Seamless Handover: The Hurricane Media Independent Handover (MIH) Complete Reference Implementation[*]

Paulo Pires, Luis Teixeira, and Luís Miguel Campos

PDM&FC
Avenida Conde Valbom – 30, Lisboa 1050-068, Portugal
{paulo.pires,luis.teixeira,luis.campos}@pdmfc.com

Abstract. The aim of the HURRICANE framework is to support seamless handovers between 3GPP, DVB, IEEE 802.11 and 802.16 networks. In this framework, the IEEE 802.21, Media-Independent Handover (MIH) [1] provides the basis for the handover-enabling functionality. In this paper, it will be described the effort taken in order to have a fully-blown IEEE 802.21 implementation with the addition of a Media-Independent Information Service server facility.

Keywords: Vertical Handover, IEEE 802.21, Media Independent Service, Cooperative Radio Networks.

1 Introduction

Currently, there is no complete IEEE 802.21 implementation in open-source available to developers. What do we mean by complete?

The standard specifies the following key facilities:

- Event Service – mechanism responsible for the propagation of link-layer events, i.e. link unavailability or signal power degradation, to both local and remote MIH Users that have previously registered for them, by means of this very service.
- Command Service – mechanism responsible for controlling the link-layer upon MIH User request, i.e. shutdown a specified link or change link parameters.
- Information Service – querying mechanism meant to provide network information to MIH Users. This service depends heavily on another facility, an Information Server, which will be described later in this document.
- Service Management – mechanism responsible for configuring a MIH entity. It provides MIH Capability Discovery, MIH Registration and Transport facilities.

[*] The presented work is supported by the project INFSO-ICT-216006//HURRICANE/ /(Handovers for Ubiquitous and optimal bRoadband connectIvity among CooperAtive Networking Environments), which is partly funded by the European Union.

In addition and to ensure communication between the different layers on one MIH entity, and between MIH entities, the following service access-points (SAPs) are documented:

- MIH SAP – interface for communications between a MIH User and the local MIHF.
- NET SAP – interface for communications between MIH peers (different MIH entities). This interface relies on L2 or L3 transport mechanisms, like TCP or UDP.
- LINK SAP – interface for communications between a MIHF and the local link-layer.

Finally, the Information Service functionality (MIIS) depends on the existence of an Information Server. Information requests (queries) are originated in MIH Users, such as a mobility management block in the Mobile Node, and are delegated to a local or remote Information Server, which instead will respond to those same queries.

This entity, the Information Server (IS), is not specified in the standard. So, it makes sense to exist in the form of an MIH User that shall be responsible for answering queries forwarded from the MIIS residing in the local MIHF (local to the IS).

The complete MIHF architecture is shown in figure 1

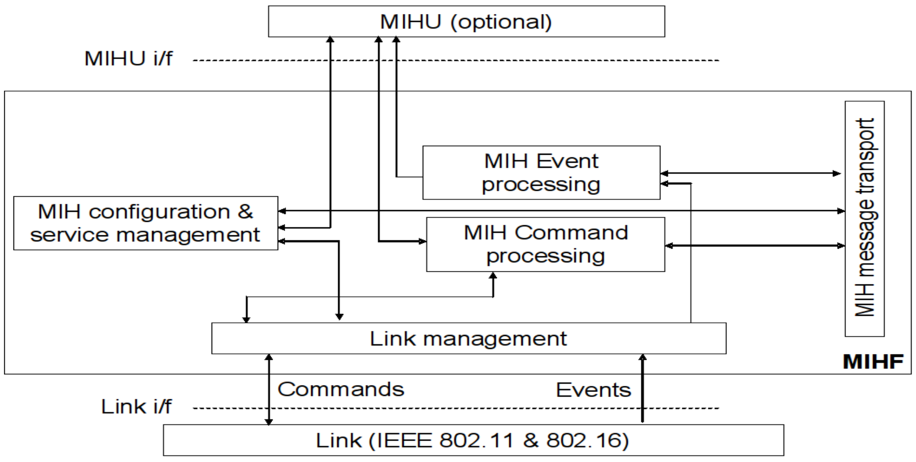

Fig. 1. Full-featured MIHF architecture

All of the above functional blocks have to be provided in order to achieve the goal of having a fully-blown MIH-aware environment. The details of the implementation developed in the scope of the EU-funded project, HURRICANE, is the subject of this paper.

