAL subsystems designed in the different work packages of the project. They cover basic functions for the wireless access, mobility management, flow management, multicast routing, Media Independent Handover and video transmission A short description of each component is provided as well as Internet links from which they could be retrieved.

A simulation environment is also proposed as an additional tool to validate complementary aspects of the performances of the MEDIEVAL architecture, in particular within the LTE wireless access context.

Finally the important point of methodology is addressed. The testbed will be setup at a unique location and will be accessed by all the project members either locally or remotely for testing their own implementations. It will also be incrementally enhanced during the lifetime of the project. Good practice guidelines are devised to be able to assess the quality of the various developments before integration on the testbed and ensure its proper operation during the architecture validation tests.

The deliverable is organized as follows. Section 2 introduces a description of the test site environment. In section 3, the inventory of existing software components along with their requirements is provided. The simulation environment is described in section 4 and the preliminary integration and methodology guidelines are provided in section 5.

2 Testbed description

2.1 Testbed specification

This section lists the preliminary requirements which have been expressed on the future platform, including environment, network, hardware and accessibility requirements**.** Since they are made at an early stage of the project, they should be taken as still flexible enough to accommodate further requirements which may arise when the implementation work becomes more mature.

The objective addressed by specifying the testbed at an early stage is to strengthen the implementation process and enable the delivery of high quality software components during the integration phase. This objective can be achieved by combining two sets of actions. First, agreeing early in the implementation process on the platforms, environment and tool versions used by all the partners at their premises. Second, accepting a set of software methodology rules and "good coding practices" that will allow the discovery of issues and bugs early in the testing process, reducing their impact on the overall integration phase. This second set of actions is developed in section 5.

Main platform choice:

- Operating system for all nodes: Linux, under Ubuntu 10.04 distribution and 2.6.32 or newer kernel (Vanilla). GNU/Linux is a very powerful operating system which allows the development of network protocols and applications. Its network stack is very advanced supporting many leading edge protocols and it is actively supported. There are also many open source codes already available, in addition to what the MEDIEVAL partners provide which is widely implemented under that Operating system. In addition, there are no license problems, so MEDIEVAL development plans to be done in Linux. The distribution and kernel versions have been selected based on the constraint of the OpenAirInterface platform (see section 3.2) which has strong restrictions due to the usage of real-time extension, RTAI, presented below.
- C/C++ compiler : gcc v4.4.3. The same compiler (tool and version) should be employed by all MEDIEVAL partners involved in development in order to avoid problems during integration. This version has been chosen because it is the one provided with the Ubuntu distribution.
- WLAN cards: The hardware related to WLAN cards will be based on chipsets Atheros or Broadcom. The final decision on what specific chipset will be used depends on the functionality that can be modified on each driver.
- WLAN driver: In the case an Atheros chipset WLAN card is used, there are several options regarding the driver to use. The first option would be to use the MadWifi[1] driver , although new alternatives such as the use of ath5k[2]/ath9k[3] are also being considered. In case Broadcom based WLAN cards are used, the b43[4] driver is required.

Other external software, tools and utilities

- RTAI (RealTime Application Interface for Linux): version 3.8 or above. RTAI is a real time extension to Linux and is used in the OenAirInterface platform for the correct scheduling of the LTE Physical layer.
- Mobile IPv6: There are several options regarding the MIPv6 protocol support. One option would be to use the MIP extensions that are part of the kernel starting from v2.6.20. These extensions require the use of a user space agent, for that purpose the UMIP[5] software will be used. Other options include the reuse of PMIPv6 functionality (extracting MIPv6 functionality from MEDIEVAL PMIP implementation) or using ad-hoc implementation tailored to MEDIEVAL requirements.

[1] http://madwifi.org/
[2] http://linuxwireless.org/en/users/Drivers/ath5k
[3] http://linuxwireless.org/en/users/Drivers/ath9k
[4] http://linuxwireless.org/en/users/Drivers/b43
[5] http://www.umip.org/

- Boost C++ Libraries, v 1.42 or higher, (only the source code) with the Boost.Build configured and GCC C++ Toolset. These libraries and tools are required to run the ODTONE software (section 3.5).
- Network traffic analysis: Wireshark (ex-Ethereal). Wireshark is a very famous tool for inspecting data packets on network interfaces, which allows an easier debugging.
- Network simulator ns-3: ns-3 is the basis of the LTE simulator that will be used to evaluate some of the advanced algorithms implemented on the LTE wireless access (see section 4.1).
- An additional C++ custom simulator is used to investigate algorithms on video coding (SVC). It is available online at: http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm .

CDN Nodes

The software that will be used to demonstrate the CDN nodes has not yet been chosen. To prepare the CDN integration, Medieval has been evaluating software from a list of ten available open source CDN projects (http://www.fromdev.com/2011/06/create-cdn-content-delivery-network.html). The first step was to cut down those ten to the three CDN solutions that we think are best suited for integration into our testbed. The choice was on:

- ModCDN, a module for the Apache server,
- OpenCDN, although it seems there was not much activity on it since 2008,
- CoralCDN, which is quite complete and complex.

Now, the next step is to perform a more detailed analysis of the existing tools in order to find out which project best meets our expectations. Therefore, we defined a list of requirements on the CDN solution:

- Maturatiy: using a stable and advanced solution is important to have a good starting point.
- Modularity and cleanness of code: strongly required for understanding and modifying the code.
- Adaptability: integration into the testbed and adapting the protocol to the Medieval architecture should not be too difficult
- Possibility to set up our own CDN: we do not want to participate in an existing content delivery network, but set up our own CDN nodes, such that our results are not falsified by external influences.
- Multicast integration: it should be possible to use a multicast protocol with the CDN solution.
- Licencing issues: The licence (BSD, GNU, Apache, …) might impose some legal issues for some partners.

The plan is to download and install the above mentioned CDN solutions in order to find out the best suited solution for the MEDIEVAL testbed.

The usage of CDNs as part of the mobility process in MEDIEVAL is centred upon a simple interaction between the CDN Decision Module and the Flow Manager. On the testbed, the integration of the CDN nodes will be performed as shown in Figure 3. The CDN will be placed close to the MAR (which is in the P-GW of the mobile network architecture). CDN nodes are controlled by the CDN Node Control (CDNNC) inside the MAR.

The idea is to include the CDNs opinion on where they think the mobile node should anchor next, so to achieve the best possible performance in terms of content accessibility. This way we can assure that the mobile node anchors as close to the content cache as possible, reducing latency and the need to cache the content on yet another location.

In a testbed point of view, this means to have the CDN software and content cached on the virtualized nodes that correspond to the MARs, allowing us to analyse the impact of these integrated mobility decisions in the node's mobility, as well as to experiment in ways to optimize the node's mobility and the CDN's cache location.

Network accessibility

The testbed network is fully supporting IPv6 and all its specific features.

An secure remote access will allow the project partners to upload, compile and if possible test, their own software contribution. Internet will be used for the remote operations of the test site. The access will be provided via an ssh login on a dedicated host. This host will be a front end router in charge of authentication procedures and will be allocated a public address. For remote ping6 or traceroute between demo sites, it will also be allocated a public IPv6 address. When not operational on their wireless interface, the mobile node terminals will be attached via Ethernet to the remote network. In that case they will be remotely operated as the other nodes.

It will be possible to interconnect remote testbeds via a VPN software (OpenVPN) used in bridge mode, according to partners requirements.

A more precise description of the accessibility will be part of the first activities on the test site deployment.

Figure 1shows the remote setting operated for a former project at EURECOM.

