Enabling the business-based
Internet of Things and Services

(FP7 257852)

# D8.1 ebbits network architecture

**Published by the ebbits Consortium**

**Dissemination Level: Public**

European Commission
Information Society and Media

**Project co-funded by the European Commission within the 7th Framework Programme**
**Objective ICT-2009.1.3: Internet of Things and Enterprise environments**

# Document control page

**Document file:** D8 1_ebbits_networks_architecture_1 1
**Document version:** 1.1
**Document owner:** CNet

**Work package:** WP8 – Physical World Sensors and Networks
**Task**: T8.1 – Network architecture and design
**Deliverable type:** R

**Document status:** ☒ approved by the document owner for internal review
☒ approved for submission to the EC

**Document history:**

| Version | Author(s) | Date | Summary of changes made |
|---|---|---|---|
| 0.1 | Peeter Kool, Matts Ahlsén (CNet) | 2011-05-09 | Initial ToC |
| 0.3 | Gonzalo Alcaraz, Mirko Franceschinis, Hussein khaleel, Claudio Pastrone (ISMB); Mark Vinkovits(FIT) | 2011-07-06 | First draft contribution - Chapter 5 |
| 0.4 | Gonzalo Alcaraz, Mirko Franceschinis, Claudio Pastrone (ISMB) | 2011-08-05 | Second draft contribution – Chapter 5 Update, Section 3.3.1 |
| 0.6 | Peeter Kool (CNet), Pietro Cultrona (Comau), Michael Jacobsen (TNM), Karul Furdik (IS) | 2011-08-12 | Added contribution from IS, TNM , COMAU |
| 0.7 | Peeter Kool, Matts Ahlsén (CNet) | 2011-08-15 | Update of overall network architecture. |
| 0.8 | Peeter Kool, Matts Ahlsén (CNet) | 2011-08-23 | Update of event management and network management. |
| 0.9 | Peeter Kool, Matts Ahlsén (CNet) | 2011-08-25 | Layout and spell checking. Version for internal review. |
| 0.99 | Peter Rosengren (CNet) | 2011-08-30 | Update and final editing after peer review. |
| 1.0 | Peeter Kool, Matts Ahlsén (CNet) | 2011-08-31 | Version submitted to the EU. |
| 1.1 | Peeter Kool, Matts Ahlsén (CNet), Gonzalo Alcaraz, Claudio Pastrone (ISMB) | 2011-11-17 | Revised version resubmitted to the European Commission. |

**Internal review history:**

| Reviewed by | Date | Summary of comments |
|---|---|---|
| Peter Kostelnik (TUK) | 2011-08-26 | Approved with comments |
| Alexander Schneider (FIT) | 2011-08-26 | Approved with comments |

Index:

# 1. Executive summary

The ebbits network architecture defines the network capabilities that allow the ebbits platform to connect the world of physical devices, sensors and actuators with an open SOA-based communications infrastructure, supporting the integration of the Internet of Things with ERP systems.

The deliverable describes how we can reuse the networking part of the base line technology from the Hydra project, i.e. the LinkSmart middleware. But in order to adapt LinkSmart to the needs of ebbits we see that we need to add functionality in the following areas:

*Opportunistic networking*: We need to be able to cope with more unreliable environments where communications will at times fail. This is fundamental in the ebbits platform because we are dealing both with harsh communication environments as well as traceability so there is need to guarantee the delivery of events and data.

*Id management and routing*: In ebbits we need to handle huge amounts of devices and sites which makes it impossible to use the Hydra approach of synchronising all IDs over the P2P network. The security aspect of the routing and addressing mechanism needs to be addressed as well. This will be done by extending the LinkSmart network manager and by creating a Border Network Manager which is responsible of routing in-between different ebbits networks.

*Wireless sensor networks*: We need to extend the device support in LinkSmart to cover WSN based networks to deal with the specific problems of those, such as latency, memory- and computational constraints. The deliverable discusses various aspects of using IpV6 and 6LowPAN and how to integrate with operating systems for WSN, such as Contiki and TinyOS.

The most important properties of the network architecture defined in this document are:

- Transparency for the developer using SOAP-tunnelling and P2P techniques.
- Scalability by using context hierarchies that make subnetting possible.
- Integration of Wireless Sensor Networks.
- Resilience and reliability using opportunistic networking.
- Provisions for adding security at different levels.

# 2. Introduction

## 2.1 Purpose, context and scope of this deliverable

This deliverable defines the initial network architecture for the ebbits system. This work has been primarily done within Work Package 8, Task 8.1.

The network architecture covers the whole scope from wireless sensor networks to P2P networking. It includes addressing and routing of communications as well as resilience and reliability.

This deliverable is primarily divided in to five sections:

- Application Domains Communication:
  Describes the current communication in the domains.
- P2P Networking:
  Discusses different issues, including Wireless Sensor Network support, with regards to the baseline P2P architecture.
- Opportunistic networking:
  Discusses and defines an architecture for enabling opportunistic networking from several viewpoints, such as frequency agility.
- Overall network architecture:
  Defines the network architecture and the necessary extensions to be made to the baseline technology with regards to the requirements of ebbits.
- Conclusions and future plans:
  Summarizes the important points from the previous chapters and also includes what the future plans are.

# 3. Application Domains Communication

## 3.1 Review of Manufacturing Application

In the manufacturing scenario there are many data flows between the equipment involved in the production process. This section aims to summarize the current and the foreseen protocols and technologies used by these devices.

### 3.1.1 Main constraints and problems

As already described in Deliverable D8.4 "Integration of physical world in manufacturing", the electronic equipment can be connected through wires or in a wireless connection. Each connection must face to several constraints and problems in order to be efficiently applied on the field.

In particular for the wired networks the **routing in hostile environment** is the most important aspect to be considered. Sometimes there is not enough space for the routing of the cables, or it is very difficult to perform an efficient maintenance in term of costs. In some particular conditions **specific cables** are required. They are resistant to oils, to chemical, to high temperature and/or extra flexible, or with a stronger insulation.

When talking of wireless networks it is important to consider the **possibility of interference** due to the coexistence of more than one network. Or more than one device working in the same frequency range can cause possible disrupts in the transmission among the network.

The table below summarizes the most used mobile technologies and their operating frequency bands:

| Mobile Technology | Frequency Band |
|---|---|
| GSM Dualband EU | 960 MHz, 1,8 GHz |
| GSM Dualband USA | 850 MHz, 1,9 GHz |
| UMTS - HSDPA | 1,9 GHz, 2.1 GHz |
| Bluetooth | 2,45 GHz |
| Wi-Fi | 2,4 GHz,  5,4 GHz |
| Cordless phone | 2,4 GHz – 5,8 GHz |

Table 1: Mobile technologies and frequencies

Other interferences can be caused by the **EMC pollution**. It is another aspect that recently is increasing the risks of communication problems. This is due to the high quantity of welding guns and motors distributed on the field, which added to the increasing number of wireless devices, generates an high EMC field. The long cables used to interconnect the electrical equipment could sometimes act as transformers and generate high voltage shocks damaging the devices connected.

### 3.1.2 Communication technologies and protocols in manufacturing

The communication technologies and protocols actually used and foreseen inside the manufacturing plants have been widely described inside the Deliverable D5.1.1 "Concept and Technologies in Intelligent Service structures", but they can be summarized as follows:

In the main family of the field buses used inside the manufacturing plants are CANOpen, Devicenet, Ethernet/IP, Interbus, Modbus, Profibus, and Profinet.

**CANOpen[1]:** a specification for distributed industrial automation systems based on Controller Area Network (CAN).

**Devicenet:** originally developed by Allen-Bradley, it adapts the technology of ControlNet with the advantages of CAN, creating a low-cost and robust network if compared with the traditional RS-485 based protocols.

**Ethernet/IP:** it is the natural evolution of the Devicenet protocol on the Ethernet network infrastructure.

**Interbus:** it was developed by Phoenix Contact. It is based on a ring network with a master and many slaves. It is full duplex and the maximum bus length is of 400 m.

**MODBUS TCP[2]:** is based on the MODBUS family. It was created in order to provide the possibility to send MODBUS messages over an Internet environment utilizing TCP/IP.

**Profibus:** it stands for Process Field Bus. It was developed by Siemens and it is a simple communication monomaster multislave bus. The communication is serial and data is exchanged through "token ring".

**Profinet:** it is a Profibus Ethernet-based, similarly to the evolution done by Allen-Bradley from Devicenet to Ethernet/IP.

During the production process, the PLC does not only communicate with the devices placed on the field through field bus, but it also communicates with higher monitoring levels like HMIs and SCADAs systems installed on an Industrial PC. In order to establish the communication with these devices a specific protocol must be used. The most common and widely used is the **OPC protocol**. As already described in the D8.4, its main aim is to ensure the communication of real-time data between control devices coming from different suppliers, in order to provide a common bridge for Windows based software applications and process control hardware.

Moreover, the use of other standards is being considered.

**IEEE 1451[3]:** a family of Smart Transducer Interface standards developed to provide an interface between transducers (sensors, event sensors and actuators) and microprocessors, instrumentation systems, and control networks.

**Zigbee:** a wireless technology developed in order to address the needs of a low-cost and low-power wireless network. It operates on the IEE 802.15.4 radio specification in unlicensed bands including 868 MHz, 900 MHz and 2.4 GHz. The main advantages of this kind of connection are a low duty cycle, a low latency time, a 128-bit AES encryption, up to 65000 nodes per network.

**Bluetooth:** it is another wireless technology used for the data exchanging over short ranges of distances. Like for ZigBee, also its radio transmission takes place in the 2400 MHz and 2480 MHz.

### 3.1.3 Foreseen communication protocols and technologies

In the manufacturing environment and in general in the industrial field, the automation professionals are very conservative and cautious to adopt new standards and technologies, unless a low risk and concrete benefits have been identified. However in the last years a big step has been done with the introduction of the Ethernet standards. The direction took by the main suppliers of field devices highlights this aspect with the massive use of Profinet and Ethernet/IP. The next steps should be of course a slow abandonment of the "Copper-Wired" technology in order to replace it with optical fiber connections and a wireless technology that ensures more flexibility with a better standardization of the communication protocols and the birth of new open source protocols.

[1] CANopen, http://www.can-cia.org/index.php?id=47
[2] MODBUS TCP, http://www.modbus.org/
[3] NIST IEEE-P1451, http://ieee1451.nist.gov/

## 3.2 Review of Agricultural Application

As described earlier (D8.2[4], D10.1[5]) there are no widely used standards in the agricultural domain. In food processing, transportation and the retail business the companies primarily use ERP systems like SAP or Microsoft Dynamics NAV to collect information. The information in these systems can be accessed by ebbits middleware via various techniques provided by the ERP systems.

The farmer does typically not have a large scale ERP system but uses a Farm Management System (FMS). The FMS can be coupled with some of his other sub systems, but usually the subsystems are accessed directly. A typical farm has, like any other company, an Ethernet bus with internet access and possibly a Wi-Fi network. The different control systems on the farm usually have their own bus to communicate on. Figure 1 describes a typical farm installation.

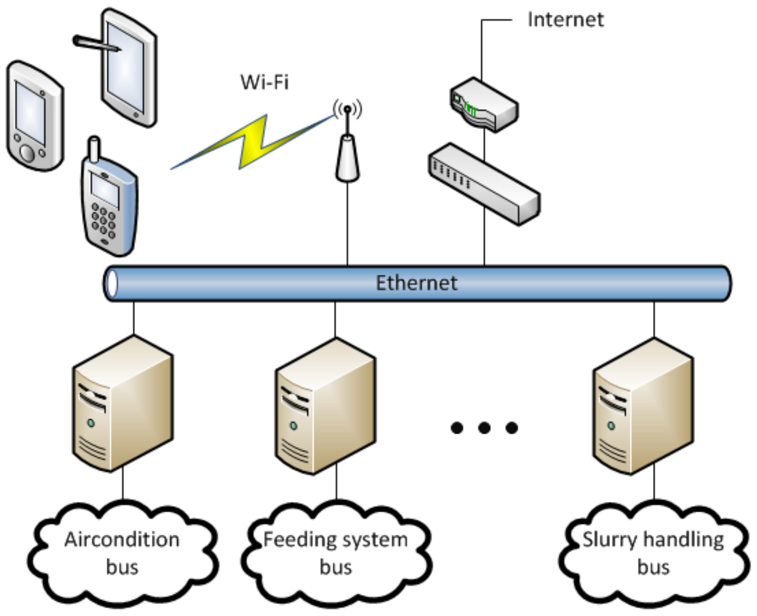


Figure 1 – Network structure on a typical farm

Subsystems often operate isolated from other systems and therefore have local control panels for the operator to change settings and parameters and their own field networks. That means that they are not always connected with the Ethernet on the farm. There are many types of field networks used in the agricultural domain. Some are proprietary but other manufacturers of systems use international standards. Some examples of buses used in these sub systems are:

- **ISO 11783**[6] – Bus system used on tractors and implements to optimize performance and ease control. This is based on CAN-bus system used in cars. More info see (Fellmeth 2003) and (Stone et al 1999).
- **P-Net**[7] – Industrial automation and process control bus.
- **EtherCat**[8] – Industrial automation and process control bus.
- **ISO Agrinet**[9] – Danish standard for farm data exchange based on ISO 11788.