2 Prior Art

There are some ongoing efforts to achieve the same objective, most notably by the Heterogeneous Networking Group [2], from IT-Aveiro. Some concrete ideas on the HURRICANE architecture in fact flourished from several ad-hoc discussions between the project developers from both research gropus, but a joint venture between the two research groups couldn't take place. In spite of this, given that the HNG project, named ODTONE [3] was made publicly available, some code blocks were adopted later on by our own implementation.

When it was first publicly released, ODTONE was on version 0.0.1-alpha. It provided encoding and decoding mechanisms for most of the datatypes described in Annex F of the standard. Also, *Event* and *Command Services* were implemented. MIH SAP and LINK SAP were partially available along with UDP transport support for remote communications.

3 Implementation

3.1 MIHF

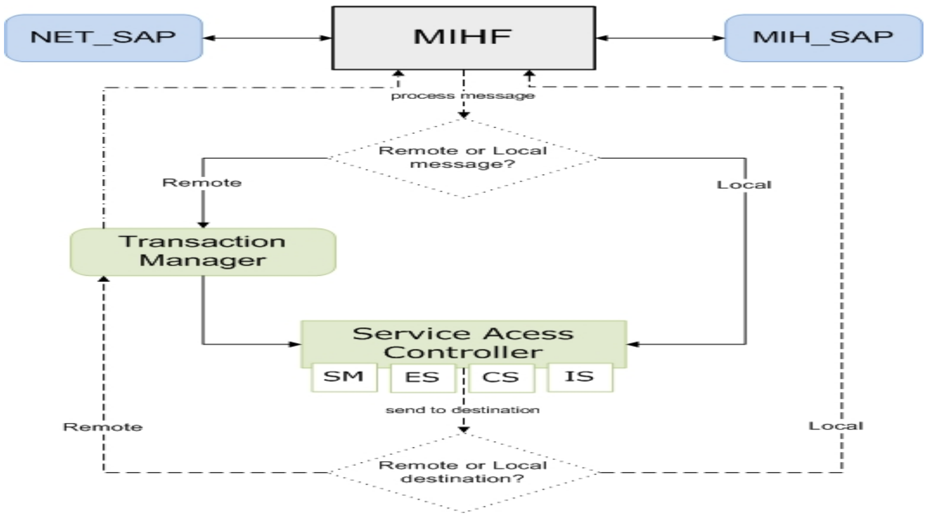

Fig. 2. HURRICANE MIHF architecture

Figure 2, shows the proposed new HURRICANE architecture, based on ODTONE.

NET_SAP is the block responsible for the communications between MIH peers. It listens for incoming messages and is also the exit point for outgoing messages. The IP version and protocol to use is configurable. Every incoming/outgoing messages must be delivered to/by the **Transaction Manager** (detailed further on).

MIH_SAP is the block responsible for the communications between the MIHF and MIH Users. It listens for local incoming messages from the application layer and is also the exit point for local outgoing messages to the application layer. The IP version and protocol to use is configurable.

Both entry points, queue the messages to be preprocessed by the **MIHF**. Depending on whether a message comes from a remote MIHF or a local MIH User, it is then delegated for further processing. Messages between remote MIH peers must always pass through a **Transaction Manager**, which is the block responsible for the source and destination transaction state machines for each message, and then dispatched to the **Service Access Controller**. Local messages go directly into Service Access Controller.

The **Transaction Manager** is the block responsible for the **source** and **destination transaction state machines**, as defined in section 8.2.3 of the standard. Its main purpose is to guarantee the state of each message exchanged between MIHF entities.

The **Service Access Controller** is basically a triage mechanism where a message is parsed and delegated to a specific service depending on the information present in its header. The possible services are the **Event Service**, **Command Service** and the **Information Service**.