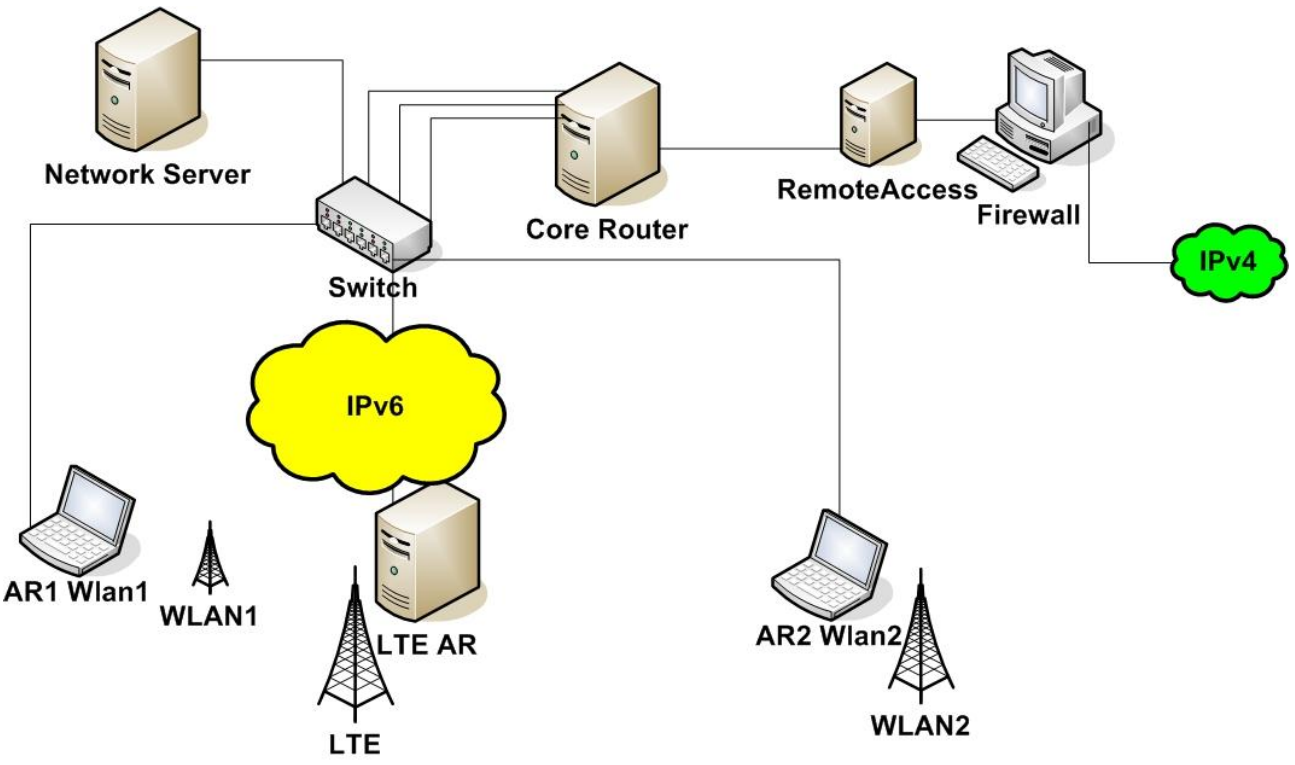

Figure 1: Remote operation of the test site

2.2 Test site description

The MEDIEVAL test site is located at EURECOM premises in Sophia Antipolis close to Nice (France).

Figure 2: Pictures of the EURECOM laboratory

As a research centre, EURECOM is connected to the Internet in both IPv4 and IPv6 through a 100Mbits link to RENATER (the French research network). The operational systems are using IPv4 while some research activities use IPv6. Computers connected to the LAN are protected by security tools: firewall, intrusion detection. The EURECOM security policy objective is to protect the LAN from Internet attack while supporting research activities by the use of several dedicated trusted zones. The Network hosts a large set of services (File servers, email servers, web and Intranet servers etc…) as well as client desktops and laptops using both wired and WLAN connections. The most vital elements of the infrastructure are clustered. The power supply is protected by inverters and generators, There have been no service interrupt in the last year. All the EURECOM infrastructures are operated by the members of the IT support service.

In the recent past EURECOM hosted demonstration and/or integration networks for various national and European projects including DAIDALOS I and DAIDALOS II, E2R, MULTINET. To facilitate the integration phase, secured remote access can be offered allowing remote operations. The site can support WLAN transmissions as well as LTE transmissions under the experimental frequency bands permit given by the French regulator ARCEP.

2.3 Testbed initial setting

According to the architecture and design work performed in the other work packages, especially in WP1 [1], we already have a preliminary idea of the MEDIEVAL testbed initial setting. This setting will evolve in the second year to accommodate all the requirements from the early implementation.

The nodes required from each WP are the following:

ALL :

- Mobile Nodes: At least 3 to demonstrate the PBS use case with one producer and two listeners

WP2 (Video Services Control):

- Video Service Portal
- Provisioning Platform
- Session Management
- ISP video server

WP3 (Wireless Access) :

- WLAN PoAs
- LTE PoAs

WP4 (Mobility):

- At least 3 MARs
- MIIS server
- at least 2 LTE PoAs and 1 WLAN PoA

WP5 (Transport Optimization):

- Core Router
- ALTO server
- at least 3 MARs
- at least 3 CDN nodes (more can be included later to better demonstrate de P2P CDN scenarios)
- at least 2 LTE PoA and 1 WLAN PoA

Figure 3 shows a picture of the planned initial setting, based on the physical placement of the MEDIEVAL sub-systems and components described in D1.1.

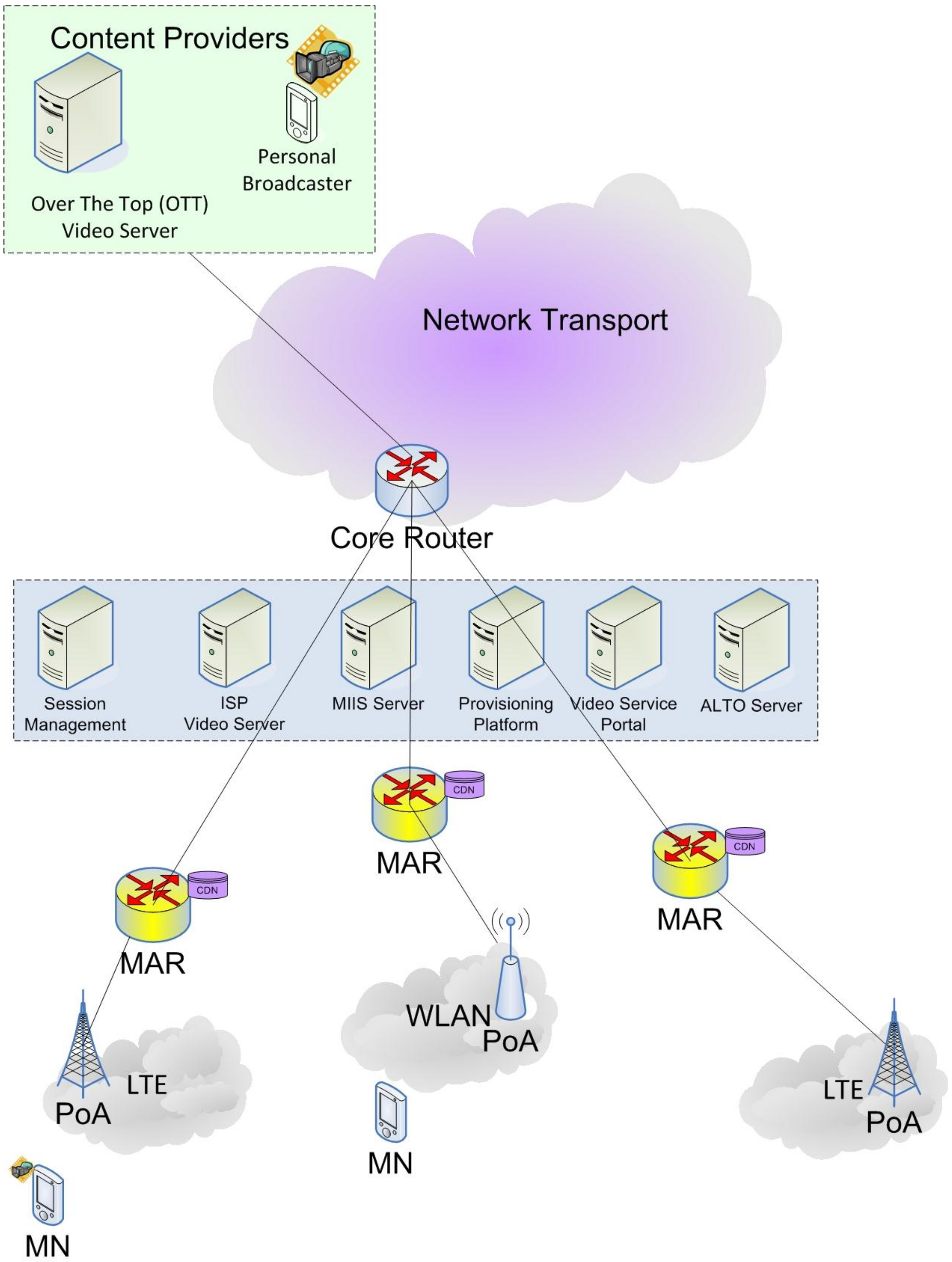

Figure 3: Initial setting for the MEDIEVAL testbed

2.4 Scalability Considerations

Building a large testbed of physical machines for the purpose of testing the scalability of some scenarios has always been very expensive, both from the hardware perspective but also from the manpower required for installation configuration and troubleshooting. Now the usage of virtualized machines has become a popular means to inexpensively create and replicate extra machines on a testbed. We intend to build our software with this spirit in mind so that is possible to virtualize whole networks. In this way, a single machine with powerful hardware can support several virtualized computer networks.

We intend to use the free tool named “Netkit”, in order to provide the basis of the environment. The tool allows us to create virtualized networks of computers and simplifies the management of the network. This in conjunction with wireless propagations models for LTE and WiFi (to simulate the wireless access media) can provide a good base on which to build a rough and simple testbed to evaluate some principles of the MEDIEVAL architecture and components.

We currently have a machine reserved for this purpose, that has an Intel Xeon Quad-core CPU (E5410 @ 2.33Ghz), 8 GB of RAM and a 250GB HDD.

Scalability Evaluation of the Testbed

By deploying this testbed we have a twofold advantage: we can run any software in the virtual machines as if they were real, and, thus we can study the scalability of the testbed by the amount of resources consumed by the virtualized networks and its virtual machines.