[4] Deliverable 8.2 – A Survey of Physical World in manufacturing and traceability scenarios.
[5] Deliverable 10.1 – Description of communication networks and common components.
[6] http://isobus.net/
[7] http://www.p-net.org/
[8] http://www.ethercat.org/
[9] http://datastandard.dk/

Although not all subsystems are connected to the Ethernet they are usually capable of providing data exchange services. That is usually implemented with a PC connected to the bus running a server/proxy to access the data. Examples of data exchange standards are OPC Data Access[10] standard which is based on OLE, COM and DCOM technologies and VIGO[11] which uses COM objects to access data on the P-Net bus.

[10] http://www.opcfoundation.org/
[11] http://www.proces-data.com/

# 4. P2P Networking

## 4.1 P2P in ebbits

The ebbits P2P network is based on the P2P network from the baseline Hydra project and uses the same model. The Network Manager is the component responsible of creating and maintaining the P2P network. The main objective of the Network Manager is to interconnect different ebbits devices and services through the network. The main problem of this task is that most of the devices and services may be hidden in Local Area Networks, behind firewalls, routers and Network Addressing Translators (NATs), so it would be difficult to interconnect directly.

However, the Network Manager solves this problem by building an overlay network, independently of the network addressing and protocols.

The Network Manager relies on JXTA P2P[12] platform in order to build the overlay network. JXTA is a set of open, generalised P2P protocols enabling any connected device on the network to communicate and collaborate. Using the JXTA protocols, devices and services are directly connected even if they are connected in different networks separated by firewalls or NATs.

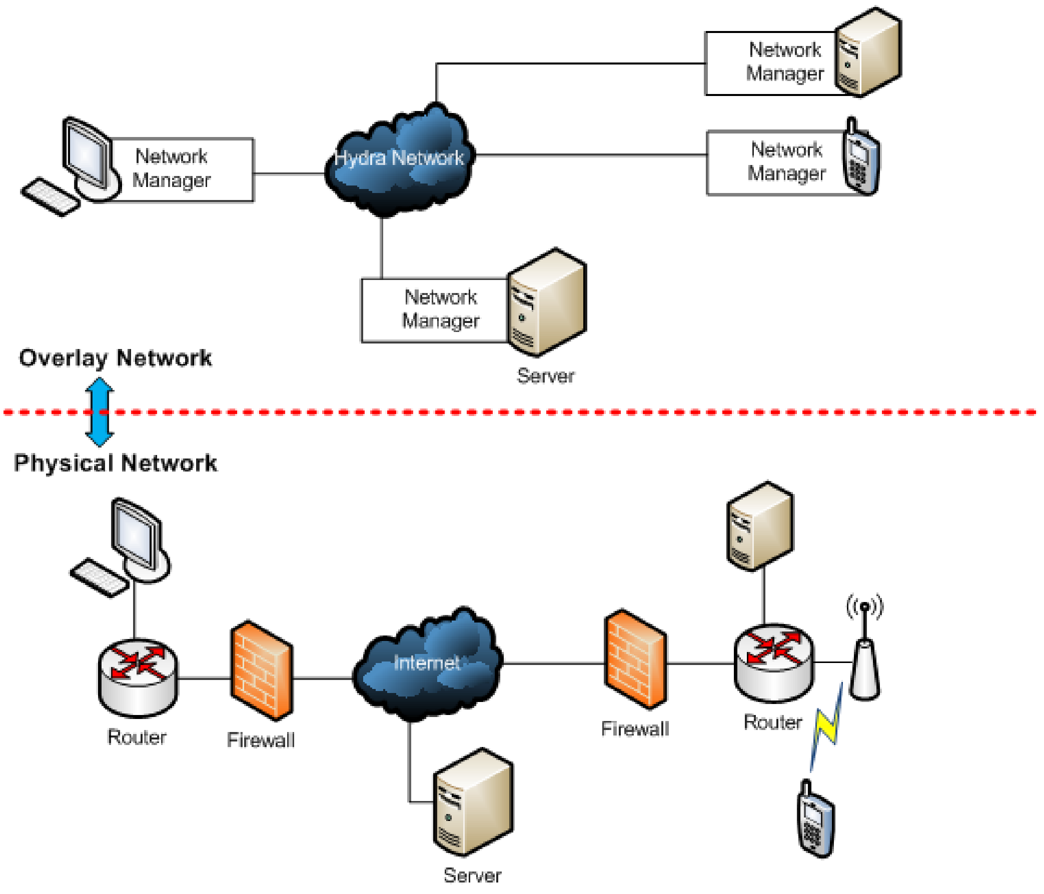


Figure 2: Overlay Network

Figure 2 shows an example of how the different devices and services are interconnected in the ebbits overlay network and how the actual network could look like.

In order to make services and devices available on the P2P network they need to register their services (i.e. endpoints) with the network manager. The network creates a unique ID for the service, called HID, which is then used for addressing the service.

[12] http://en.wikipedia.org/wiki/JXTA

The HIDs are shared and synchronized among the network managers in the network. In effect all network managers know which HIDs are available in the network. This table of HIDs is referred to as the ID table, see Table 1:

| HID | endpoint | Description | Peer ID |
|---|---|---|---|
| **223.122.33.33** | http://127.0.0.1:8093/svc | Thermometer | 1 |
| **223.888.1.33** | | Thermometer | 2 |
| **223.877.33.22** | | OntologyManager | 2 |

Table 1 Example of the Network Manager ID table

The ID table contains the following data:

- HID: That address that is used for the service
- Endpoint: The actual endpoint of the service (in fact this field is not synchronised in-between network managers, for security and performance reasons). So the endpoint is only known to the Network Manager were the service registered. In the example above only the services belonging to Peer ID=1 are known
- Description: an optional field where a simple description of the service can be stored.
- Peer ID: The ID of the network manager that manages the service.

### 4.1.1 SOAP Tunnelling Approach for Device Communication and Service invocation

As the Hydra, and ebbits, architecture is service-oriented, where web-services (WS) is the technology used to implement it, the communication between applications running in different Hydra-enabled devices will be based on SOAP messages. Usually, SOAP messages are forwarded through TCP connections to the destination. The destination address corresponds to the endpoint contained in the message.

Traditional WS architectures are based on client-server architectures, where the server is an always-on end system with a well known endpoint address, which should be known by clients beforehand (using either service descriptors or UDDI registries). The SOAP tunneling approach proposes a way to replace this client-server architecture for a distributed one, using the Network Manager P2P platform (Lardies 2009). In this architecture, all the peers will act as clients and servers at the same time. Figure 3 shows an example of a client-server based architecture and the distributed approach.

Furthermore, actual WS communications require direct connection between the client and the server, making it impossible to consume services across networks.

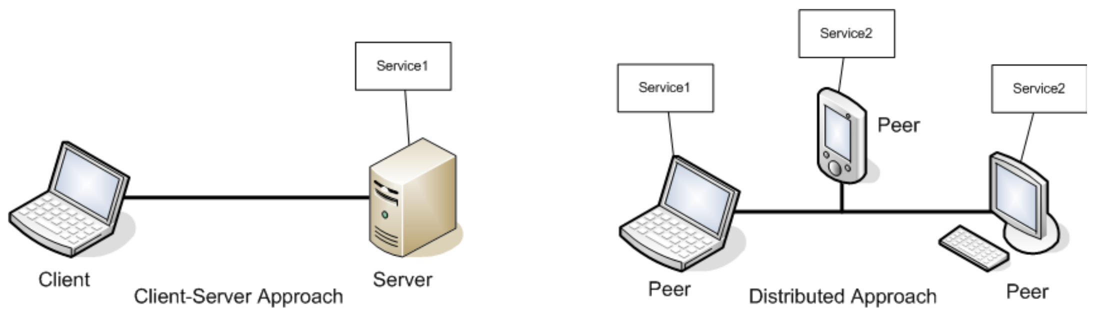


Figure 3: Client-Server vs. Peer-peer

Moreover, in the Hydra middleware, devices are presented as UPnP devices by the Device Manager. But UPnP discovery information is usually restricted to Local Area Networks. Using the SOAP tunneling the Device Manager will be able to exchange the UPnP information between different Discovery Managers in the Hydra Network. Thus other Device Managers will be able to control UPnP devices located in remote networks using the SOAP technique presented in this section.

Therefore the main objective of the SOAP tunneling approach is to enable SOAP messages exchange across different networks, making it possible to consume services provided by different Hydra Enabled devices/applications or controlling UPnP devices located in different Local Area Networks**.** Figure 4 shows an example of the application of SOAP tunneling. Thanks to the Network Manager and the SOAP tunneling approach, HED2 is able to discover UPnP devices located at home network (weigh scale and thermometer) and to consume the web services offered by the application running on the HED1.

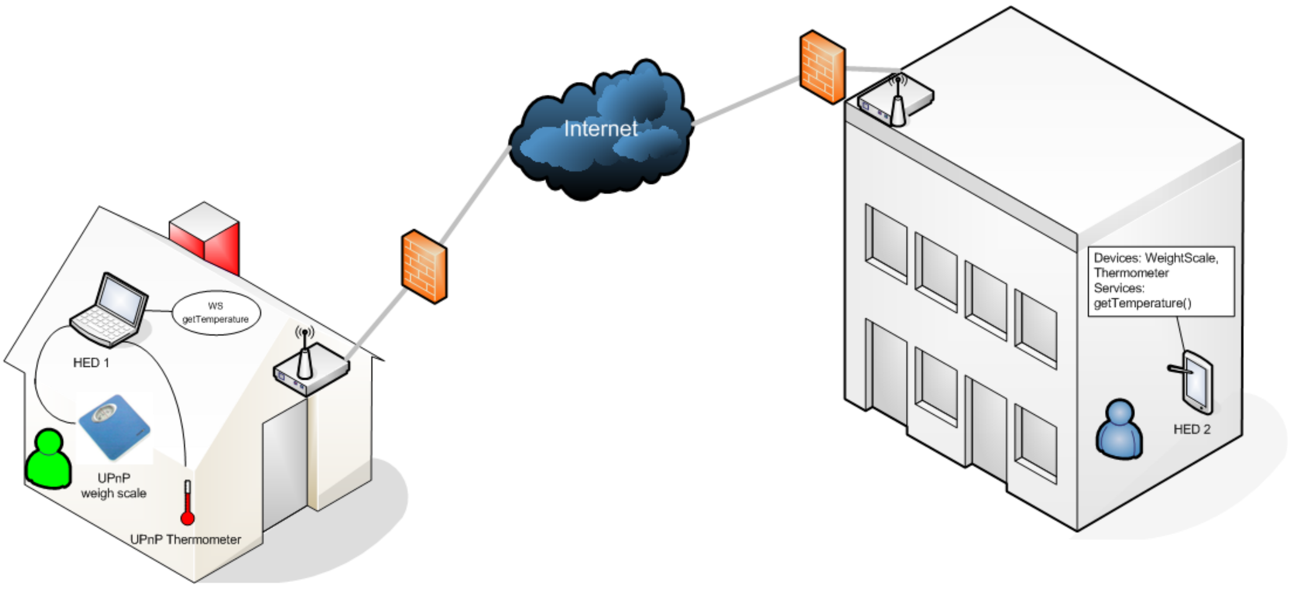


Figure 4: SOAP tunnelling example

### 4.1.2 SOAP Tunnelling

Thus, the Network Manager enables a way to communicate with different devices transparently, building an overlay network in which resources (devices, services and contents) can be addressed. The main objective of the SOAP tunneling communication used in the Hydra project is to provide SOAP messages exchange using the P2P transport schemes provided by the Network Manager.