ODTONE already provides these same facilities. Some have different names, but their purpose is of course the same. The differences lie in the different implementation approaches. Those include:

- <u>Elimination of singleton pattern</u> - With scalability in mind we have removed all singleton code portions. The access to the affected objects is now achieved through object-factories, which main purposes are to guarantee the uniqueness (i.e.: **MIH_SAP**) and proper access control to objects;
- <u>One generic communication-responsible object</u> - What you see as **NET_SAP** and **MIH_SAP** are actually two instances of the same object but with different property-values, like ports to be listening on, IPv6 support, etc. Others can be easily added, like MIH_LINK_SAPs for every network device available;
- <u>Configurable UDP/TCP and IPv4/IPv6 support</u> - When the program starts, a configuration file is read and the communication mechanisms are configured, among others, according to it. ODTONE only supports UDP over IPv4 and it's hard-coded;
- <u>Information Service</u> – We provide only binary-querying (TLV) support. This relies on a facility not described in the standard – the **Information Server**, which will be described further on.
- <u>Basic MIH Registration support</u> – Registration happens between two MIHF entities after *Capability Discovery* is performed. This was not very clear to us at the beginning, so we are now looking towards a new approach on this block.
- Mature IEEE 802.11 (WLAN) LINK SAP implementation – Linux-only and confined to hardware limitations. This is described in more detail in the following sub-section.

The code is 100% C++ and it has been tried in all major Linux distributions with success. Even though compilation is possible in Windows and Mac OS X (and eventually some other Unix flavors). Some open-source libraries [5] are used, like *boost::asio* for networking and timed-operations; *boost::variant* is used for the basic datatype CHOICE implementation and *boost::optional* is used to deal with optional object attributes.

3.2 LINK SAP

Even though this is outside of the scope of the standard, we had to implement it in order to test our MIHF. Like ODTONE, we provide an application just like any other MIH User. But, instead of aiming only to provide a way of others to implement such facilities, we actually provide a working one for managing IEEE802.11 devices.

The implemented software listens for MIH Commands coming from the local MIHF, generates proper responses and delivers them back to the MIHF, which then forwards them to the requester MIH User. On the other hand, it detects changes in the link-layer and reports them to the local MIHF by means of MIH Events.

You may notice that we do not refer to MIH Link Commands or MIH Link Events. This is because we have decided to reuse the MIH_SAP primitives instead of using the MIH_LINK_SAP ones. MIH_SAP primitives are enough for the task at hands and by reusing them we are saving on processing resources. Just think that we are not translating the messages to the destination MIH Users, from MIH_LINK_SAP to MIH_SAP primitives.

One example is the *MIH_Link_Configure_Thresholds*. Thresholds can be set up so that a report is generated whenever a link parameter crosses an arbitrary boundary.

Ideally, one should be able to ask the network link driver to report back whenever a watched parameter crosses the threshold previously setup. However, there must be a fallback plan in case there's no such support from the network stack. Our fallback plan is active polling. On an interval of N seconds the software inquiries the networking device for the desired parameter. The value of N is now hard-coded but will eventually become configurable.

When a threshold is crossed, a *MIH_Link_Parameters_Report* message containing the parameter type, value, and the threshold that has just been crossed is created and sent to the local MIHF. The local MIHF, be means of the **Event Service** then forwards this MIH Event to any MIH User that has registered for this particular event.

3.3 Android Support

Our MIHF implementation is proven to run on top of Android, but this same platform provides a JAVA-only API to control the networking environment (WLAN and 3G). Some tests have been successfully performed, but in order to get a working LINK SAP, we have to engage in another development effort [10] to be taken outside of the HURRICANE project scope.

3.4 Information Server (IS)

Mobile networks are highly dynamic environments, especially in what concerns to resource availability. This fact leads to high amounts of data being loaded, queried and updated simultaneously into an identifiable repository to which MIH Users will interact with. This repository is what we called, Information Server (IS).

With this in mind, we had to think of a scalable solution to manage our data while at the same time provide an interface for processing multiple-concurrent queries. An SQL RDBMS is the back-end solution we selected.

Currently, a JAVA application has been developed to behave as our Information Server. We have decided to go with JAVA, because it's a high-level language with access to a wide range of tools and frameworks, and because we don't have to worry about scarce resources, as we will be operating on the network side. Also, multi-platform support was a strong consideration.