Scalability Evaluation of DMM

Setting up a simple DMM scenario for demonstration usually requires a small amount of machines (some MARs plus at least a mobile node). With our virtualized testbed, we have an opportunity to run several DMM scenarios, in a simple and clean way. This comes, without the need of deploying and setting up of real machines, and thus allowing us to have larger and more complex networks with several mobile nodes, which in turn will allow us to more accurately observe the behaviour of the DMM architecture in networks of several sizes and different types of scenarios.

3 Software Components

This section presents the software components that are already available and will be reused for installation on the MEDIEVAL testbed because they can map with the modules defined in D1.1 [1]. They will serve as support for the MEDIEVAL implementations and be adapted within the different work packages (WP2, WP3, WP4 and WP5). Further step in this study will be to investigate how they can be enhanced to fulfil the MEDIEVAL architecture and what are the missing parts that need fresh implementations, such as the ALTO server or the CDN nodes.

3.1 Flow Manager Implementation

In the context of the IETF activities standardizing flow mobility extensions for RFC 5213 [14], ALBLF developed an early version of the Flow Manager (FM) described in D4.1 [15]. The Flow Manager is implemented as user space program and is located in the Local Mobility Anchor (LMA). Figure 4 depicts the relation of the FM with the other mobility modules and its location in the network.

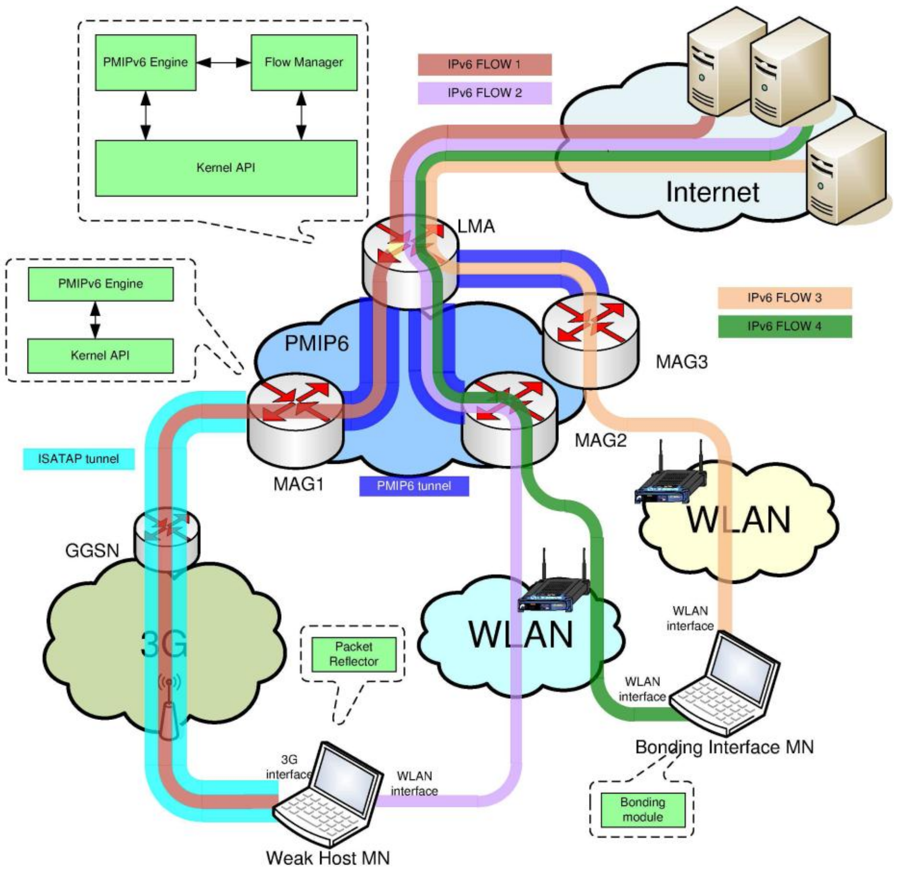

Figure 4: Flow Manager Implementation

The current version of the FM is mostly focused on classifying the packets traversing the LMA and to set specific routing rules for the selected (according to operators' specific rules) IP flows. As specified in D4.1 [15], the FM talks with a module residing on the MN, the Connection Manager (CM).

Figure 5 depicts the different functional blocks of the FM.

The first functional block is the module that extracts the 6-tuple parameters from the packet header. The FM stores in the flow table any new stream with the related parameters, i.e. the 6-tuple, the flow ID. The flow table also stores the tunnel-ID through which the flow is forwarded. When the flow has to be moved (the FM can receive external triggers), the FM generates a "move" request indicating the flow-ID and the tunnel in use. The request dispatcher on the PMIPv6 daemon side checks for the availability of tunnels for that MN by inspecting the BCE (Binding Cache Entry). If the lookup succeeds, the rule manager block adds a ”fwmark-rule” pointing to a route that specifies as default device the tunnel retrieved before. After this rule is set, the packets are forwarded through the new tunnel, bypassing the default route based on longest prefix matching method. Both dispatchers can delete a flow by means of the ”del” request by which a flow is removed from the flow table and from the rule table, if present. Beside the main thread, three additional threads have access to the dispatcher.

The polling thread monitors the flow table looking for expired flows (that is, flows without activity during a certain interval), and it determines the congestion on the tunnels.

The BCE Delete thread listens to BCEs’ de-registration events and deletes (or moves) flows that carry the deleted prefix. Regarding the deletion phase, this feature optimizes performance of the polling thread, in that it anticipates an operation that the polling would have done later on. This feature is very useful to test loss of coverage scenarios. In fact a BCE de-registration might be triggered because one of the active host’s interfaces lost wireless connectivity. In this case, to preserve the seamless mobility service, it would be desirable to move all the streams associated to that IF. Upon receiving a BCE delete event (PBU with lifetime value set to zero), FM moves all the flows that match the prefix and were using the tunnel to the lost-connectivity interface.

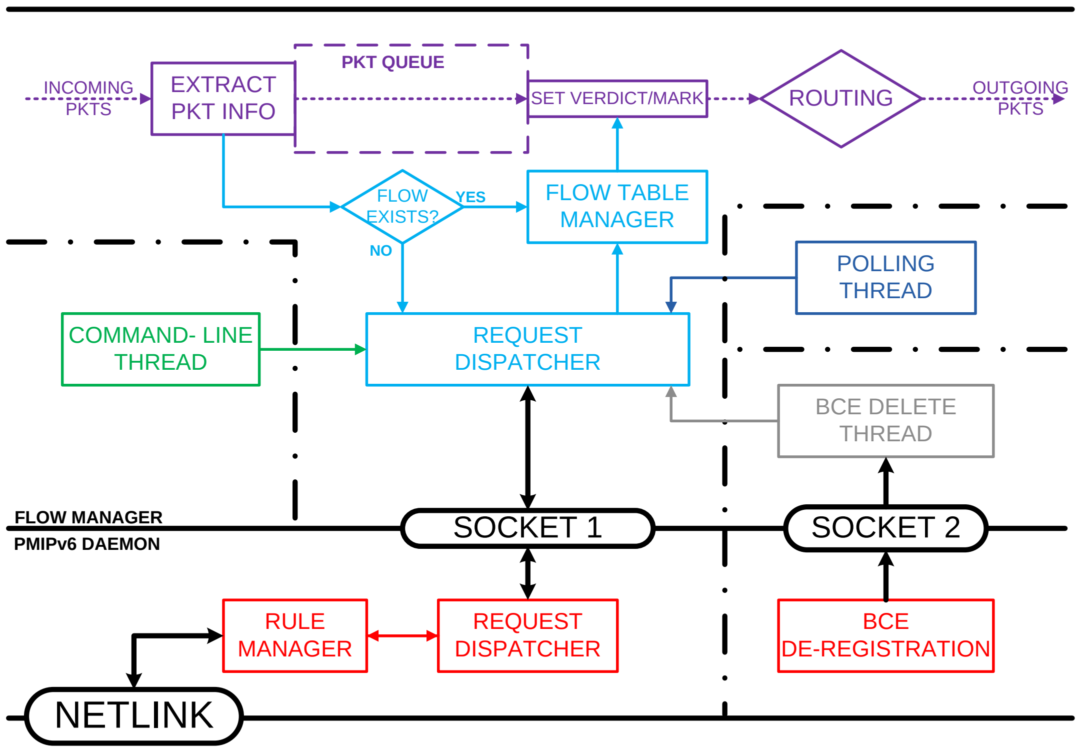

Figure 5: Internal FM functionalities

The FM software will serve as a baseline for the FM module to be developed in the project. The FM will be extended with the required inter-work package interfaces and will be transformed into an MIH client to exchange the IEEE 802.21 protocol. It should be further noted that being the current code user space there are no specific requirements on the Linux kernel version. Finally the FM leverages the IPv6 advanced routing capability, usually available in recent kernel versions.

3.2 OpenAirInterface LTE Platform

The OpenAirInterface platform implements the LTE protocols of the radio interface between a MN (Mobile Node), also referred as UE (User Equipment), and a Base Station (eNodeB).