In order to use P2P networking/addressing/transport schemes together with web services and UPnP we need some kind of virtualization of endpoints that allow us to use P2P networking. For this reason, all endpoints for UPnP and web service calls are grounded in a SOAP sink (ideally locally) which repackages the SOAP message and routes it through the Network Manager, as shown in Figure 5. The Network Manager is responsible of the message transmission and finally calls the SOAP sink that performs a local SOAP call to the intended SOAP endpoint.

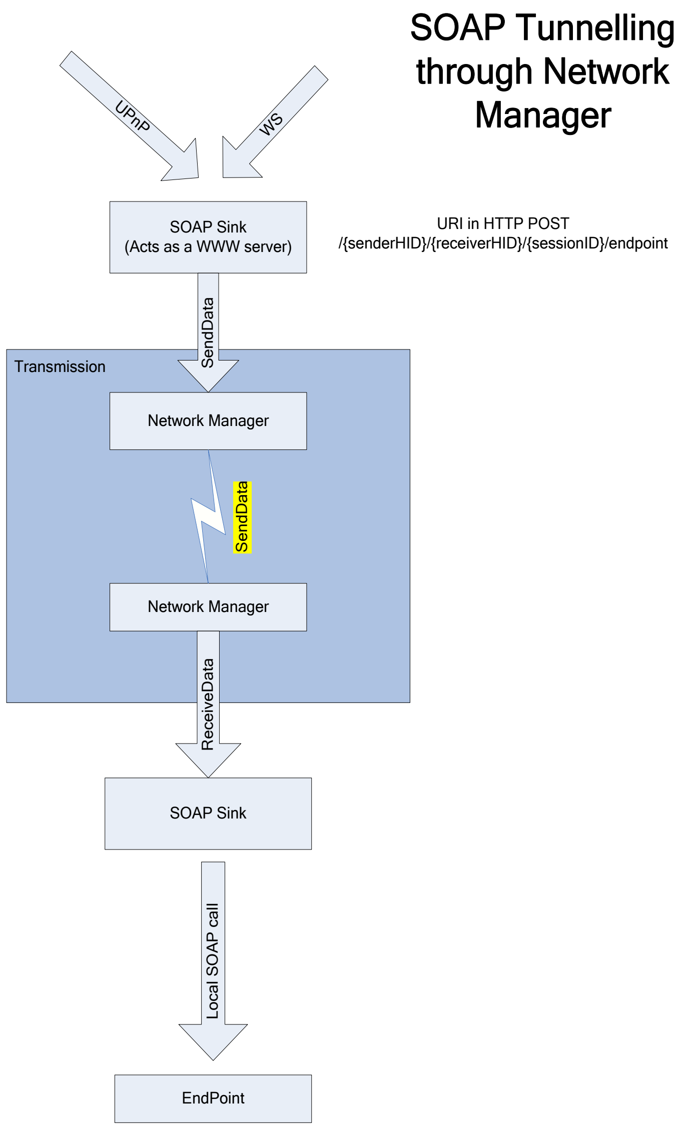

SOAP Tunnelling through Network Manager
UPnP
WS
SOAP Sink (Acts as a WWW server)
URI in HTTP POST /{senderHID}/{receiverHID}/{sessionID}/endpoint
SendData
Transmission
Network Manager
SendData
Network Manager
ReceiveData
SOAP Sink
Local SOAP call
EndPoint

Figure 5: SOAP tunnel

The P2P networking with the SOAP tunneling technique will facilitate event management, as well as SOA in general, in the ebbits architecture.

## 4.2 Eventing in a P2P based network

### 4.2.1 Basis

In general, most event driven architectures include a couple of well known building blocks – event producer, event consumer, event processing agent, event channel, which together with a global state represent an event processing network (Kostelnik et al. 2011) (Etzion et al. 2010). From the communication point of view the most important part is the implementation of the event processing network that connects event producer, consumers and processing agents via event channels. In a decoupled event processing network an event producer does not depend neither on an event processing agent nor on an event consumer. In a similar way, an event consumer does not depend on an event processing agent or on an even producer (with exception of the fact that the event was produced). The ways how an event channel can be implemented include:

- an intermediary service or other piece of software (sometimes called a broker),

- a multicast protocol, such as IP Multicast,
- thru a Message Oriented Middleware (MOM), such as a Java Message Service (JMS) provider, or,
- as part of a generic service oriented architecture (SOA) middleware, such as the LinkSmart middleware from the Hydra project.

The ebbits deployment environment will be highly flexible and dynamic, the dynamic consumer registration (subscription) will be preferred – i.e. the publish/subscribe pattern. The process of interaction between involved parties in publish/subscribe systems can be briefly described as follows. Producers and consumers are independent entities that exchange information by publishing events via event channels and by subscribing to the classes of events they are interested in again via event channels. Publishers publish information in the form of events and subscribers express their interests in an event or a pattern of events in the form of subscription filters. A data event specifies values of a set of attributes associated with the event. The subscriptions can be very expressive and specify complex filtering criteria by using a set of predicates over event attributes. When an event channel receives an event published by a publisher, it matches the event to the subscriptions and delivers the event to the matched subscribers. A subscriber installs and removes a subscription from the channel by executing the subscribing and unsubscribing operations respectively. The publish/subscribe systems can be divided according to the following three criteria (Shen 2010):

- Expressive power of subscription models – topic-based, content-based and type-based.
- Routing solution of the notification service – filter-based approaches and multicast-based approaches.
- System topology – centralized and distributed, whereby the distributed can be further divided into broker-based and Distributed Hash Table (DHT)-based systems that belongs to the structured peer-to-peer (P2P) systems.

For our purposes the most important type are content-based systems with application of filter-based approaches, whereby distributed topology based on P2P network is used. To the advantages of such solution belongs fine-grained expressiveness of subscription, improved matching between subscriptions and events and more efficiently routing of the matched events to the destinations. DHT-based publish/subscribe systems inherit advantages like scalability, efficiency, reliability, fault-tolerance, self-organizing from the underlying DHT overlay network infrastructure.

### 4.2.2 P2P Eventing in ebbits

The initial ebbits event mechanism is thus based on a topic/content based, publish-and-subscribe architecture. In future iterations we will consider using other more complex event processing engines, such as Esper (Esper, 2010). The ebbits event manager is based on an adaptation of the Network and Event Managers from the Hydra project, as described in previous deliverables (Ahlsen et al. 2011). The ebbits Event Manager is deployed in an ebbits architecture as a service in close cooperation with other components, among them the Network Manager, providing publish/subscribe functionality, i.e., the ability for publishers to send a notification to multiple subscribers while being decoupled from them (as described in the previous paragraph).

The architecture of ebbits has been characterized as *event-driven SOA,* integrating intelligent services with advanced semantic event processing and business rules, and the platform thus relies on the secure delivery of events (Ahlsen et al. 2011). The basic event manager is being extended in several ways in order to support reliable eventing. Among the extension are,

- Delay tolerance. In the event (!) of communication failures, the system must still prevent loss of events. Store and forward functionality should be provided to guarantee delivery. This will make use of the opportunistic networking features such as the delay tolerance (see 5.3).
- Storage. Delay tolerance by means of store and forward implies that the event manager must have storage available. Local caching will be combined with event databases accessible by all networked nodes that process events. An event database will also be used by the

functions above the network layer, such as the business rule engine for evaluating rules which may depend on previous events.

- Time synchronisation. Time is of essence when processing events, there is a need to synchronise time in the distributed ebbits system architecture because the business rules can express time dependence in the rules. Even though delay tolerance will guarantee delivery of events, it does not guarantee the order of arrival. There is a need to time stamp events in order to be able to order the events in the correct time sequence at the ebbits central node. Since the network might bridge firewalls etc it might not possible to use a NTP[13] server for this. Therefore the Network Manager part of the event management architecture should be extended with functionality to synchronise time in-between the different nodes in the ebbits network.
- Stateful event processing and persistency. The processing of an event may be dependent upon one or several other events. It must be possible to maintain an event history (log) and to have access to any results or side-effects of the events occurred. The latter is supported by provision of persistent storage on the event processing nodes in the architecture.

## 4.3 WSN and P2P

### 4.3.1 WSN approach

The aim for the integration of Wireless sensor networks (WSN) in the P2P platform is to make all sensors addressable and usable in the P2P network of ebbits. This will enable the application developer to use WSN devices as any other device on the P2P network.

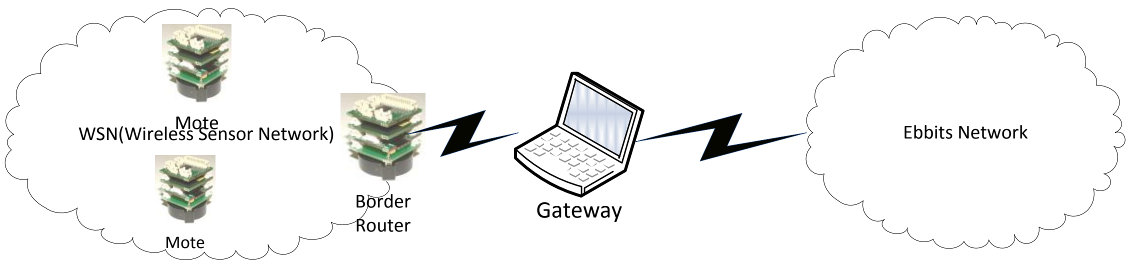


Figure 6: Typical WSN network configuration

Typically the interface to a WSN is done using a so called border router which acts as a bridge in-between the normal network and the WSN and which knows of all the motes on the WSN. Usually this border router is connected to a gateway, for instance a PC, which runs software that interacts with the WSN, see Figure 6. The sensors themselves are usually attached to so called motes that provide the communication platform and limited computational power.

Integration of P2P technologies together with wireless sensor networks (WSNs) poses some unique challenges:

- Routing in-between address spaces
- Often sensors report values at intervals instead for being queried interactively because of energy (i.e. battery) constraints.
- Depending on the communication and network topology calls may take very long to finish.
- The amount of memory and computational power in the nodes is very limited.
- Sensors which are mobile can move in-between different border routers.

[13] Network Time Protocol http://www.ntp.org

There are basically two approaches that can be adopted for integrating the WSN in to the ebbits network:

- *Common address space*: Since most WSNs today use IPv6 based protocols one could make the actual sensor addresses available in the ebbits network using IPv6 addressing capabilities.
- *Proxies*: The use of proxy objects that embed the WSN functionality and act as an interface in-between the ebbits network and the WSN.

Of these two options we believe that proxies are generally the best solution because of:

- The proxy can cache the latest received value. This will enable programs to poll the value without requiring any WSN communication.
- The proxy can carry more metadata about the device, i.e. the amount of metadata is not limited to devices memory.
- The proxies can have the services independent of the underlying WSN protocol. For instance a ZigBee based thermometer can have the same services as a 6LowPAN thermometer.
- Errors in the WSN network can be handled by the proxy itself.

One problem that this solution does not solve is managing the addressing of sensors that move in-between different border routers, i.e. identifying that a mote that is discovered is the same mote that was previously controlled by another border router. This identification needs to be resolved at a higher level such as using the device ontology.