This decision will boost development, and provide easy functionality additions in the future (i.e.: Web services for business tools integration, native integration in a clustered environment, etc.).

Let us describe a simple query/response scenario, between a Mobile Node and a remote Information Server.

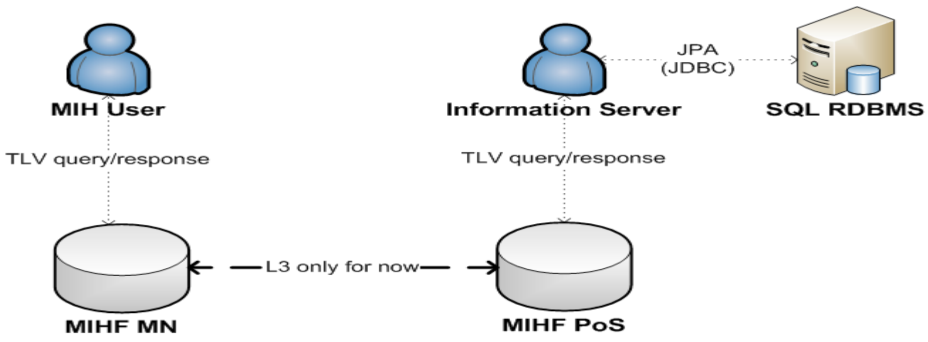

Fig. 3. Query/response scenario in the context of HURRICANE

Figure 3 shows that all the information resides in a centralized database, which is queried through this JAVA application by means of the Java Persistence API[6].

JPA is a JAVA standard that provides a Plain-Old Java Object (POJO) persistence model for object-relational mapping, initially designed inside the EJB standardization effort. Currently, JPA is on its second incarnation (JPA 2.0) providing support for applications outside the EJB context, namely Web applications and standalone clients.

Some advantages of following this approach are:

1. Vendors like Oracle and JBoss have open-source implementations of this standard, TopLink Essentials [7] and Hibernate [8], respectively.
2. Many RDBMS are supported.

3. Provides a query-language (JPQL) and a criteria-query API, for standard querying in your code without worrying about the underlying RDBMS query syntax.
4. You application can be rapidly and easily ported to support any of the supported RDBMS, so you don't lock-in the mobile operator to the choice of RDBMS taken.

Currently, we are using H2 database engine [9] in our testbed. Our database is currently populated by means of another JAVA application in a static way.

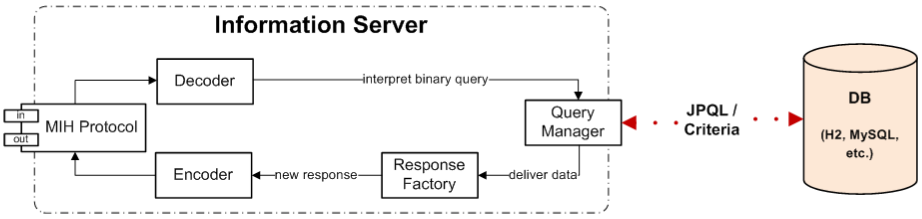

Fig. 4. Information Server internal architecture

Obviously, the MIH protocol, described in the Chapter 8 of the standard, had to be implemented partially. Not all primitives are implemented, but only the ones related to the Information Service, which are:

- *MIH_Get_Information.request*
- *MIH_Get_Information.indication*
- *MIH_Get_Information.response*
- *MIH_Get_Information.confirm*
- *MIH_Push_Information.indication*

The message encoding and decoding mechanisms are responsibility of **Encoder** and **Decoder**, respectively.

After a message is received, the Decoder disassembles it and sends it to the **Query Manager**, the responsible block for interpreting the binary query provided. This is also responsible for querying the database, by means of JPQL/Criteria APIs.

When the results from the database are received, they are then assembled into a new message in the **Response Factory**, which will pass the new response to the Encoder for proper assembly. The outgoing message is then dispatched to the source of the query, usually the local MIHF.

Below is an example of a query and subsequent response. The scenario is as follows:

- We have two networks of type 802.11 and one of type 802.16 in our database. All of them have different costs.
- We want to query for all 802.11 networks but select the cheapest one as our handover target network.