The MN and eNodeB code is implemented on PC platforms running Linux and a real-time extension named RTAI. The current LTE implementation requires **Ubuntu 10.04** as distribution and **Linux kernel 2.6.32.**

The radio elements are based on proprietary cards developed by EURECOM. These cards are providing transmission of LTE radio signal in TDD (Time Duplex Division) mode over 5MHz bandwidth at 1.9 GHz. The radio cards are viewed as PC-Express devices by the PCs. All the processing of the LTE physical layer is done in software on the PC.

From the user plane perspective, the Linux IPv6 stack is accessing the LTE modem as a device driver. The control plane interface is ready to present a MIH_Link_SAP to upper layers both at the UE and the eNodeB side (collocated with the Access Router).

Figure 6 depicts the integration of OpenAirInterface code in a global setting between a User Equipment and a base station acting also as an access router.

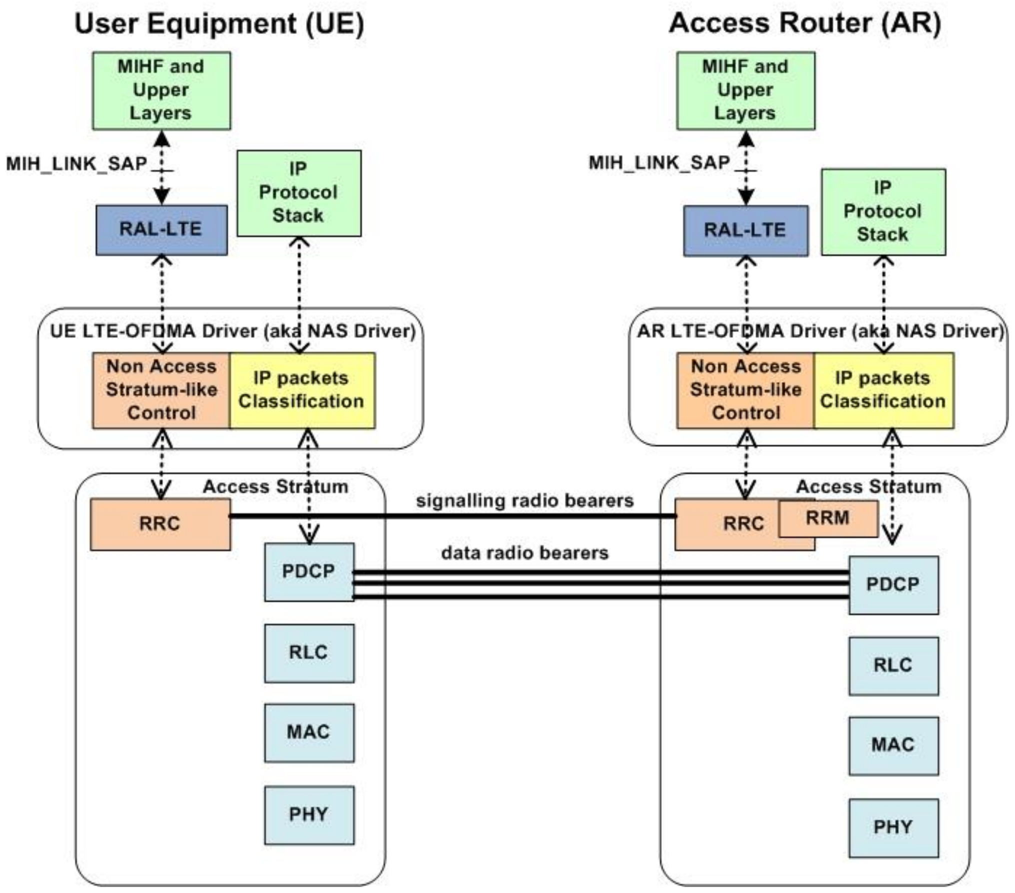

Figure 6: OpenAirInterface LTE software components

The platform contains the following protocol entities:

NAS: The direct inter-connection between LTE and IPv6 is performed using an inter-working function, located in the NAS driver and operating in both the Control Plane and the Data Plane. This function provides the middleware for interfacing IPv6-based mechanisms for signalling and user traffic with 3GPP-specific mechanisms for the access network (e.g. for mobility, call admission, etc.). It is developed as an extension of a standard IPv6/IPv4 network device driver.

RRC: The RRC layer, shared between the UE and the eNodeB, performs the control of the radio interface. It is based on 3GPP 36.331 v9.2.0. The control procedures available in the LTE platform are the following:

- System Information broadcast
- RRC connection establishment
- Signalling data transfer

- Connection reconfiguration (addition and removal of radio bearers, connection release)
- Paging and measurement collection and reporting at UE and eNodeB

These procedures are extended to support MBMS for multicast and broadcast.

MAC: The MAC layer implements a subset of the 3GPP 36-321 release v8.6 in support of BCH, DLSCH, RACH, and ULSCH channels. The MAC implementation includes:

- RRC interface for SI and CCCH, Schedulers, HARQ Support
- Random Access procedures and RNTI management
- RLC interface (AM, UM)

The MAC layer can be extended to provide an interface to external schedulers based on Femtocell forum specifications.

RLC: The RLC layer implements a subset of the 3GPP 36-322 release v8.6 with TM, UM and AM modes.

PDCP: The current PDCP implementation is compatible with 3GPP Release 4, and does not support the security or header compression functionalities with respect to 3GPPP 36-323.

3.3 PMIPv6 Protocol Stack

3.3.1 Open PMIP

Open PMIP (OPMIP) is an open-source implementation of PMIPv6 (RFC 5213 [16]) protocol. Both LMA and MAG capabilities are provided by the software, and may be installed in computers running Linux, which will act as routers providing intra-domain mobility transparently to Mobile Nodes. Although OPMIP has been tested successfully, it is currently still in a beta stage, requiring some support for working appropriately. We are, however, available for direct contact regarding implementation discussions.

OPMIP is implemented as a library of reusable components. The implementation builds on the Boost.Asio library [3], which is a state of the art asynchronous network library, based on the Proactor design pattern [4]. The Proactor is an asynchronous software design pattern for concurrent execution. Thus, the MAG and the LMA components are designed to support concurrency in single and multi-threaded execution.

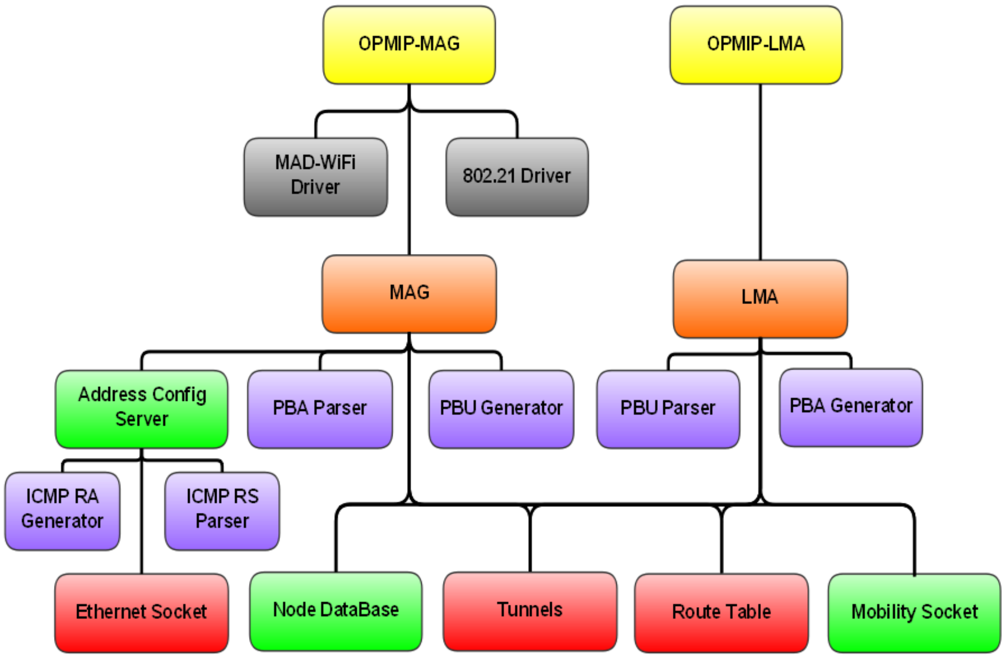

Figure 7: OPMIP hierarchical view of component dependencies.

Figure 7 shows the hierarchical dependencies among the various components that compose the OPMIP implementation. The components in green and red are the lower level components required for MAG and LMA services to manage tunnels, route tables and to communicate with the mobility protocol, the red ones differentiate the components which are abstractions, dependent on the host operating system. The components in purple are helper components to generate and parse mobility protocol messages and ICMP router advertisement/solicitation messages. The orange boxes are final libraries components which provide the MAG and LMA functionality. At last, yellow boxes represent applications / executables which use the MAG and LMA library components to provide the corresponding functionality. In the particular case of the MAG we have additional driver components, in gray, to get attach and detach events to be delivered to the MAG service.