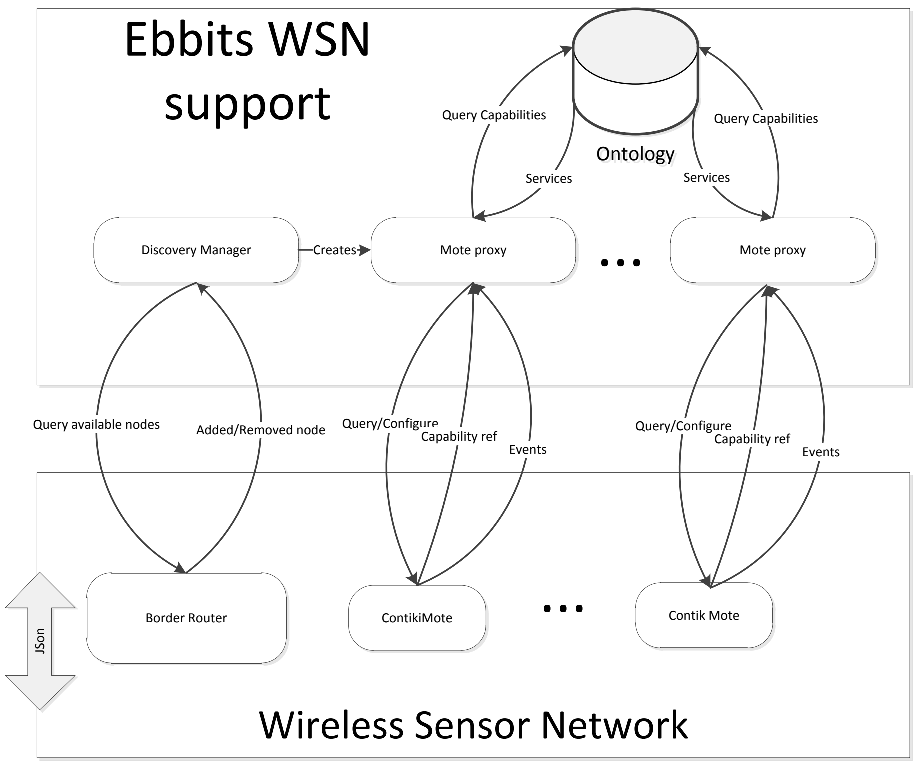


Figure 7: ebbits integration of WSN using proxies.

The above figure shows how the WSN will be integrated using proxies in ebbits. This example will refer to the Contiki (Contiki 2010) Operating System platform running 6LowPAN, but the principles will be the same for other WSN Operating systems such as TinyOS. In runtime the process of discovering devices and creating proxies will follow these steps:

1. A Contiki discovery manager is started on the gateway.
2. The discovery manager will query the border router of available motes.
3. For each mote the discovery manager will create a "mote proxy". The mote proxy will query which capabilities the motes has (i.e. which sensors/actuators are attached). In the case of Contiki this is just a set of identifiers.
4. The mote proxy will query the device ontology using the capability information received to determine which services (sensors/actuators) are connected to this mote.
5. The mote proxy creates the service proxies for these and offers the services to the ebbits network.

Because we have proxies and connections to the device ontology for the proxy services the ebbits system has all necessary metadata, such as unit of measurement etc. available without needing it to be carried on the mote itself.

This is of course a slightly simplified model which leaves out the actual handling of sensor values and configuration of the motes in the network. Configuration of motes usually involves setting the interval of measurements etc.

### 4.3.2 6LoWPAN

The reference IoPTS scenario considered within the ebbits project foresees the evolution from a network of computers to a network of people, smart objects and services. As far as Internet of Things is concerned, a multitude of heterogeneous smart objects, provided with self-configuring capabilities, will be required to interoperate with each other. In this context, the adoption of the Internet Protocol (IP) standard solution could play a key role. In particular, the last available version of the IP protocol, i.e., IPv6 (IEEE 2009), presents expanded addressing capabilities and specific improvements related to security, quality of service, and packet forwarding. Consequently, IPv6 solutions are being increasingly adopted in different low bandwidth wireless communication technologies, particularly suited for the actual realization of the Internet of Things. More specifically, the Internet Engineering Task Force (IETF) standard 6LoWPAN[14] enables the adoption of IPv6 protocol in Low power Wireless Personal Area Networks (LoWPANs) based on standard IEEE 802.15.4-2003

The 6LoWPAN format defines how IPv6 communication is carried in IEEE 802.15.4 frames and specifies the adaptation layer's key elements. 6LoWPAN has three primary elements:

- Header compression: IPv6 header fields are compressed by assuming usage of common values. Header fields are elided from a packet when the adaptation layer can derive them from link-level information carried in the IEEE 802.15.4 frame or based on simple assumptions of shared context;
- Fragmentation: IPv6 packets are fragmented into multiple link-level frames to accommodate the IPv6 minimum MTU requirement;
- Layer-two forwarding: to support layer-two forwarding of IPv6 datagrams, the adaptation layer can carry link-level addresses for the ends of an IP hop.

The key concept, on which adaptation layer is founded to reduce packet size, is to limit at just few bytes adaptation, network and transport layer header fields. This is possible because we observe that header fields often carry common values or that we can deduce them from shared context. Another feature to compress the header fields is to elide redundant information across protocol layers; for instance, IPv6 addresses are derived from lower-layer headers.

[14] http://datatracker.ietf.org/wg/6lowpan/

# 5. Opportunistic networking

## 5.1 Introduction

The integration of heterogeneous communication technologies and physical devices in a unified overall infrastructure governed by the ebbits middleware represents the main purpose pursued in ebbits. The approach adopted allows to define a core system architecture supporting different application scenarios.

The overall ebbits system consists of several components, both physical devices and distributed software elements. The ebbits network architecture, indeed, constitutes a communication overlay infrastructure which allows to interconnect the several managers of the ebbits architecture (e.g., Ontology Manager, Event Managers, etc.) as well as the different and heterogeneous devices belonging to the physical world. Through the ebbits architecture, and software elements in particular, network devices are able to discover services and other nodes, access to resources and information, share contents and so on. Mainly depending on their hardware resources, but also on other internal capabilities, physical world elements can be univocally classified as Powerful, Constrained or Sub-systems. Only those network nodes which are endowed with a Network Manager can be strictly considered ebbits elements. On the contrary, simple peripheral and technology-dependent network nodes can communicate with each other using their native communication technology but they need to pass through other ebbits elements, specifically ebbits Gateways (GW), which are in charge of exposing a homogeneous sub-system to the rest of the ebbits world. In other words, an ebbits GW has been designed to interface generic devices in ebbits, independently of the specific device category.

Definitively, according to ebbits overall description, multiple communication technologies and heterogeneous physical devices are integrated in a larger architecture. Note that for each communication technology, multiple sub-networks can be deployed at the same time and at arbitrary distances. In addition, the physical deployment area of each sub-network and its density are as general as possible. In this scenario, the ebbits network architecture acts as glue integrating the heterogeneous communication technologies and physical world devices. More in details, according to the needs, the integration of specific communication technologies can be performed either by leveraging on IP or by adopting proper ebbits GWs. In the second case, the GW hosts the Network Manager functionalities necessary to enable the interaction between the ebbits domain and any single representative of the sub-network (a constrained device). Moreover, the GW is in charge of managing the sub-network, e.g., running dynamic channel selection to decide if and when to adaptively switch to a different radio channel is under its competence, as it will be described in Section 5.4.1 for 6LoWPANs.

To conclude this short ebbits system communication overview, it is worth adding that it also includes powerful devices, typically mobile nodes endowed with multiple radio interfaces. During their mobile wanderings, such nodes might repeatedly join and leave distinct sub-networks, i.e. technologically different domains or, more simply, homogeneous but physically separate networks, exposed to the external ebbits network by different ebbits GWs. The duration of the connection times to clouds as experienced by mobile powerful devices can result in variable time periods. Sometimes mobile powerful devices are disconnected from any network, in other occasions instead they could communicate by means of one or multiple radio technologies.

## 5.2 Communication flavors

Such a complex scenario characterized by a large number of open dynamics lends itself to several opportunistic strategies that may result in increased overall system efficiency. In particular, making reference to the scenario described above, we identified the following opportunistic communication flavors, which are intended to be developed and included in ebbits:

- Delay Tolerance Networking (DTN) and Multiradio in the core network
- Frequency Agility (FA) and DTN capabilities in WSN/6LoWPAN networks

Ideas and concepts behind opportunism in the mentioned flavors will be described in details respectively in Sections 5.3 and 5.4. The declination of opportunistic communication exploiting multiradio and delay tolerant networking applies to the overall ebbits system. Concerning the ebbits middleware, opportunistic features will be developed and included within the Network Manager. At the same time, *frequency agility* applies only to the WSN/6LoWPAN domain, i.e. just one of the possible radio communication clouds/technologies considered in ebbits. The same holds for 6LoWDTN, an opportunistic communication transport layer proposed in the context of delay tolerant applications in WSN/6LoWPAN networks.

In fact, three managers will be included named Multiradio Manager, Frequency Agility Manager and 6LoWDTN Manager respectively as shown in Figure 8 where the Network Manager global architecture is represented as well.

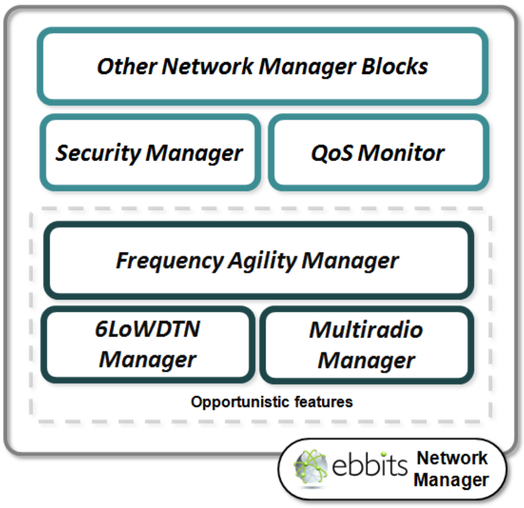


Figure 8 –Opportunistic features included within the network manager

## 5.3 Opportunistic communication in core ebbits network

The adoption of opportunistic communication paradigm allows improving ebbits core network infrastructure characteristics in terms of both availability and reliability in mobility scenarios. Specifically, we refer to the availability of multiple heterogeneous communication technologies and to applications tolerant to delivery delays. We thus propose to enrich ebbits networks by introducing a multiradio manager within the network managers already present in the ebbits architecture. This section is devoted to first introduce the two main pillars, delay tolerant applications and multiradio technologies, and then to present the related architectural modules.

### 5.3.1 Delay tolerance

From the design perspective, each network application can be described and outlined making reference its specific requirements, which are defined in terms of both sensitivity to particular performance metrics and even needs for resources. Typical examples can be the reliability of information transfer, for instance measured through the rate of successful packet transmission, and the information delivery delay. Delay tolerant applications are the term that identifies the set of those applications for which the latencies experienced in the end-to-end data delivery process are not critical. Such applications find a favorable networking context in those scenarios where network nodes are characterized by either frequent radio switch off or mobility patterns such that, jointly and dynamically, cause network topology disconnections and node unavailability. Under such conditions, communication is challenging and can require unforeseeable and unbounded time periods during which information is temporarily memorized in suitable and distributed storing areas and, possibly, carried by mobile nodes.

### 5.3.2 Multiradio

During the last years, the multiradio concept has gained increasing interest. Devices able to adopt multiple wireless technologies are a real fact. Actually, the need for network management features capable of orchestrating in an "opportunistic" way the many radio interfaces is a reality and a challenge at the same time. At the moment, only mobile phones with the goal of keeping up the network connection while moving are capable of performing seamless handovers between 3GPP[15] technologies.

Moreover, the overall quantity of devices equipped with several radio interfaces progressively grows and it is commonly thought that it will continue increasing. Users willing to explore the maximum of their devices will demand to be connected anytime and anywhere. Satisfying this request means that users should be able to move without losing the experience of being "*always connected*" as described in (Gustafsson 2003). In order to provide that experience to the final user, several frameworks and standards regarding seamless handovers (IEEE 2009) (Ferrus et al. 2010) and mobility management (Berggren et al. 2005) (Mansor and Wan 2010) have been developed in recent times by the research community.

It is worth mentioning that within ebbits the implementation of the multiradio concept is not such demanding, considering that the main focus is to implement not seamless mobility but in a certain way a "reliable" mobility scheme. Therefore, ebbits devices should be able to manage their radio interfaces in order to optimally select the appropriate network technology based on certain policies related to applications requirements (e.g., priority, data quantity) and radio context (e.g., network availability, signal quality). In addition, some interaction with store-and-forward features would be of help in situations where devices are out of wireless coverage.

### 5.3.3 Adapted approach for ebbits

In ebbits, the multiradio and delay tolerance approaches are principally intended to be used within the traceability scenario. This scenario includes several use cases sharing the occurring of situations where network connectivity could be intermittent or even not available. In such cases, the multiradio approach will also need store-carry-and-forward features, typical of delay tolerant networks, in order to face the intermittencies of wireless connectivity without losing important information.