Table 1. Query code snippet

```
// declare MIH message object

odtone::mih::message msg;

odtone::mih::iq_bin_data_list iq_list;

odtone::mih::iq_bin_data iq_data;

odtone::mih::net_type_inc net_type;

// Define network type as IEEE802.11

net_type.set(odtone::mih::net_type_inc_ieee802_11);

// Put networking type into IQ_BIN_DATA

iq_data._net_type_inc = net_type;

// Put IQ_BIN_DATA into IQ_BIN_DATA_LIST

iq_list.push_back(iq_data);

// Encode a MIH_Get_Information.request message

msg << odtone::mih::request(odtone::mih::request::get_information, _mihfid)

& odtone::mih::tlv_info_query_bin_data_list(iq_list)

& odtone::mih::tlv_unauthenticated_info_req(true); // set UIR in the payload

msg.uir(true); // set UIR in the header

// Send the message to the local MIHF for remote dispatch.
```

The message is dispatched to the local MIHF which then forwards it to the **Information Server**. One can notice that it is not provided the cost information. This is due to the decision that the Information Server will not perform any arithmetic or logical operations as we want to provide the information to as soon as possible. It is up to the MIH User to compute the operations it sees fit.

The existing code to handle the message in the Information Server is way too much to put an example here, but we provide some logging info.

Table 2. Decoding of the query and database fetch

```
21:11:46.678 ] TRACE FrameToRequestDecoder - Received MIHFrame:
MIHFrame{header=MIHHeader{version=1, ackReq=false, ackRsp=false, unauthenticatedInfoReq=false,
moreFragment=false, fragmentNumber=0, reserved1=0,
messageID=MIH_GET_INFORMATION_INDICATION, reserved2=0, transactionID=0},
payload=MIHPayload{source=MIHF_ID{mihfId='mihf'},
destination=MIHF_ID{mihfId='information_server'},

21:11:47.029 ] DEBUG FrameToRequestDecoder - Created request:
com.pdmfc.hurricane.miis.command.GetInformationRequest@5d0e8647

21:11:47.031 ] TRACE ServiceRequestHandler - Setting reply address to: /127.0.0.1:6666

21:11:47.033 ] DEBUG ServiceRequestHandler - Executing ServiceRequest:
com.pdmfc.hurricane.miis.command.GetInformationRequest@5d0e8647

21:11:47.040 ] TRACE GetInformationRequest - Executing a MIH_Get_Information.indication

21:11:47.045 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.MIH_Get_Information_indication.binaryDataList

21:11:47.052 ] TRACE TLVSerialiser - Starting to deserialise a stream of bytes

21:11:47.052 ] TRACE TLVSerialiser - TLV_01 type=INFO_QUERY_BINARY_DATA_LIST

21:11:47.053 ] TRACE TLVSerialiser - TLV_01 length=11

21:11:47.055 ] TRACE TLVSerialiser - TLV_01 value=SlicedChannelBuffer(ridx=0, widx=11, cap=11)

21:11:47.055 ] TRACE TLVSerialiser - TLV_02
type=UNAUTHENTICATED_INFORMATION_REQUEST

21:11:47.056 ] TRACE TLVSerialiser - TLV_02 length=1

21:11:47.056 ] TRACE TLVSerialiser - TLV_02 value=SlicedChannelBuffer(ridx=0, widx=1, cap=1)

21:11:47.082 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.IQ_BIN_DATA.querierLoc
```

Table 2. (*continued*)

```
21:11:47.084 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.IQ_BIN_DATA.netTypeInc

21:11:47.085 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.query.NET_TYPE_INC.linkTypes

21:11:47.087 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.IQ_BIN_DATA.netwkInc

21:11:47.087 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.IQ_BIN_DATA.rptTempl

21:11:47.089 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.IQ_BIN_DATA.rptLimit

21:11:47.089 ] TRACE GenericDecoder - Decoding field
com.pdmfc.hurricane.miis.mihf.IQ_BIN_DATA.currPref

21:11:47.090 ] DEBUG GetInformationRequest - Processing:
MIH_Get_Information_indication{binaryDataList=[IQ_BIN_DATA{querierLoc=null,
netTypeInc=NET_TYPE_INC{linkTypes={5}}, netwkInc=null, rptTempl=null, rptLimit=null,
currPref=null}]}

21:11:47.244 ] DEBUG GetInformationRequest - Fetched 2 networks from database

21:11:47.300 ] DEBUG GetInformationRequest - Creating reply frame
```