Some of the more relevant components are described below:

- **Node Database component**: implements the conceptual mobile node’s policy profile of the PMIPv6 standard. It is a permanent database of router and mobile nodes, where router nodes refer both to MAGs and LMAs. This database is shared by both the MAG and the LMA services for retrieving the mobile node’s policy profile and for authorization and authentication purposes.
- **Tunnel**: this component manages the creation and deletion of bi-directional tunnel endpoints. It also handles sharing of the same tunnel for common remote router nodes.
- **Routing table**: this component is an abstraction layer of the host operating system interface for managing the creation and deletion of route table entries.
- **Address Config Server**: provides the service for stateless and stateful address autoconfiguration for the mobile nodes.
- **MAG and LMA**: provide the respective PMIPv6 router functionality. In the particular case of the MAG service, it must be notified by mobile node attach and detach events. Thus the OPMIP-MAG application relies on technology specific drivers to get attach and detach events.

The machines acting as mobility management components are required to have a Linux kernel version greater or equal to 2.6.30, with the following enabled capabilities:

- CONFIG_IPV6_SUBTREES enabled;

- IPv6 forwarding[6]
- IPv4 and IPv6 tunnel;
- IPv6 source routing.
- For the MAG, the node is required to have a configured wireless interface card and working as an access point. NOTE: Currently, only wireless cards with MadWifi drivers are supported. Because the opmip-mag attach/detach event monitor driver is only implemented for MadWifi.

3.3.2 OpenAirInterface PMIP

The OpenAirInterface PMIPv6 is an Open Source implementation of PMIPv6 (RFC 5213). It has been tested from Ubuntu 7.02 with Linux 2.6.20 to **Ubuntu 10.04** under Linux vanilla **kernel 2.6.32** reusing Mobile IPv6 for Linux (MIPL) v 2.0.2. All the basic building blocks of MIPL are used in an efficient way as shown in Figure 8. The distribution of source code and documentation can be found at [7].

The OpenAirInterface PMIPv6 implements **LMA** and **MAG** components in software in the same code. In MIPL, Mobile IPv6 is implemented using multi threads: one for handling the ICMPv6 messages, one for handling Mobility Header messages, and another one for handling tasks and time events. To support PMIPv6, the implementation has extended these elements and implemented handlers for all necessary messages and events. ICMPv6 messages and Mobility Header messages are parsed by the Handler as inputs to the Finite State Machine, which is the heart of the system. Two different Finite State Machines are defined for LMA and MAG. They are in charge of making appropriate decisions and controlling all the other elements to provide correct predefined protocol behaviour. The PMIPv6 Binding Cache stores all information about MNs' points of attachment and it is kept up-to-date with the mobility of MNs.

As PMIPv6 implementation is built on top of MIPL version 2.0.2, it could be, in the future, easily integrated in MIPL, growing in line with the standards as well as with MIPL source code.

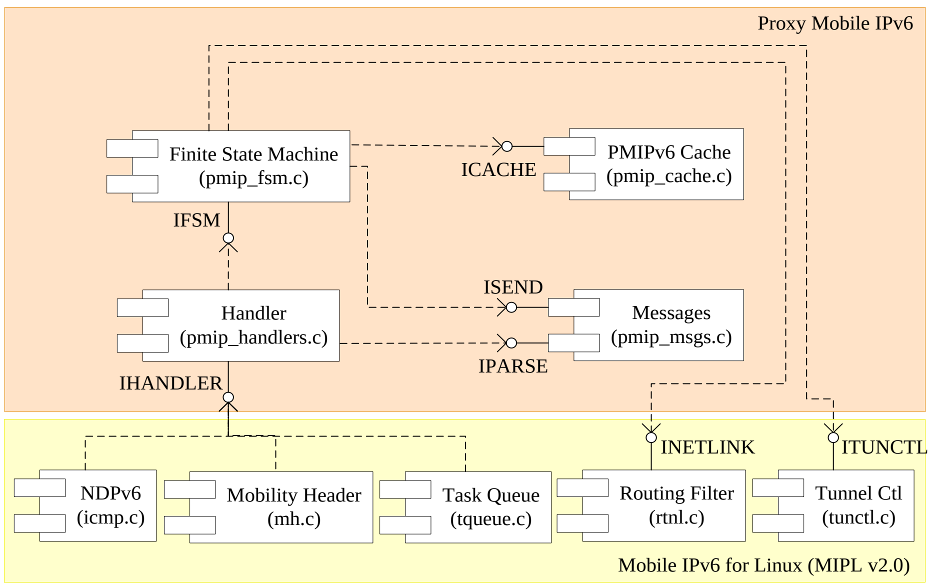

Figure 8: OpenAirInterface PMIPv6 software architecture

[6] * Typically, this can be configured at your /etc/sysctl.conf file, edit it and add the line: *net.ipv6.conf.all.forwarding=1*

Additional functionalities:

- ***Attachment and detachment phases*****:** standard PMIPv6 does not specify any functionality for these two phases as its main purpose is to define only the elements and the signalling messages inside the PMIPv6 domain. As point of reference we have considered [8], in which suggestions on the MN-MAG interface are provided. One possibility is to use an IP layer-based solution; the second one is to develop a specific link-layer mechanism. We have chosen the latest as the use of triggers at layer 2 allows faster movement detection. The current demonstration use the Syslog messages sent by Cisco APs to the MAGs containing "associate", "disassociate" and "reassociate" information to detect attachments and detachments of the MN from the PMIPv6 domain. The integration with the IEEE 802.21 Media Independent Handover (MIH) protocol can be performed by replacing the L2 event handling with MIH events.
- ***Unicast RA (Router Advertisement)***: as the HNP (Home Network Prefix) is unique per MN, it needs to be sent in a unicast RA message by the MAG to the specific MN. This function is integrated in the PMIPv6 daemon for MAGs based on RADVD daemon to unicast RAs. MN's address is auto-configured through IPv6 Stateless Address Auto Configuration.
- ***MAG's link-local address configuration*****:** as specified in the RFC 5213, the MAG is the IPv6 default-router for the mobile node on the access link. However, as the MN moves from one access link to another, the serving MAG on those respective links will send the RA messages. If these RAs are sent using a different link-local address or a different link-layer address, the MN will always detect a new default-router after every handoff. For solving this problem, standard PMIPv6 requires all the MAGs in the domain to use the same link-local and link-layer address on any of the access links wherever the MN attaches.
- ***Tunnelling*****:** bi-directional tunnel is used for routing data traffic to and from the MN between the MAG and the LMA. A tunnel hides the topology and enables a MN to use the address from its HNP from any access link in the PMIPv6 domain. A tunnel may be created dynamically when needed and removed when not needed. However, implementations may choose to use static pre-established tunnels instead of dynamically creating and tearing them down on a need basis. A static and shared tunnel has been implemented between each MAG and the LMA in order to serve all the MNs attached to the same MAG with the same tunnel.
- ***Initial configuration:*** the association between MN identities and Network Prefixes can be learned through a local table configured in the LMA or can be managed through a AAA server co-located with LMA. The code is ready for this interaction.

3.4 Live Video Transmission Units

Live video transmission units to be used in the testbed will be based on dedicated prototypes that will be connected to cameras, and on the other side of the network connected to multiple clients simultaneously. In between there are several ways to implement the distribution system to many clients which may include off the shelf streamers such as SilverLight, FlashMedia, Apple streamers and more. Various encoding profiles of H264 will be provided; in particular we will implement AVC and SVC encoding types. Today off the shelf common streamers does not support the SVC format, thus we may implement a prototype to stream SVC layers to different users in a proprietary manner for the purpose of allowing features like layer dropping and priority handled by the MEDIEVAL system.

A service of Live content includes the content generation part, where the uncompressed video whether it is an analog interface such as SDI or a digital interface such as HDMI between the camera and the encoder will be tested, a real-time compression unit to generate a compressed video according to video control algorithms will be implemented (note that the encoder itself is off the shelf and will not be part of MEDIEVAL development). When off the shelf streamers will be used as reference, the output of the encoding system will be MP4 stream. The streamers will distribute the stream to multiple users using HTTP protocol.

When a proprietary SVC based streamers will be implemented an RTSP stream will be used, in that case the stream will be over UDP and will allow MEDIEVAL system to drop layers and protect layers in un-equal manner.

The MEDIEVAL approach also supports multi IP flows simultaneously to gain throughput and reliability, the Live Units for personal broadcast use case (PBS) will be used as “On the go Producer” units that will deliver streams over multiple LTE and WLAN connections from anywhere anytime. In that case, a server that collects those streams and combine them will also take part in the setup prior to the distribution to the different users.

The following diagram represents the Live Units involved in the service testbed.