As mentioned when introducing the definition of multiradio, with this term we intend that it is fundamental for a multiradio device to be as much as possible "connected" but at the same time to deliver information with a certain reliability. In addition, it must be taken into consideration that the data included may have time constraints considering that it could be no longer useful after some time. In such case, a mechanism should validate the data before any transmission, which will then possibly and accordingly follow.

All emerging issues regarding working with several network interfaces will be managed by the multiradio manager included within the network manager previously introduced. The following diagram shows the basic steps, that devices should follow in any circumstance when the need to send some type of information rises.

[15] http://www.3gpp.org/

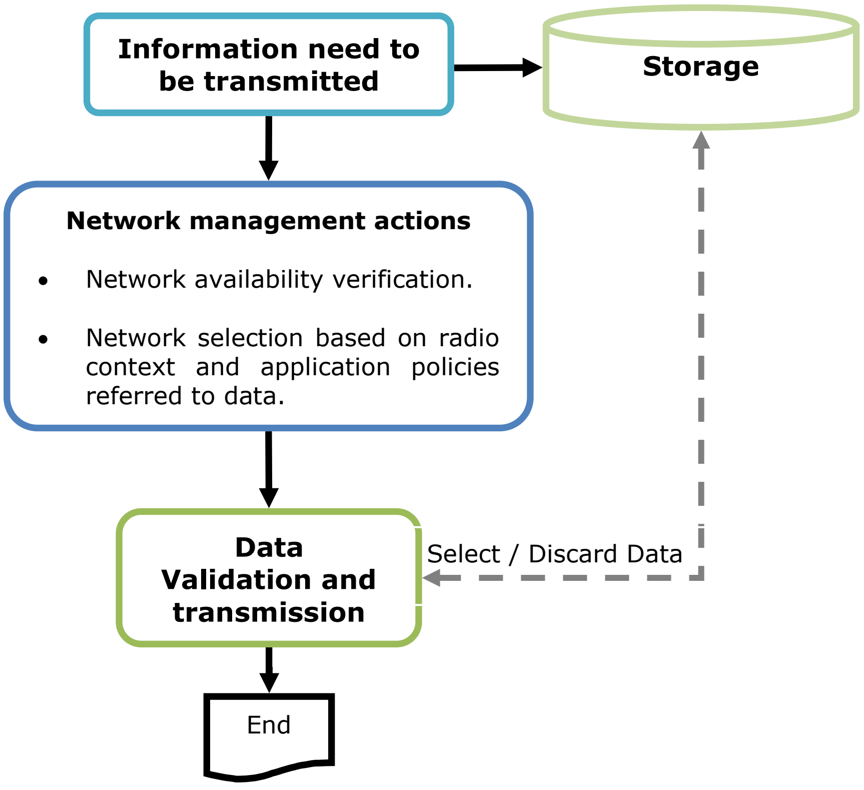

Information need to be transmitted
Storage
Network management actions
Network availability verification.
Network selection based on radio context and application policies referred to data.
Data Validation and transmission
Select / Discard Data
End

Figure 9 – Information transmission process

Figure 9 could be associated with the following use case example:

1. A handheld device (PDA/Tablet PC) gathers information from certain source within the traceability scenario.
2. Information is stored in case it is not possible to send it immediately or data transmission is interrupted due to connection loss.
3. The opportunistic manager, aware of information which needs to be sent, checks the available network interfaces and performs a network selection in order to choose the best interface to transmit, based on radio context and application policies. In case of no network availability, the process is iterated until a proper interface is available.
4. Once the interface has been chosen, the device verifies if the information is still valid and, if so, sends it using the selected network technology. If the information is not valid, the device will discard the data and will provide the proper notification (if possible).

### 5.3.4 Architecture

The current implementation of the LinkSmart middleware from the Hydra project does not provide the means to verify, before sending data, if a network is available or not. For such reason, and as explained before, the network manager should be able to cope with these issues adopting multiradio network management functionalities. It is worth mentioning that in order to have higher data transmission reliability, re-transmission mechanisms should be implemented outside the multiradio manager (e.g. within the network manager) in case for example of network loss or poor network connectivity.

Figure 10 reflects the multiradio architecture intended to be applied in ebbits.

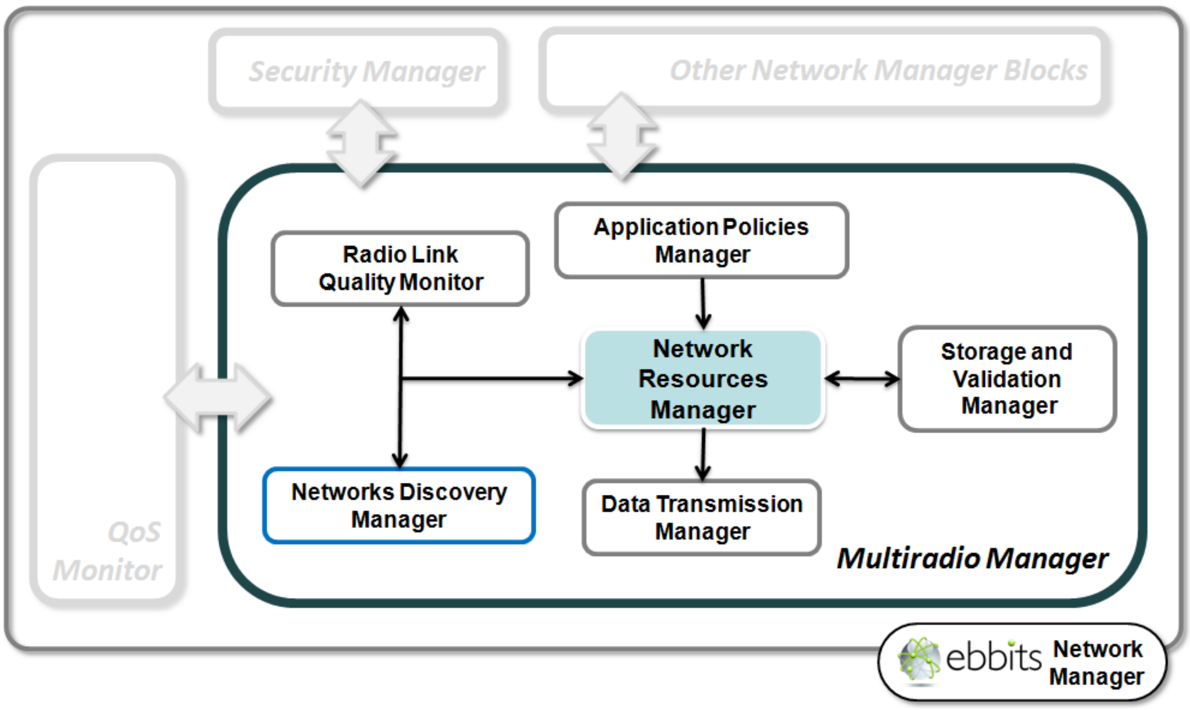

Security Manager
Other Network Manager Blocks
QoS Monitor
Radio Link Quality Monitor
Application Policies Manager
Network Resources Manager
Storage and Validation Manager
Networks Discovery Manager
Data Transmission Manager
Multiradio Manager
ebbits Network Manager

Figure 10 - Multiradio Manager Architecture

Each of the managers, and their responsibilities, included in the multiradio architecture are detailed as follows:

- **Networks Discovery Manager:** This block should be able to gather information concerning new available networks within the area range by event alerts when a network has been discovered. This information will be shared with the Radio link quality manager and network selection manager.
- **Radio Link Quality Monitor:** This block will be responsible for monitoring the radio parameters (e.g. frequency band, bandwidth, Packet Error Rate (PER), Signal-to-Noise Ratio (SNR)) which will be used during the network selection. In addition, it should check (when requested) network connectivity between the available interfaces within a device.
- **Application Policies Manager:** Based on the type, size, priority and Quality of Service (QoS) requirements of the data to be transmitted, within this block, specific policies will be defined. As mentioned before, these policies will be used for the network selection process as well.
- **Storage and validation Manager:** The data storage will be used mainly in cases where there are no networks available. Data will be stored until it is possible to transmit it using an available interface chosen by the network selection manager. In the other hand, the data validation function means that the data to be transmitted will have a "Time to Live" value which specifies the time frame in which the information is valid. Thus, the validation process will consist in verifying if the time frame is still valid or not. In case the data is no longer valid, it will be discarded providing a delete notification to the storage.
- **Network Resources Manager:** This block will gather inputs from the radio link quality and application policies managers, in order to select the most proper available interface to transmit data. Finally, the information related to the chosen interface will then be sent to the data transmission manager after the data validation process.
- **Data Transmission Manager:** This block will in charge of transmitting the data over the interface selected by the network selection manager.

### 5.3.5 Security aspects

The use of multiradio techniques is new and unexplored. This also means that available protocols often lack security procedures or that vulnerabilities have not been exhaustively investigated yet. In this section we will briefly list already identified or in our opinion possible security weaknesses.

If we consider the possibility of switching among different types of communication interfaces we have to remember that every protocol has different methods to protect the messages. If we wish to change the radio we must have some kind of security policy for the messages. For example if we require a high level of integrity of our messages we cannot use Ethernet without having cryptographic procedures implemented on the application level whereas WPA2 for example includes this kind of protection de facto. Similarly an attacker may jam some interfaces to force the device into using a less secure protocol. These kinds of policies have been used in the SERENITY project[16] which could be of help in ebbits.

If delay tolerance is built into the system by using intermediate puffers it is essential to suppress the bandwidth when connectivity returns. If this is not considered the innocent node may accidentally overload the network or the server creating a Denial of Service (DoS) attack. A malicious entity may also use this mechanism by jamming the shared channel of some nodes. While the channel is occupied the nodes would withhold their packets and unleash them the moment the channel gets free. This way an attacker could start a DoS attack without using many resources.

(A. Naveed and S. S. Kanhere 2006) identified and exploited some vulnerability of Wireless Mesh Networks. The attacks identified by them are also applicable for our case. Their attacks are based on the lack of security procedures in the channel selection protocols. The two main types of attacks which were created are the silent overloading of frequencies and the creation of quasi-stable states. In the first case a node uses high priority channels without declaring to behave this way to the neighbors. As a consequence the frequency has a silent load which results in a decrease of bandwidth for the others. This behavior is used in the Network Endo-Parasite Attack (NEPA) and Channel Ecto-Parasite Attack (CEPA). The latter case can be reached by forcing nodes into premature channel adjustment. This can create an avalanche effect as the readjustment of frequencies can get further away nodes to also change interfaces. The low-cost ripple effect attack (LORA) does this by transmitting misleading channel assignments.

## 5.4 Opportunistic support in WSN/6LoWPAN networks

In the previous section we presented an architectural solution able to support opportunistic communication within ebbits core network architecture. However, opportunistic features can be introduced also at lower level, specifically within well-defined and limited technological domains.

More specifically, WSN/6LoWPAN communication networks are taken into account as a promising technology suitable for pervasive monitoring tasks in different environments. 6LoWPAN characteristics in terms of flexibility, interoperability and scalability make this technology an appropriate solution for both manufacturing and food traceability scenarios.

As far as the manufacturing scenario is concerned, 6LoWPAN networks can be used to define near real-time applications aiming, e.g., to monitor the production process and increase energy efficiency in manufacturing plants. In this scenario, opportunistic strategies can be applied to 6LoWPAN technology to introduce frequency agility capabilities and make the solution more robust to radio interference typical of industrial environment.

In the food traceability scenario, 6LoWPAN networks can be effectively used (along with other heterogeneous technologies) to monitor the different phases of the food supply chain. This second scenario is actually characterized by high mobility and possible intermittent connectivity. The opportunistic paradigm can be here declined in the introduction of delay tolerant networking capabilities within the 6LoWPAN protocol stack.

[16] http://www.serenity-project.org/

A more detailed description of the frequency agility and delay tolerant networking features applied in the 6LoWPAN technology is presented in the following subsections.

### 5.4.1 Frequency agility

#### Introduction

Wireless communications in a typical industrial environment can be significantly degraded due to noise generated by multiple emitting sources such as industrial equipment. Under such interference, wireless networks may experience reduced throughput and higher packet losses, and may even stop operating.