Table 3. The response being encoded with query results

```
21:11:47.794 TRACE GenericEncoder - Encoding field class
com.pdmfc.hurricane.miis.mihf.identification.MIHF_ID.mihfId

21:11:47.795 TRACE GenericEncoder - Encoding: MIHF_ID{mihfId='information_server'}

21:11:47.795 TRACE GenericEncoder - Encoding field class
com.pdmfc.hurricane.miis.protocol.frame.MIHPayload.destination

21:11:47.795 TRACE GenericEncoder - Encoding as @SEQUENCE: MIHF_ID{mihfId='mihf'}

21:11:47.795 TRACE GenericEncoder - Encoding field class
com.pdmfc.hurricane.miis.mihf.identification.MIHF_ID.mihfId
```

Table 3. (*continued*)

```
21:11:47.796 TRACE GenericEncoder - Encoding: MIHF_ID{mihfId='mihf'}

21:11:47.796 TRACE GenericEncoder - Encoding field class
com.pdmfc.hurricane.miis.protocol.frame.MIHPayload.serviceSpecificTLVs

21:11:47.823 TRACE GenericEncoder - Encoding:
MIHPayload{source=MIHF_ID{mihfId='information_server'}

21:11:47.823 TRACE FrameCodec - Encoding frame header: MIHHeader{version=1, ackReq=false,
ackRsp=false, unauthenticatedInfoReq=false, moreFragment=false, fragmentNumber=0, reserved1=0,
messageID=MIH_GET_INFORMATION_RESPONSE, reserved2=0, transactionID=0}

21:11:47.924 TRACE FrameCodec - Encoded frame dump: 00 00 48 01 00 00 03 42 01 13 12 69 6E 66
6F 72 6D 61 74 69 6F 6E 5F 73 65 72 76 65 72 02 05 04 6D 69 68 66 03 01 00 30 82 02 9F 01 01 10 00
03 00 82 02 96 10 00 03 01 82 01 04 10 00 00 00 19 01 13 01 00 00 00 00 00 00 00 01 0C 74 79 70
65 5F 65 78 74 5F 30 30 30 10 00 00 01 14 12 48 55 52 52 49 43 41 4E 45 2D 46 4F 52 45 49 47 4E 42
00 10 00 00 02 15 14 73 65 72 76 69 63 65 5F 70 72 6F 76 69 64 65 72 5F 30 30 30 10 00 00 03 02 41
41 10 00 01 00 0E 0D 6E 65 74 77 6F 72 6B 5F 69 64 5F 30 30 10 00 01 01 0E 0D 6E 65 74 5F 61 75 78
5F 69 64 5F 30 30 10 00 01 03 0A 01 00 00 00 32 00 00 45 55 52 10 00 01 05 06 00 00 00 00 00 00 10
00 01 06 04 00 00 00 00 10 00 01 07 03 41 41 00 10 00 01 08 05 01 00 00 00 00 10 00 01 09 13 00 00 00
02 01 00 01 09 31 32 37 2E 30 2E 30 2E 31 00 00 10 00 01 0A 04 00 00 02 00 10 00 01 0B 01 00 10 00
01 0C 02 00 02 10 00 01 0D 05 00 00 00 00 00 10 00 01 0E 11 01 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 10 00 01 0F 01 01 10 00 03 02 78 10 00 02 00 03 03 01 30 10 00 02 01 12 00 00 41 41 0D
63 69 76 69 63 5F 61 64 64 72 5F 30 30 10 00 02 02 08 00 00 00 00 00 00 00 00 10 00 02 03 25 01 13
01 00 00 00 00 00 00 00 00 01 0A 74 79 70 65 5F 65 78 74 5F 30 04 0B 33 47 50 50 5F 41 44 44 52 5F
30 00 10 00 02 04 0C 08 00 01 08 31 30 2E 30 2E 30 2E 30 10 00 02 05 0C 00 01 09 31 32 37 2E 30 2E
30 2E 31 10 00 03 01 82 01 04 10 00 00 00 19 01 13 01 00 00 00 00 00 00 00 01 0C 74 79 70 65 5F
65 78 74 5F 30 30 31 10 00 00 01 14 12 48 55 52 52 49 43 41 4E 45 2D 46 4F 52 45 49 47 4E 43 01 10
00 00 02 15 14 73 65 72 76 69 63 65 5F 70 72 6F 76 69 64 65 72 5F 30 30 31 10 00 00 03 02 42 42 10
00 01 00 0E 0D 6E 65 74 77 6F 72 6B 5F 69 64 5F 30 31 10 00 01 01 0E 0D 6E 65 74 5F 61 75 78 5F
69 64 5F 30 31 10 00 01 03 0A 01 00 00 00 3C 00 01 45 55 52 10 00 01 05 06 00 00 00 00 00 00 10 00
01 06 04 00 00 00 01 10 00 01 07 03 42 42 01 10 00 01 08 05 01 00 00 00 01 10 00 01 09 13 00 00 00 02
01 00 01 09 31 32 37 2E 30 2E 30 2E 31 00 00 10 00 01 0A 04 00 00 02 00 10 00 01 0B 01 01 10 00 01
0C 02 00 02 10 00 01 0D 05 00 01 01 01 01 10 00 01 0E 11 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 10 00 01 0F 01 01 10 00 03 02 78 10 00 02 00 03 03 01 31 10 00 02 01 12 00 00 42 42 0D 63
69 76 69 63 5F 61 64 64 72 5F 30 31 10 00 02 02 08 00 00 00 01 00 00 00 01 10 00 02 03 25 01 13 01
00 00 00 00 00 00 00 00 01 0A 74 79 70 65 5F 65 78 74 5F 31 04 0B 33 47 50 50 5F 41 44 44 52 5F 31
00 10 00 02 04 0C 08 00 01 08 31 30 2E 30 2E 30 2E 30 10 00 02 05 0C 00 01 09 31 32 37 2E 30 2E 30
2E 31

21:11:47.924 DEBUG ListOfNetworkEncoder - Returning encoded frame:
CompositeChannelBuffer(ridx=0, widx=842, cap=842)
```