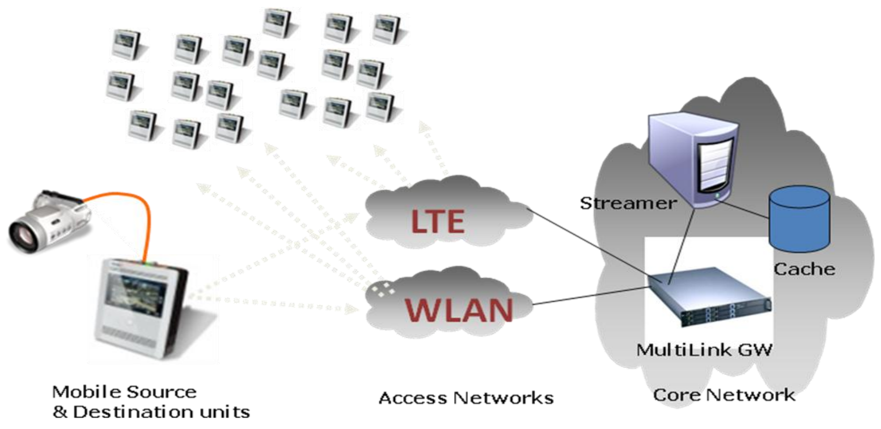

Figure 9: LiveU units for MEDIEVAL testbed

3.5 ODTONE IEEE 802.21 MIH Implementation

ODTONE stands for Open Dot Twenty One [6]. This is an open-source implementation of the Media Independent Handover framework from the IEEE 802.21 Media Independent Services standard. It provides the implementation of a Media Independent Handover Function, including its inherent services (the Media Independent Event Service, the Media Independent Command Service and the Media Independent Information Service) along with supporting mechanisms such as Capability Discovery, MIHF Registration, Event Registration, among others.

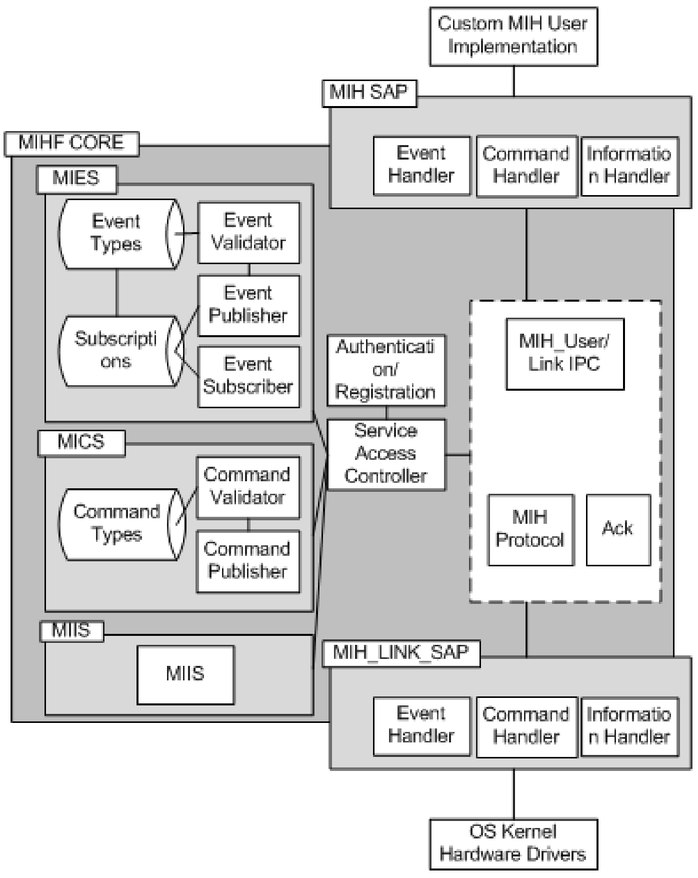

Figure 10: ODTONE Architecture

This software is organized as follows, according to Figure 10:

- The **Communication Handler** is responsible for collecting messages received from the different SAPs or other MIHFs entities and forwarding them to the Service Access Controller.
- The **Service Access Controller** analyzes the header of MIH messages and identifies to which MIH Service they pertain. It is then responsible to forward that message to the respective service component.
- The **Link Manager** provides knowledge to the MIHF about local links available for interaction, as well as specific required configurations.
- The **MIH-User Manager** acknowledges which MIH-Users, both local and remote, interface with the MIHF.
- The **Transaction State Machine Controller** has the responsibility of keeping the state of each remote communication with peer MIHFs entities.
- The MIHF also features the three core MIH Services, each with a set of logical components:
- The **MIES** provides modules to manage the **Subscription**, **Validation** and **Publishing** of events. Respectively, these allow the MIHF to verify which MIH-Users (both local and remote) have subscribed to events originated from that node, if they are properly formatted according to the standard and determine if they need to be forwarded locally or remotely.
- The **MICS** also provides its own set of **Validation** and **Publishing** modules which allow the MIHF to validate received commands (also both local and remote) and handle their publishing.
- Regarding the **MIIS**, the definition of an Information Server (IS) is out-of-scope from the standard. However, most works on 802.21 realize that the inherent database features of the IS place it as a MIH-User. Following this approach, in ODTONE we provide the basic **IE schema** in both **binary** and **RDF**, facilitating the implementation of ISs.

ODTONE possesses unique characteristics that aim to facilitate its deployment into any IEEE 802.21 scenario. One important characteristic is its ability to run in different operating systems. In research projects, such as MEDIEVAL, different devices can each provide their own requirements and come with different operating systems installed. We have developed ODTONE in a way that compilation requirements were reduced to a minimum, which maintaining code flexibility and extendibility. To achieve this aim, ODTONE has been implemented in C++ using Boost libraries, which complement the Standard Template Library with, for example, portable datatypes, networking and low-level I/O, which are also able to be supported in a plethora of different platforms. So far, ODTONE has been successfully compiled and tested in Windows and Linux. Compilation and simple testing have been done in the Android platform, but full-fledged testing using link primitives has not been yet possible due to unavailability of link layer APIs. We are, however, in contacts with interested parties that aim to provide such link layer interactions for ODTONE.

An important consideration which as been impacting the adoption of 802.21 in large scale since its conception, is the availability of 802.21 support at the device driver level. In order for MIH-enabled scenarios to be possible, a MIHF needs to have 802.21 supportive link layers, in order to execute the 802.21 primitive mappings with the link layer primitives. However, since no device drivers which support 802.21 are coming out, this has become a difficult process. To overcome this, ODTONE provides a unique approach reutilizing the MIH Protocol (which is used only for remote interactions) for its internal interaction. This means that the communication between the MIH-Users and the MIHF (e.g., the MIH_SAP) and the communication between the MIHF and the LINK_SAPs is achieved through messages formatted in the MIH Protocol between sockets, instead of a shared API. With this approach, MIH-Users and LINK_SAPs can easily become MIH-enabled by just importing the ODTONE MIH Protocol library, and opening a socket. Then, their already-existing internal procedures just need to de-serialize and serialize MIH Protocol messages, for receiving and sending 802.21 commands and events.

ODTONE is currently in its version 0.3. Instructions for retrieving ODTONE can be found at http://helios.av.it.pt/projects/odtone/ . The project features samples of a very simple MIH-User, as well as 802.11 LINK_SAPs for Windows and Linux featuring one command, used for LINK_SAP testing and development example. We also feature a sample MIIS MIH-User implementing the MIIS RDF Basic Schema, using a sample Turtle database able to understand SPARQL queries.

ODTONE aims to be an IEEE802.21 implementation able to run in different operating systems. So far, it has been successfully compiled in Linux, Windows and Android.

For all platforms:

- Boost C++ Libraries (only the source code) with the Boost.Build configured.

For Linux platforms:

- GCC C++ Toolset

For Windows platforms:

- Windows SDK Libraries and Toolset.

For Android platforms:

- CodeSourcery GNU Toolchain for ARM Processors

3.6 MRD6 Multicast Router Software

Multicast routing daemon for IPv6 (MRD6) is an open-source implementation for Linux and BSD, which supports the following main features:

- MLDv2 (and MLDv1 compatibility)
- PIM-SM (including Source-Specific Multicast support)
- BSR and Embedded-RP
- Partial MBGP support (IPv6 multicast Sub-address Family Identifier Information for populating local MRIB, for routers at the network edge)

- Native and virtual interfaces (IPv6-IPv4, IPv6-IPv6 and TUN/TAP tunnels)
- Remote configuration (using CLI)
- Automatic translation of IPv4 multicast traffic to IPv6

mrd6 is a useful tool in such scenarios as quick network prototyping, testing new network features and even to run in embedded systems. It has as build requirements the following: a GNU build system, including G++ and Make.

mrd6 packs mrd6sh, a tool that, along with built in telnet support, allows to interactively configure and obtain information from mrd6. Most commands may be called by the least common command prefix: e.g. 'sho gr' may be used instead of 'show group'. The available options include the possibility to display groups' information, per group and per source forwarding statistics and others.

More information about this tool, the source file, and configuration examples are available in [5].

3.7 NEMO Basic Support Protocol Implementation

3.7.1 Functionality

The Network Mobility Basic Support (NEMO BS) protocol defined in RFC 3963 [2] is designed as an extension to MIPv6 in order to handle the mobility of a whole network. It introduces the role of Mobile Router (MR), who handles the mobility support in the mobile network, relieving any other node of the mobility management. Therefore, mobility management is transparent to the nodes attached to the MR, enabling session continuity while the network moves.

Figure 11 presents a basic Network Mobility scenario. It consist of three entities: *i)* the Home Agent (HA), in charge of defending the set of prefixes delegated to the mobile network, *ii)* the mobile network itself, accessible through the MR, and *iii)* the Access Routers (AR) that act as access points in a visited network.