6LoWPAN relies on IEEE 802.15.4 PHY and MAC layers, and while current IEEE 802.15.4 based wireless technologies apply some methods at MAC/PHY layers to cope with interference, e.g. CSMA and spread spectrum, they do not implement advanced frequency management techniques like dynamic channel allocation. As a consequence, when interference occurs at the operating channel, wireless communications can be seriously degraded and inefficiencies of static frequency allocation appear evident.

The introduction of spectrum awareness and dynamic frequency allocation capabilities to wireless networks allows efficient spectrum usage and increased network reliability. Adopting a cognitive approach, the networks become capable of constantly monitoring the occupancy state of all possible channels, and able to dynamically select the best available channel (almost) real-time when interference is detected on the current operating channel.

Experimental measurements indicate that the average throughput of IEEE 802.15.4 based nodes can be severely degraded in the presence of interference. More specifically, Figure 11 shows the results of an experiment performed by measuring the maximum, average, and minimum throughput achieved by a IEEE 802.15.4 node, given an offered load of 30 kb/s. Such measurements were carried out in realistic indoor environment with no interference and with different Wi-Fi interference data rates. The graph shows a considerable drop in the achieved throughput as the interference data rate increases.

Such behaviour strongly motivates the development of frequency agile 6LoWPANs that are capable of dynamically selecting the best available channel and adaptively reacting to interfering traffic. These self-adaptation capabilities represent the key features for improved network reliability. Thus, this section proposes the integration of such Frequency Agility (FA) features into a standard IPv6 stack that could be implemented in WSNs within the manufacturing scenario.

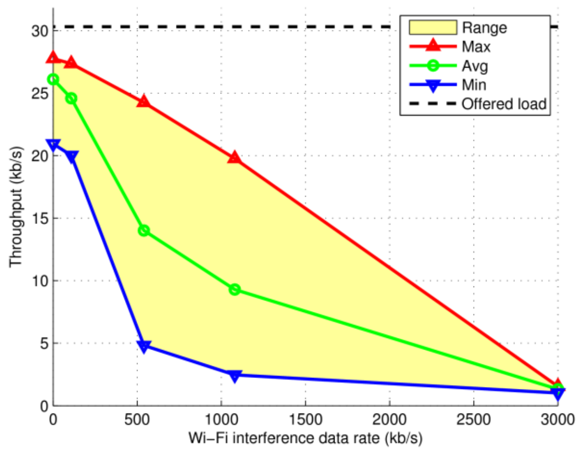


Figure 11 – Relative achieved throughput Vs. Wi-Fi interference data rate in an indoor environment

**Architecture**

A logical network architecture for the proposed 6LoWPAN based system enabling FA features is shown in Figure 12. It is to be noted that such system architecture does not necessarily reflect the physical network topology.

The FA-enabled 6LoWPAN nodes are organized into clusters to improve scalability. Inside a cluster, nodes can be assigned different roles:

1. The Cluster Head (CH) connects the cluster to the Border router (B), and hence to the network backbone.
2. Spectrum Sensing Nodes (SSNs) provide information about the spectrum occupancy state by periodically monitoring each of the available IEEE 802.15.4 channels. SSNs are capable of performing relevant PHY layer functions such as energy detection.
3. Non-SSNs carry out only application functions, e.g. wireless monitoring of water and energy consumption in a manufacturing plant. Non-SSNs do not participate in spectrum sensing.

Both SSNs and Non-SSNs provide information about the link quality and packet losses.

It is possible that nodes change their roles during run-time, according to the current environment status and network requirements.

Spectrum sensing and link quality information are communicated to an entity called the Frequency Agility Manager (FAM) that is actually part of the ebbits Network Manager. The FAM processes the data collected from the network and determines the spectrum occupancy state. This state can be determined as the combination of various parameters, including energy detection, link quality estimation, and packet loss rate calculation. The latter is required by the FAM in order to detect self-interference generated by the network when operating at relatively high data rates. In fact, using only energy detection measurements would not allow distinguishing between internal and external interference.

In fact, the FAM, following a cognitive approach, fuses different pieces of information collected from the environment and adaptively optimizes the overall network performance.

The FAM continuously keeps a record of the best available channel in the spectrum, e.g. the channel that has the lowest average energy.

If the FAM detects that the current operating channel (OC) is under critical conditions it generates an OC-switch command to reallocate the network to the current best available channel.

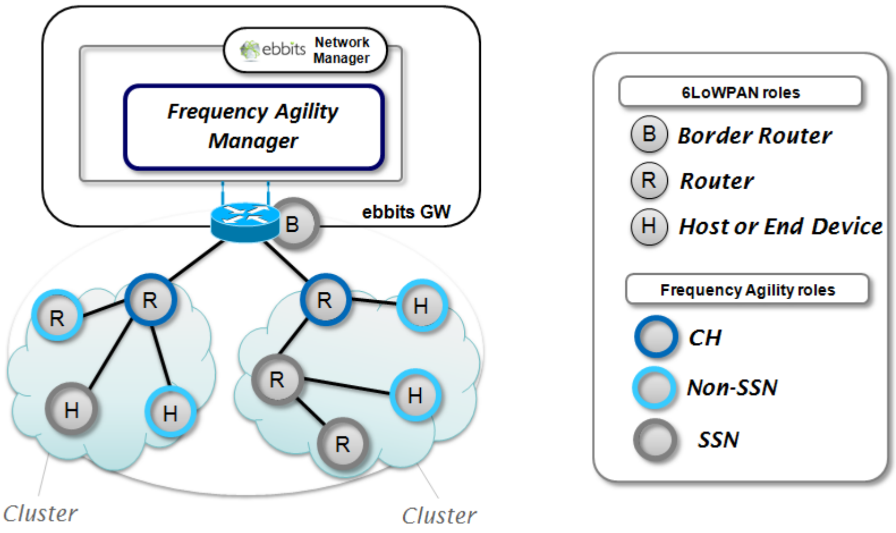


Figure 12 – Logical architecture for the proposed FA capable system

In order to provide wireless nodes with FA capabilities, we propose the extension of the IPv6 stack with two modules, the Frequency Agility Driver (FAD), and the Frequency Agility Agent (FAA), as shown in Figure 13. The FAD is a physical-layer module in charge of detecting the spectrum state and changing the operating channel. To perform these tasks, the FAD exploits functionality provided by IEEE 802.15.4 standard. The FAA manages the local FA operations of the node, and coordinates such operations with the FAM.

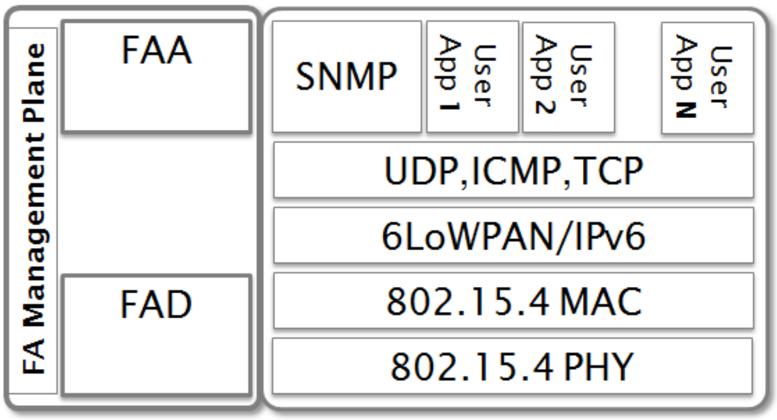


Figure 13 – Proposed FA extensions to 6LoWPAN stack

In order to join the network, we assume that 6LoWPAN nodes are able to discover the target network through scanning the available IEEE 802.15.4 channels. Afterward, nodes could leverage on optional DHCP functionalities or some other standard internet mechanisms e.g., DNS, in order to obtain the FAM address.

### 5.4.2 6LoWDTN

When limiting the attention to the restricted context of a single isolated 6LoWPAN network, opportunistic communication strategies oriented to delay tolerant applications can be developed mainly addressing routing and forwarding issues. To this aim, in this section we propose and present a DTN transport layer, called 6LoWDTN, to be positioned on top of the existing 6LoWPAN stack, just below the application layer.

#### Motivation

Several communication technologies are expected to coexist in ebbits, e.g. IEEE 802.15.4/WSN, Bluetooth, RFID, IEEE 802.11. While communication within each of such domains occurs according to well defined specifications, upper layer solutions based on multiradio gateways as well as on middleware support allow to interconnect natively heterogeneous domains and to transparently manage network nodes belonging to different domains. However, the nature of diverse communication technologies can be radically different; the features even extremely opposite, the protocol stack only conceptually similar but with rather different solutions as far as standardized and implemented protocols are concerned. Finally, the same is true for the application purposes that have originally driven conception, design, development and standardization: typical application domains and communication paradigms can thus substantially differ.

In the following, on the one side we make reference to the traceability scenario, one of those considered in ebbits. Traceability is not associated with strict requirements in terms of end-to-end delays; latency control is generally not a must: in other words, delay tolerance can be evaluated as a reasonable assumption and opportunistic communication strategies can be pursued. On the other side, we focus the attention in particular to the IEEE 802.15.4/WSN communication domain integrated in the TCP-IP world through the IPv6 and the 6LoWPAN adaptation layer.

Moving from these two pillars, WSN communication in IP domain and traceability scenario with no delay constraints and enabling opportunistic communication paradigm, we propose an architectural solution consisting of a DTN transport-oriented layer positioned on top of the 6LoWPAN stack and wrapped between the UDP protocol and the general application layer.

The goal of such a new layer, that we call 6LoWDTN and that is depicted in Figure 14 in conjunction with the rest of the stack, is to manage delay tolerant applications by means of opportunistic communication schemes derived from CHARON (Soares and Rocha, 2009)(Soares et al, 2011) key points. Although in strict ISO/OSI terms 6LoWDTN is a transport layer just as UDP, the apparently redundant presence of UDP in the global stack is explained by the easier integration that UDP guarantees with existing applications based on best-effort packet-oriented communication. In addition, from an implementation perspective it is efficient to exploit the UDP services made available by Contiki, the very popular operating system supporting 6LoWPAN stack for real solutions development.

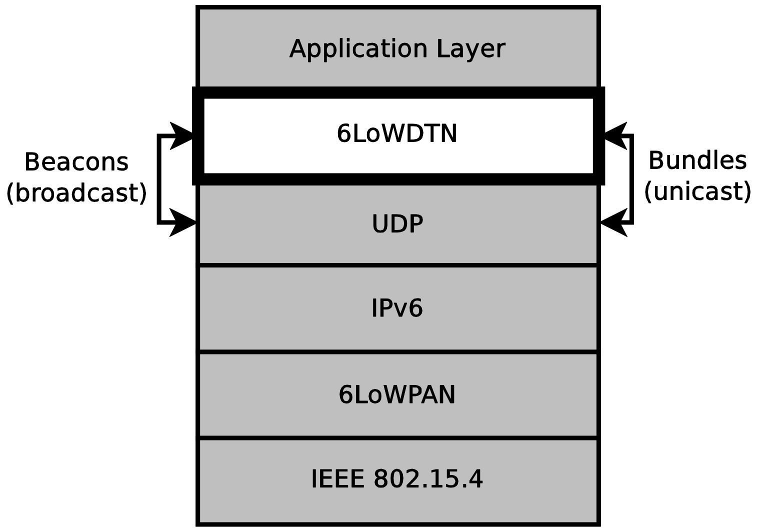


Figure 14 - The 6LoWDTN transport layer, positioned on top of the 6LoWPAN stack.

**CHARON basics integrated in 6LoWDTN**

CHARON is mainly an opportunistic routing protocol for WSN networks based on IEEE 802.15.4 standard. At the same time, such definition is limitative since CHARON also includes the development of solutions covering protocol layers other than network (routing) layer, e.g. application data traffic generation, nodes synchronization and energy saving management. Definitively, CHARON can be seen as a cross-layer and flexible solution oriented to delay tolerant scenarios in WSN networks, not necessarily over IP domain, envisioned for low consumption applications.

The most relevant aspects of CHARON design, inherited by 6LoWDTN transport layer, first of all concern the core of CHARON, i.e., the data packet forwarding policy in case of contacts with other nodes, but they also include the global network organization based on a loose distributed synchronization and the beacons exchange scheme fundamental for the overall system functioning.