Looking at the timestamps in the logs, one can conclude that it took proximately one second and twenty centesimal seconds from the time the Information Server receives the query, queries the database, receives the results, encodes a response message and sends it to the MIHF. The MIHF then sends the message back to the requester MIH User, which we will detail next.

Table 4. MIH User snippet code to calculate and output the cheapest network based on the response message from the Information Server

```
// Declare objects to decode into

odtone::mih::status st;

odtone::mih::ir_bin_data_list ir_data_list;

odtone::mih::ie_container_list_of_networks l;

// Decode message into objects previously declared objects

msg >> odtone::mih::confirm()

& odtone::mih::tlv_status(st)

& odtone::mih::tlv_info_resp_bin_data_list(ir_data_list);

odtone::mih::ir_bin_data data = ir_data_list[0];

data.input() & odtone::mih::tlv_ie_container_list_of_networks(l);

// output queries results

for (uint i = 0; i < l.size(); i++) {

log_(0, "Network Identifier: ", l[i].operator_id.opname);

odtone::mih::cntry_code &country = boost::get<odtone::mih::cntry_code>(l[i].country_code);

log_(0, "Country Code: ", country.val[0], country.val[1]);

odtone::mih::cost &cost = boost::get<odtone::mih::cost>(l[i].cost);

log_(0, "Cost: ", cost.value.integer, ".", cost.value.fraction,
```