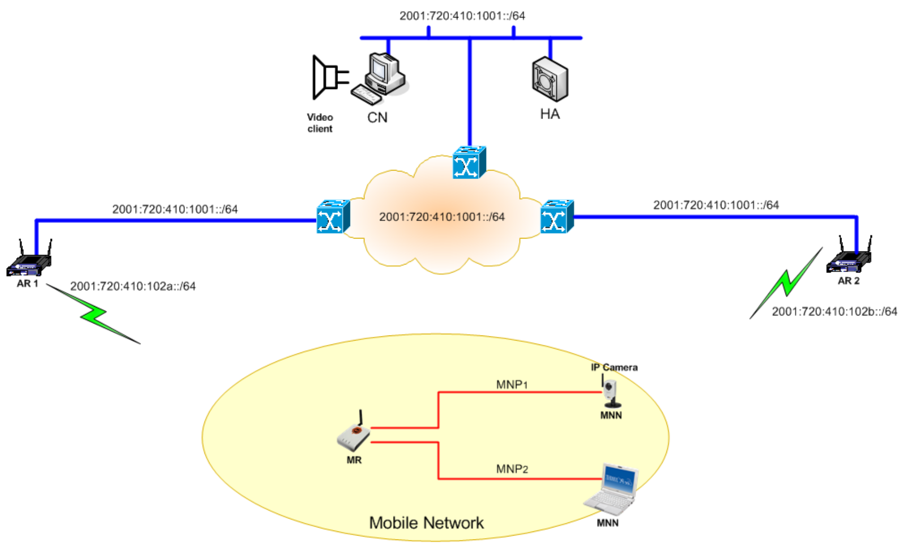

Figure 11: Network Mobility scenario

At UC3M, we have implemented a prototype of the NEMO B.S. protocol. The implementation is programmed in C and it is formed of two different modules that cover the basic functions of the Home Agent and the Mobile Router.

- **Mobile Router**:

The mobile router checks whether it is attached to its home network or to a foreign network. When it detects – by receiving a Router Advertisement– that it is connected to a new network, it acquires a new Care-of Address (CoA), a topologically correct address in the visiting network, and informs the home agent by sending a Binding Update (BU) message of its new location. After receiving confirmation from its home agent, it is established a bidirectional tunnel that will encapsulate all the traffic addressed to or sent from the mobile network. Whenever the mobile router changes the point of attachment, it will detect this movement by the reception of a new Router Advertisement and will update its home agent.

- **Home Agent**:

When the home agent receives a BU from a mobile router that depends on him, it updates and stores the new location of the mobile network in the Binding Cache and confirms the establishment of its tunnel endpoint to the mobile router.

Both modules have an initial stage for configuration of the IP addresses and interfaces. While execution is running and depending on the scenario, the IP addresses and IP routes will be updated according to the signalling messages. When the modules are stopped, there is a clean up process that removes the routes, addresses and tunnel interfaces in order to get the system back to its initial state.

3.7.2 Hardware requirements

Essentially, two processes are needed to run the home agent and mobile router modules respectively. In this case, the home agent has been tested under different distributions in a Linux Ubuntu machine and the mobile router has been tested in a lighter Linux version, OpenWrt[7], in COTS devices. Due to the different architectures and the limited memory and computational resources of these devices, a cross-compilation environment – provided by OpenWrt as well - needs to be installed in a PC in order to generate a compatible executable file, built specifically for the router architecture.

Table 1 summarizes the hardware under which the NEMO BS implementation has been tested.

Table 1: NEMO BS Compatible Hardware

Home Agent	Mobile Router	Access Router
PC Ubuntu 8.04.2 Kernel 2.6.24	Fonera OpenWrt Kamikaze 7.09 Kernel 2.6.21.5	Linksys WRT54GL OpenWrt Kamikaze 7.09 Kernel 2.4.34
PC Ubuntu 8.10 Kernel 2.6.27	Asus WL500g-Premium OpenWrt Kamikaze 8.09 Kernel 2.6.25.17	Any hardware with forwarding capabilities

3.7.3 Software specifications

Two software processes are needed: home agent and mobile router respectively. In addition to that, configuration text files used as input of the modules for initial configuration have to be loaded in each device.

Both modules have implemented a configurable level of verbosity in order to make debugging process easier. Each event implemented is registered by a *syslog* call and a message and a timestamp are printed to the standard output. There are four levels available:

- LOG_DEBUG
- LOG_INFO
- LOG_WARNING
- LOG_ERROR

The user can choose the most useful level of information depending the kind on information to be notified and insert it as the first parameter in the execution command line (i.e ./home_agent –d LOG_DEBUG).

[7] http://www.openwrt.org

It is worth to mention that the sending of Router Advertisements by every AR is unavoidable for the MR to work and start the signalling process, as it is the mechanism that makes it aware of the movement of the network.

- Additional features:
    - Mobile router supports several Mobile Network Prefixes (MNPs). The exact number and the specific set of prefixes handled are provided in the configuration file.
    - Home agent supports several mobile routers. The exact number and the specific set of prefixes handled is provided in the configuration file.
    - Home agent forwards traffic belonging to a mobile router and all of its MNPs.
    - Binding Refresh messages are sent periodically to increase stability.
- Not supported:
    - Several egress interfaces.
    - Nested mobile networks.

4 Simulator

Network simulation is an effective means of researching in network protocols; it is more and more common to see all the proposed approaches formulated to optimize the communication systems also opportunely integrated into modules of existing network simulators.

4.1 Ns-3 LTE Simulator

The focus is ns-3, an open source discrete-event network simulator written in C++, targeted primarily for research and educational use. The LTE module architecture, that has been recently integrated into the latest stable release, ns-3.10, and initially developed during the Google Summer of Code (GSOC) 2010, can be found in [9] , while the current development of the 3GPP compliant framework can be downloaded from [10]. A graphical representation of the proposed protocol stack is given in Figure 12. The actual work tackles the issue of multimedia traffic optimization in LTE networks, whose approach comprises two parts: first, the analysis and design of novel communication techniques; second, the implementation and extension of the network simulation tool to test and validate the proposed approaches.

The LTE PHY model is created starting from the Spectrum Framework proposed in [11]. The LTE propagation loss model is developed in order to compute the power of the received signal. The propagation loss is obtained considering 4 different models (shadowing, multipath, penetration loss and path loss).

As proposed in [12], these models are described in the following:

- Pathloss: PL=128.1+(37.6 log10(R))
- Multipath: Jakes model
- Penetration loss: 10 dB
- Shadowing: log-normal distribution (mean = 0 dB, standard deviation = 8 dB)

For what concerns the protocol stack entities, the framework is the one depicted in Figure 12.

Our first contribution comes from the fact that in the context of LTE, link abstraction models are needed to condense the wideband channel quality measurements performed by the User Equipment (UE) into a small set of Channel Quality Indicators (CQI). To estimate the system performance, a link performance model is then required to produce the results for evaluating the link quality, which is based on resource and power allocation in a certain radio propagation environment. With the proposed modelling technique, accurate link abstraction can be obtained based on the mean mutual information per coded bit (MMIB) metric which is the mean mutual information between coded bits and their Log- Likelihood Ratio (LLR) values. Then, a mapping function providing the associated Block Error Rate (BLER), that represents the Link to System Level (L2S) interface, is integrated in ns-3 in order to design an efficient system error model: a binary random variable with that probability is then drawn and a corresponding error event is generated.

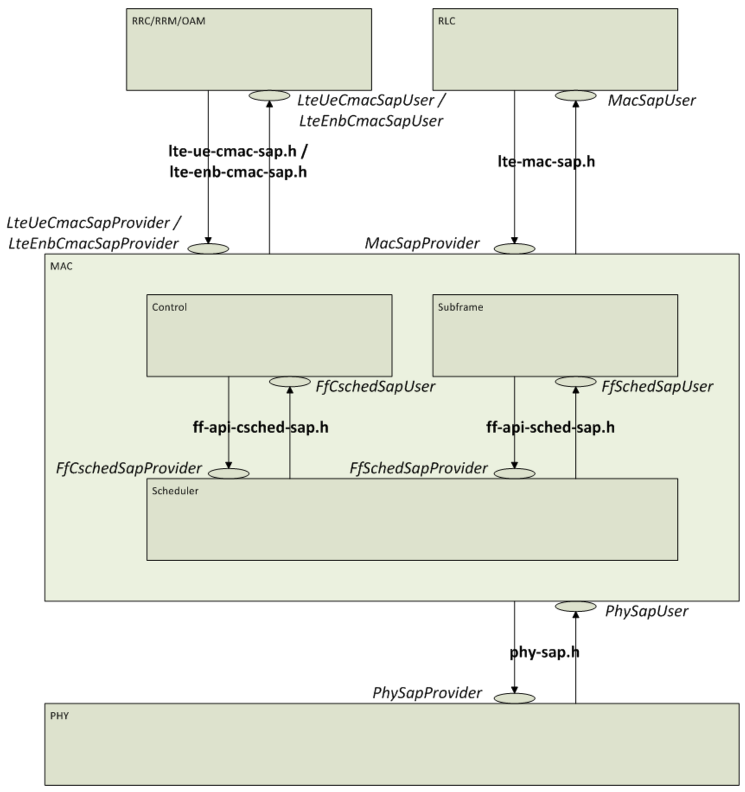

Figure 12: Ns-3 LTE Simulation Framework

In Figure 13 is provided an intuitive description of the mapping function that performs the BLER estimation based on SINR and MCS associated to the different flows.