As previously touched on, a simple and not particularly accurate synchronization is maintained among nodes, in the order of milliseconds, what is enough to let network carrier nodes alternate coherent (short) activity periods to (much longer) sleep periods. Indeed, in CHARON time synchronization is employed both for network overlay maintenance and for bundles meta-data. Sink nodes, always switched on and that we can assume some way permanently connected to the Internet and to a reliable time source as for instance NTP, are considered sources of perfectly correct reference time for synchronization purposes. When any two nodes enter in contact during an activity period they exchange beacons containing, in particular, synchronization information (i.e., TimeStamp – the current local time – and TimeAge – the time elapsed since the last synchronization event) such that the node that synchronized less recently always resynchronizes itself using the other node as the  current reference. While multiple sink nodes can coexist in a WSN/6LoWPAN, when considering such a network as a cloud connected to the ebbits network we can assume the existence of a single sink node which corresponds to the ebbits GW. Referring to Figure 8, the

6LoWDTN Manager included in the Network Manager resides just in the ebbits GW. It is worth observing that when messages flow from the WSN/6LoWPAN to the ebbits network the GW actually plays the role of sink, while in the case of opposite direction, from the ebbits world to the WSN/6LoWPAN, the GW is the sink of the ebbits network but it will proceed to start the delivery to a particular 6LOWPAN node. Figure 15 depicts the WSN/6LoWPAN integrated in ebbits.

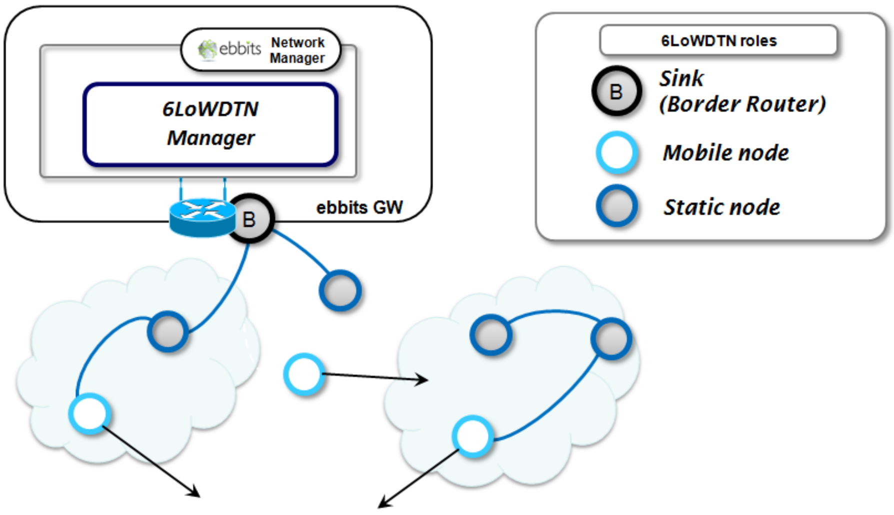


Figure 15 - 6LoWDTN Logical architecture

Beacons, which are always transmitted in broadcast, also contain other fields, namely the Estimated Delivery Delay (EDD) and the score of a Utility Function. The EDD is the estimated time, according to the statistic processing based on data collected by a node in the past contacts, that a data packet currently owned by that node could take before reaching a sink node. The EDD is the minimum time between two alternative kind of paths: the one assuming a direct contact with a sink and the one where the data packet is first forwarded to a specific neighbor (in the latter case the EDD is the sum of two times, one regarding the EDD of the neighbor and one the contact frequency with the neighbor). The score, instead, is the result of a utility function in which different metrics (EDD, storage buffer availability, battery level, etc.) are weighted as desired and which is at the basis of the decision whether forwarding or not a data packet to the node just met. As a graphical summary, the described beacon fields are reported in Figure 16 representing the structure of a beacon.

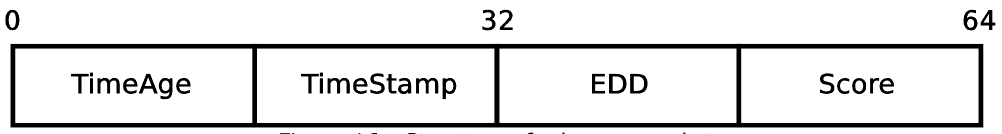


Figure 16 - Structure of a beacon packet.

In order not to flood the network with multiple replications of the same data packet, CHARON and 6LoWDTN are usually single-copy protocols, although multi-copy versions can be implemented in order to support quality of service on a goal base instead that on a connection base. Even in the most common single-copy realization, CHARON increases efficiency and flexibility through the use of zombie copies. When forwarding a data packet to a neighbor node, instead of definitively deleting it, the node keeps a frozen copy in the buffer until it is convenient. Its suppression can be decided at any moment; typically this occurs when memory space is needed. For sake of coherency about single-copy existence in the whole network, zombie copies cannot be forwarded, but only directly delivered to a sink, if met.

The header of a data packet, depicted in Figure 17, contains: a sequence number – SeqNo – in order to keep trace of any different packet; a Type field to deal with class of service and also distinguish between data packets and related ask; the current local time copied in TimeStamp and useful for calculation of end-to-end delays; the identity of the node that originally generated the data packet, reported in the field named Origin.

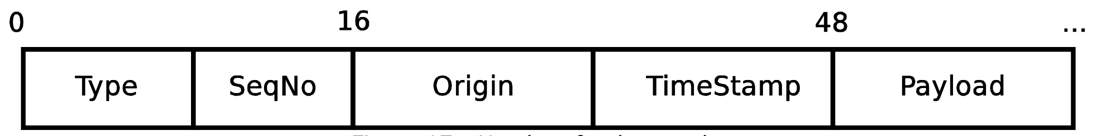


Figure 17 - Header of a data packet.

# 6. Overall network architecture

## 6.1 Base line architecture

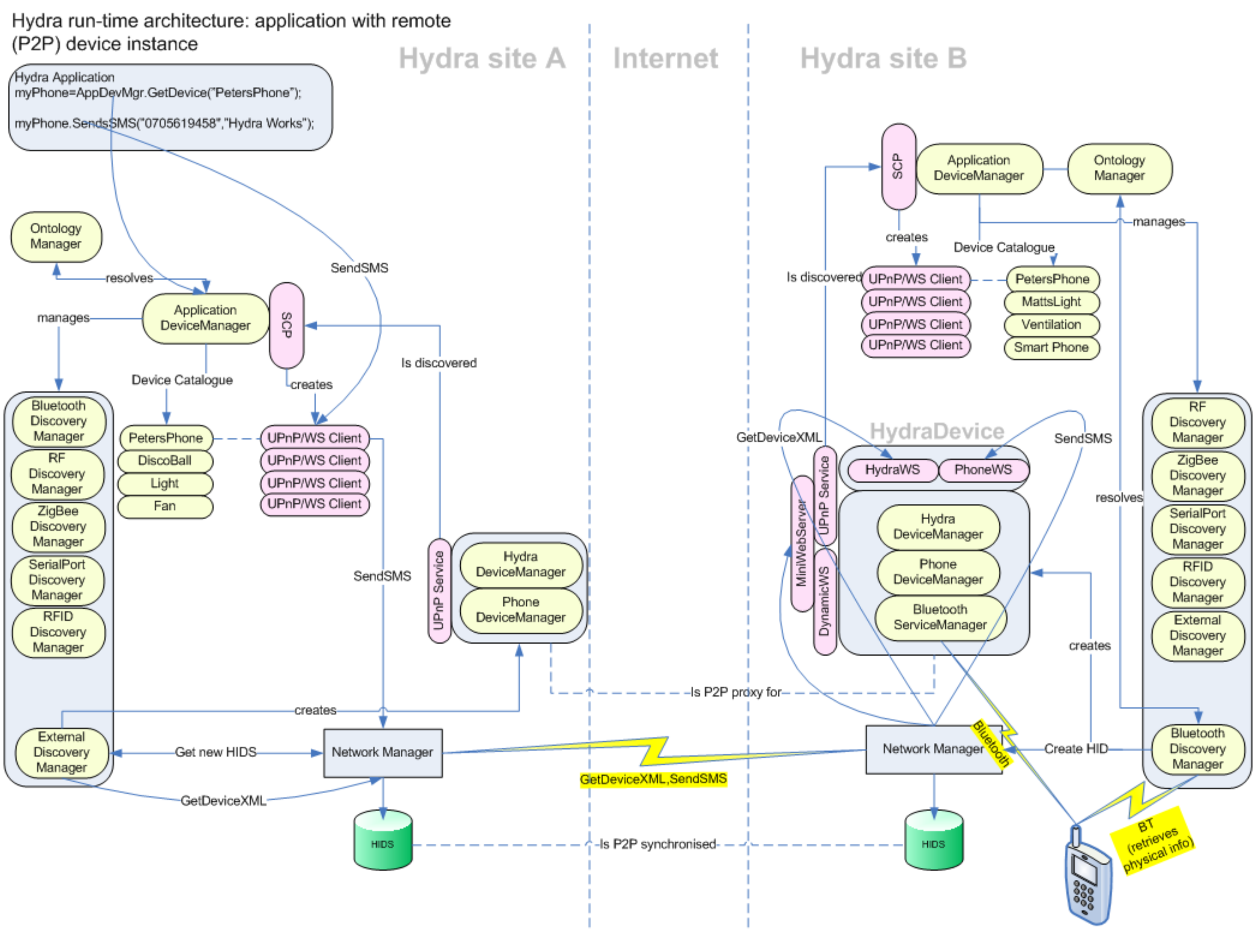


Figure 18 - Base line run-time architecture from the Hydra project

In order to understand the network architecture and the distributed discovery architecture of the LinkSmart middleware from the Hydra project it is important to understand the basic concepts/modules that are involved in these processes. The underlying enabling network functionality (i.e. P2P technology and SOAP tunneling) are described in more detail in the previous chapter 4.1. Detailed component descriptions of the LinkSmart network manager are described in chapter 6.2

The main modules in hydra that provide the functionality of entering services in the network are:

- Discovery Managers: They are responsible of discovering new devices (and services). Examples of discovery managers are Bluetooth and 6LowPAN discovery managers. Since they are working at a very low protocol level they are typically specific for each protocol used.
- Application Device Manager: Responsible for maintaining the device application catalogue (usually referred to as the DAC). The DAC is a list of all the known devices in IoT (Internet of Things) network. This list contains descriptions of which services are available as well as the HIDs for the services.
- Device Manager: Implements services that are exposed in the IoT network. For instance a Basic Switch service might implement toggling the switch on and off.

- Device proxy: Is a combination of Device Managers and a device service manager that acts a proxy in the hydra network for the actual device exposing its functionality as Web Services and/or UPnP services. This proxy manages the actual communication with device.
- Ontology Manager: Manages the device ontology, i.e. descriptions (meta data) about the device such as which events it generates et c.

Device discovery in the Internet of Things (IoT) network is coordinated by the Application Device Manager. It creates the discovery managers and also resolves the discovered devices using the device ontology. Resolving a device means that the system creates a proxy device that is addressable in the IoT network that exposes the services of the device, i.e. it picks together the proper device service manager for the protocol and the Device manager that exposes the functionality to the IoT network as a Web Service and/or an UPnP service.

When application wants to invoke devices it first needs to query the device application catalogue to find the devices. The DAC offers a query service to interact with catalogue, enabling both finding specific devices as well as all devices of a certain type. A number of DACs can exist in the network which each manages the information regarding the devices it has discovered.

The actual invocation of the services is done using SOAP tunneling and using HIDs as the addressing mechanism.

## 6.2 LinkSmart networking components

This section will provide a detailed description and break down of the different components in the network architecture.

### 6.2.1 Network Manager

**Purpose**

The Network Manager is the bottom layer of the LinkSmart middleware deployed in Gateways and in Internet-of-Things-enabled devices. It is the entry and exit point of information of the LinkSmart middleware. There is only one Network Manager per device where the middleware is deployed.

The Network Manager provides a Web Service interface (which is the main interface of the Network Manager), which is the information entry point for the middleware. Data transferred between Internet-of-Things-enabled devices and gateways should always pass through the Network Manager.