Table 4. (*continued*)

```
cost.curr.val[0], cost.curr.val[1], cost.curr.val[2]);

}

// Select the cheapest network

float lower_cost = 0;

odtone::mih::ie_container_network cheapest_net;

BOOST_FOREACH(odtone::mih::ie_container_network& net, l)

{

// get current network cost

odtone::mih::cost &cost = boost::get<odtone::mih::cost>(net.cost);

float fcost = cost.value.integer + ((float)cost.value.fraction / 100);

log_(0, "Lower cost so far: ", lower_cost);

log_(0, "Current cost being analyzed: ", fcost);

if ((lower_cost == 0) || (fcost < lower_cost)){ // if ZERO then first time

// set new lower value

lower_cost = fcost;

log_(0, "New lower cost: ", lower_cost);

// set new cheapest network

cheapest_net = net;

log_(0, "New cheapest network: ", cheapest_net.operator_id.opname);

}
```

Table 4. (*continued*)

```
}

// output result

log_(0, "\n[CHEAPEST NETWORK INFO]");

log_(0, "Network Identifier: ", cheapest_net.operator_id.opname);

odtone::mih::cntry_code &country = boost::get<odtone::mih::cntry_code>(cheapest_net.country_code);

log_(0, "Country Code: ", country.val[0], country.val[1]);

odtone::mih::cost &cost = boost::get<odtone::mih::cost>(cheapest_net.cost);

log_(0, "Cost: ", cost.value.integer, ".", cost.value.fraction,

cost.curr.val[0], cost.curr.val[1], cost.curr.val[2]);
```

And finally, the log.

Table 5. MIH User outputting the cheapest network determination and post information

```
MIH-User has received a MIH_Get_Information.confirm with status 0 and the following results:2

Lower cost so far: 0

Current cost being analyzed: 60.01

New lower cost: 60.01

New cheapest network: HURRICANE-FOREIGNC

Lower cost so far: 60.01

Current cost being analyzed: 50.0

New lower cost: 50.0

New cheapest network: HURRICANE-FOREIGNB

[CHEAPEST NETWORK INFO]
```

Table 5. (*continued*)

Network Identifier: HURRICANE-FOREIGNB
Country Code: PT
Cost: 50.0EUR

Even though this is a very simple query, much more complex queries are supported. But for demonstration and example purposes we have follow the simplest query we managed to find.

On a last note, the **Information Server** also supports transport over TCP/UDP and IPv4/IPv6.

Following the MIHF, the Information Server also supports transport over TCP or UDP over IPv4 or IPv6.

4 Conclusion and Future Work

The referred MIHF implementation has a strong focus on the network facilities that IEEE 802.21 is meant to deploy and support, namely the **Information Server** module.

We reached for maximum openness, portability and easy integration for others to use our implementation, develop their own MIH Users, or simply reuse some of the facilities provided. Also, by reusing open-source libraries, extending and open-source project, and providing installable packages (for Linux only) with a basic configuration, we are targeting a fast and simple adoption of this product.

On what concerns the Information Server, we provide a package that will run on any platform with Java 1.5 (SUN JRE is recommended). It's pretty straightforward to get it to run with the default dummy scenario, but we also provide a programmable way of getting custom-made scenarios.

In the future, we will be targeting the following:

- Implement a way of supporting transport on TCP and UDP over IPv4 and IPv6, simultaneously (dual-stack). This will apply both to the MIHF and the Information Server.
- Specify and implement a dynamic way of populating the Information Server database with real network information. Eventually, this will be provided through a mix of configuration (i.e. graphical user interface) and link-layer monitoring.
- ODTONE is working towards getting support for the Android platform. We will pick this task too.
- Implement more Radio Access Technologies support to LINK SAP, like IEEE 802.16 (WiMAX)

Acknowledgement. The presented work in this paper is supported by the project INFSO-ICT-216006 HURRICANE (Handovers for Ubiquitous and optimal bRoadband connectIvity among CooperAtive Networking Environments), which is partly funded by the EU.

References

1. `http://www.ieee802.org/21/`
2. `http://hng.av.it.pt`
3. `http://hng.av.it.pt/projects/odtone`
4. `http://redmine.pdmfc.com/hurricane`
5. `http://boost.org`
6. `http://java.sun.com/developer/technicalArticles/J2EE/jpa/`
7. `http://www.oracle.com/technology/products/ias/toplink/jpa/index.html`
8. `http://www.hibernate.org`
9. `http://www.h2database.com/`
10. `http://code.google.com/p/openj21`