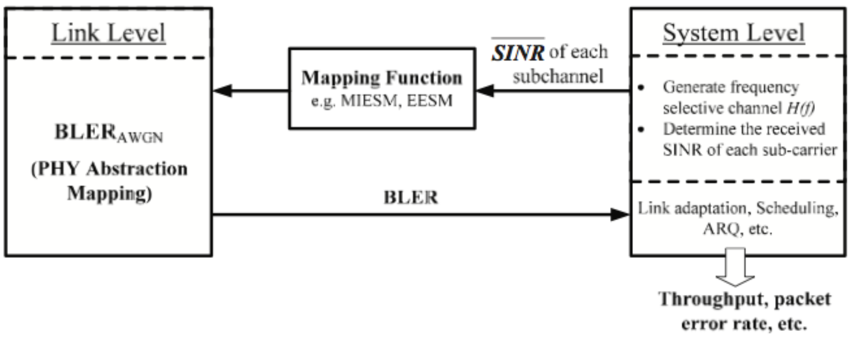
Link Level
BLER_AWGN
(PHY Abstraction Mapping)
Mapping Function
e.g. MIESM, EESM
SINR of each subchannel
System Level
Generate frequency selective channel H(f)
Determine the received SINR of each sub-carrier
Link adaptation, Scheduling, ARQ, etc.
BLER
Throughput, packet error rate, etc.

Figure 13: Ns-3 LTE BLER Estimation

Moreover, error control techniques (HARQ) standard compliant have been implemented and integrated into the LTE module [13]; such benefits, in term of throughput and delay, have been presented along with the comparison among different retransmission policies. Regarding scheduling and resource allocation strategies, the algorithms are still under construction, for both unicast and multicast scenarios.

As next steps, we plan to integrate realistic video traces in ns-3, and implement admission control policies.

5 Preliminary methodology for development and integration

The MEDIEVAL methodology adopts an incremental approach, scheduling an iteration of design and prototyping phases, while maintaining an always-on testing platform starting from the end of the first year of the project. The WP6 implementation plan includes the following steps:

- First year : design phase, architecture and initial interface specification
- Second year : first implementation with selected integration and refinement of the specifications
- Third year : revised implementation with full integration and final specification.

This incremental approach will ensure the adequate quality of the produced systems specifications allowing early testing of MEDIEVAL components, thus reducing the risks of a very late integration plan

This approach is completed by a common and light product assurance methodology for development and software implementation defined with the objective to enable a fast integration process. This methodology is kept at a basic level to ensure its applicability in the project. Some quality criteria will further evaluate the advancement and maturity of each software component versus its specifications, before it is included in the incremental testbed.

Good coding practices

- Each software component is packaged together with text files describing the author and his contact information, the pre-requisite pieces of software and assumptions, how to install the component, how to compile it, how to execute it.
- Each source code file starts with a header that contains its name, the component to which it belongs, the author name and his organization, the description of bugs corrected and modifications applied after the first delivery.
- The source code contains inline documentation.
- The component configuration is contained in a separate file that can be modified without needing to recompile the whole component. It should be self explainable or well documented with comment lines.
- All external links are defined as relative, and not fully defined, paths
- Hard-coded values/names are avoided, the use of global variables is limited
- Blocking state machines should be avoided, especially if messages are missing, syscalls fail or values are corrupted
- Easy debugging should be enabled, if possible by logging traces and outputs in external files with incremental file names.

Testing and delivery

- Each component will be uploaded in its folder located a common repository structure, if possible both in source and binary format.
- Before delivery, each piece of code must have achieved successfully a unit test (function by function) and a functional test (full scenario according to specification and involving a dummy module for testing the external interfaces). These tests should be prepared and documented for regression testing after each problem correction.
- Each component must be delivered together with its documentation including its description as defined in the “good practice” section, the specifications and interfaces it supports, the report of unit and functional tests executed, including the percentage of software covered by these tests (ideally 100%). The functional test report should be accompanied by an example trace of successful result.

- Problems found during tests are logged in a common database and accompanied by a full description of the environment and conditions under which it happened and of the logs that were recorded when it happened. The developer is expected to investigate the problem and correct it in the shortest delay possible.

6 Summary and Conclusion

This document described and presented the initial setup of the MEDIEVAL test site. It presented the preliminary testbed description, the resources committed to the project by the different partners, their hardware and software requirements and the requirements for the test site. Even though these are still in early stages and will be developed during the second year of the project, it was important for the success of the project to push the agreement of partners involved in the various subsystems. The definition of a work plan for proper deployment integration and test of the software components and a process to guide the implementation on the platforms, environment and tool versions used by all partners in their own controlled laboratories will facilitate the integration and bugs fixing during the pre-testing process phases, reducing their impact in the duration of the project.

This deliverable also introduces some software components that are already available and that are candidates for installation on the upcoming MEDIEVAL Test site: Flow Manager Software, OpenAirInterface LTE Platform, Open PMIP Protocol Stack, and OpenAirInterface PMIP Protocol Stack. Live Video Transmission Units, ODTONE IEEE 802.21 MIH Protocol Stack, MRD6 Multicast Router Software and NEMO Basic Support Protocol.

Next, it described some additional simulation tools and platform that will be used by the project, and finally listed some good practices to be followed before and during deployment of software in a preliminary version of the project methodology for development and integration.

In the coming months of this work package, the MEDIEVAL project will be working into improving the test plan and starting the integration of these modules on the testbed. First, by building a test plan which will take into account a common and light product assurance methodology for development and software implementation, allowing a fast integration process with some quality criteria that will further evaluate the advancement and maturity of each software component versus its specifications. Next WP6 will define the software modules and functionalities that will be required for a first MEDIEVAL prototype and also timeline for the integration and test of these components during the next year. Overall, the aim will be to define, build and maintain the testbed, including the integration of the chosen set of software components. Still this will be done in an incremental way, adding modules as they are developed by the partners, but only when these are properly tested and have passed successfully some validation procedures, in order to avoid compromising the usability of the testbed by other partners. Such approach allows an early availability of the testbed so testing may also start early and the results fed back into the work packages and member teams.

Concerning LTE simulation, first MEDIEVAL partners plan to contribute in the open source community providing new NS-3 modules for LTE, such as link error prediction model and HARQ modules. The aim of this effort is to consolidate and extend the features of the LTE simulator in NS-3, looking at the needs of the MEDIEVAL project. For example, new modules to extend the functionalities of the wireless access are planned to be implemented in the LTE NS-3 simulator. The goal is to assess the performance of the LTE network with techniques proposed and currently under investigation by other partners of the project, favouring the exchange of ideas and skills. Finally, MEDIEVAL will use the NS-3 platform to consolidate the analytical analysis proposed for the project.

Acknowledgements and Disclaimer

This work was partially funded by the European Commission within the 7th Framework Program in the context of the ICT project MEDIEVAL (Grant Agreement No. 258053). The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the MEDIEVAL project or the European Commission.

References

[1] MEDIEVAL, Deliverable D1.1, Preliminary architecture design, June 2011

[2] Devarapalli, V., Wakikawa, R., Petrescu, A., and P. Thubert, "Network Mobility (NEMO) Basic Support Protocol", RFC 3963, January 2005.

[3] http://www.boost.org/

[4] http://www.boost.org/doc/libs/1_42_0/doc/html/boost_asio/overview/core/async.html

[5] http://fivebits.net/proj/mrd6/

[6] ODTONE - http://atnog.av.it.pt/projects/odtone

[7] OpenAirInterface ProxyMIP http://www.openairinterface.org/components/page1095.en.htm

[8] J. Laganier, S. Narayanan, and P. McCann, ‘‘Interface between a Proxy MIPv6 Mobility Access Gateway and a Mobile Node’’, draft ietf-netlmm-mn-ar-if-03, IETF Internet draft, February 2008.

[9] G. Piro, N. Baldo, M. Miozzo, "An LTE module for the ns-3 network simulator," accepted to WNS3, SIMUTools 2011.

[10] ns-3 Lena Project [Online]. Available: http://code.nsnam.org/nbaldo/ns-3-lena-trunk/

[11] N. Baldo and M. Miozzo, Spectrum-aware Channel and PHY layer modeling for ns3, Proceedings of ICST NSTools 2009, Pisa, Italy.

[12] 3GPP TS 25.814 (http://www.3gpp.org/ftp/specs/html-INFO/25814.htm)

[13] LTE contributions (DEI) [Online]. Available: http://sourceforge.net/projects/ns3-lte/

[14] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury and B. Patil, "Proxy Mobile IPv6", RFC 5213, August 2008.

[15] MEDIEVAL Project, Deliverable D4.1 - "Light IP Mobility architecture for Video Services: initial architecture", June 2011

[16] Gundavelli, S., Leung, K., Devarapalli, V., Chowdhury, K., and Patil, B., "Proxy Mobile IPv6", RFC 5213, August 2008