**Main Functionalities**

The Network Manager is responsible of managing the communication between Hydra-enabled devices. In order to do this, the Network Manager:

- Creates and overlay P2P network, where all the Internet-of-Things-enabled devices appear directly interconnected, no matter if they are behind a NAT (Network Address Translator) or Firewall.
- Provides indirection architecture for addressing Web Services hosted by Hydra devices using the HID addressing mechanism. Each service is identified in Hydra through an HID, which is a global and unique identifier. The Network Manager provides interfaces for other managers, applications and Hydra devices for HID creation, modification and deletion. It also offers the possibility to select the transport protocol for the service invocation between TCP, UDP and Bluetooth.
- Provides a transport mechanism over the overlay P2P network for invoking Web Services hosted by Hydra devices (SOAP Tunneling) using the HID addressing mechanism. The SOAP messages addressed to an HID are routed by the Network Manager through the overlay network to the Network Manager hosting the service. Therefore, using the SOAP Tunneling and the Network Manager any device or application is able to transparently publish and consume services anywhere, anytime, breaking the network interconnectivity barriers and independently of the service endpoint location.

- Provides a transport mechanism over the overlay P2P network for multimedia content exchange between UPnP AV or DLNA devices.
- Provides session management mechanisms between HIDs during service invocations.
- Provides time reference synchronization between different Network Managers.
- Provides a status page for developers, which the developer can use for monitoring dynamic information about the Hydra Network and the HIDs available.

**Internal Components**

The Network Manager consists of the following internal components that achieve the different networking tasks of this manager:

- Routing Manager
- Session Manager
- HID Manager
- Time Manager
- Backbone Manager
- SOAP Tunneling
- SecurityLibrary

Figure 19 presents the sub-managers included in the Network Manager and their relationships.

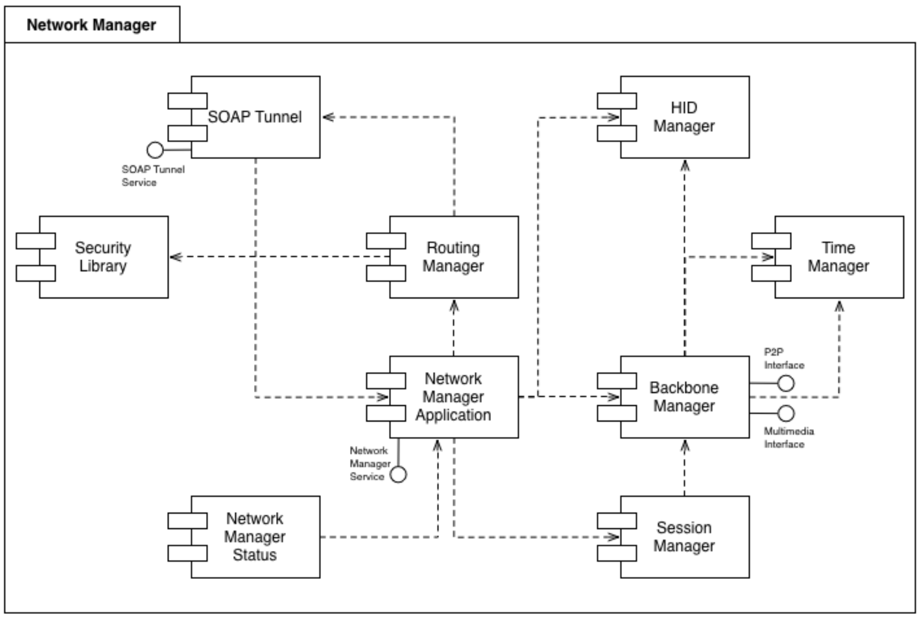


Figure 19: The Network Manager Sub-Components

The following paragraphs describe each of the Network Manager components in detail.

**Routing Manager Component**

**Purpose**

It is responsible for interacting with the others network entities. It allows sending and receiving messages to/from the network.

**Main Functionalities**

Sending data to another Network Manager

Receiving data from another Network Manager

### Description

The Routing Manager is in charge of sending and receiving the messages to and from the network. At the Network Manager level, it is the single point of communication with the other devices. It establishes a direct communication with the destination Network Manager and proposes two interfaces to send data and to receive data to and from the target node. Figure 20 shows a class diagram as the static definition of the Routing Manager.

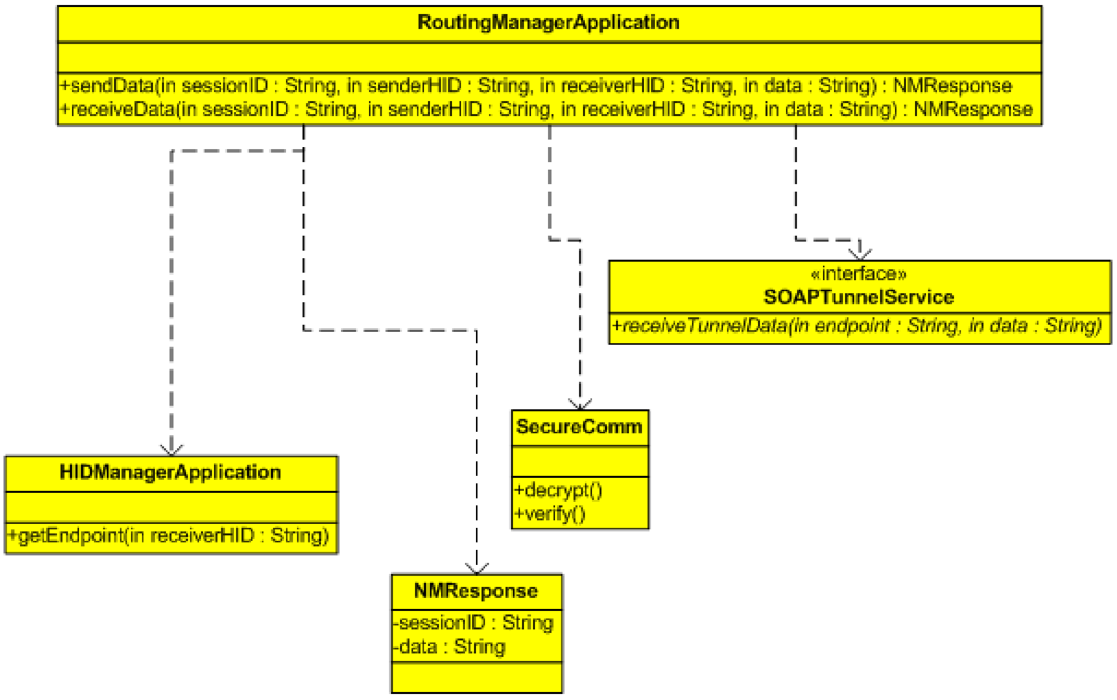

RoutingManagerApplication
+sendData(in sessionID : String, in senderHID : String, in receiverHID : String, in data : String) : NMResponse
+receiveData(in sessionID : String, in senderHID : String, in receiverHID : String, in data : String) : NMResponse
«interface»
SOAPTunnelService
+receiveTunnelData(in endpoint : String, in data : String)
SecureComm
+decrypt()
+verify()
HIDManagerApplication
+getEndpoint(in receiverHID : String)
NMResponse
-sessionID : String
-data : String

Figure 20: Routing Manager Class Diagram

### Session Manager Component

### Purpose

The Session Manager is responsible for managing sessions within the network communications inside LinkSmart.

### Main Functionalities

- Opening and closing the session with another IoT-enabled device
- Setting and retrieving user data inside the session

### Description

The session management mechanism is controlled by a Session Manager integrated within the Network Manager. The Network Manager uses the Session Manager to create and maintain the lifecycles of the session objects. Then the Network Manager Web Service interface has been updated to add the necessary session operations. The Session Manager fulfils the following features:

- Client Server model
- Session ID generated and attributed by the server
- Session ID passed through the requests to identify each communication
- Session ID used by both client and server
- Session data stored only at server side
- Session expiration enabled

- Session synchronization using the P2P communication mechanism.

The class diagram in Figure 21 shows the static definition of the Session Manager.

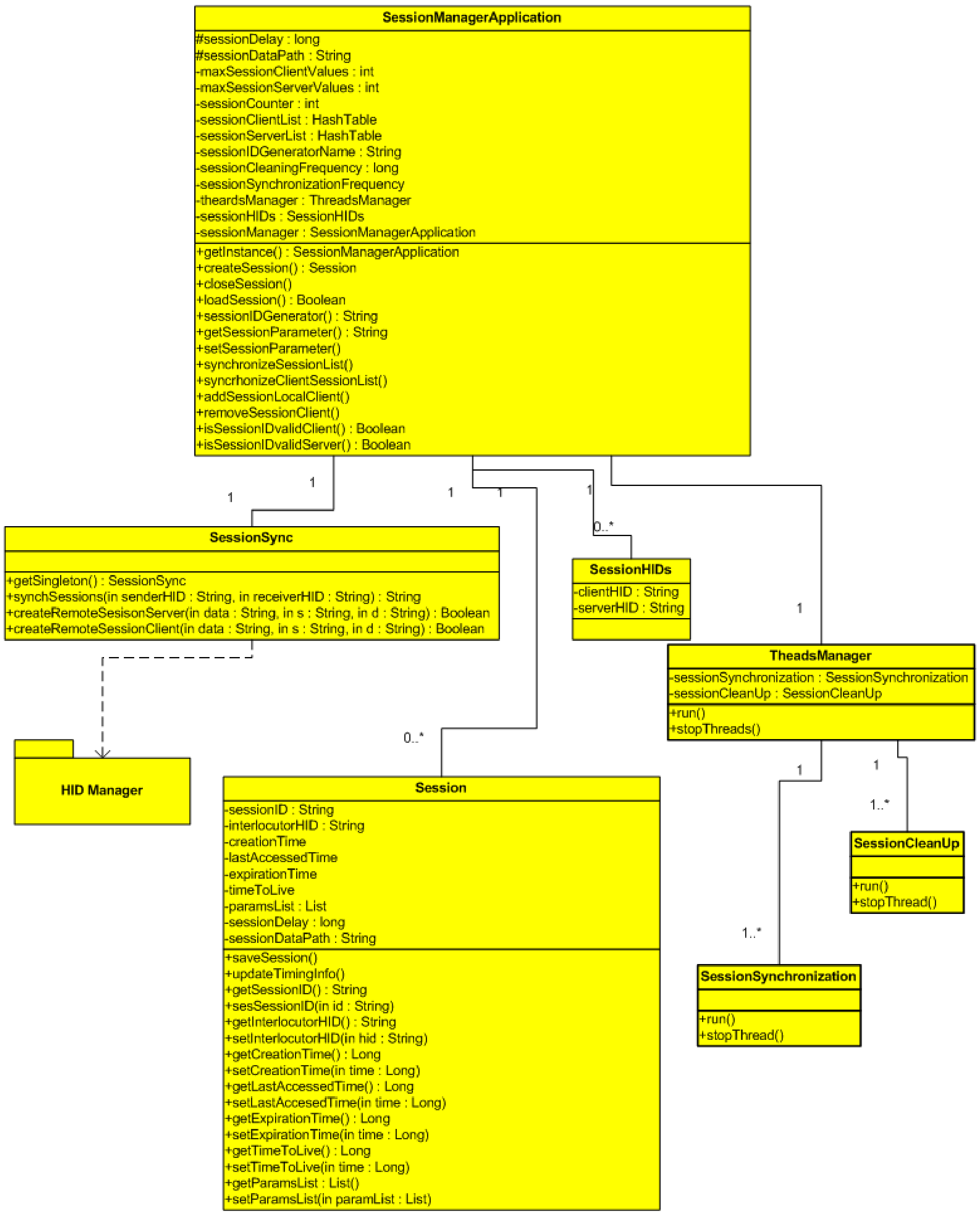

SessionManagerApplication
#sessionDelay : long
#sessionDataPath : String
-maxSessionClientValues : int
-maxSessionServerValues : int
-sessionCounter : int
-sessionClientList : HashTable
-sessionServerList : HashTable
-sessionIDGeneratorName : String
-sessionCleaningFrequency : long
-sessionSynchronizationFrequency
-theardsManager : ThreadsManager
-sessionHIDs : SessionHIDs
-sessionManager : SessionManagerApplication
+getInstance() : SessionManagerApplication
+createSession() : Session
+closeSession()
+loadSession() : Boolean
+sessionIDGenerator() : String
+getSessionParameter() : String
+setSessionParameter()
+synchronizeSessionList()
+syncrhonizeClientSessionList()
+addSessionLocalClient()
+removeSessionClient()
+isSessionIDvalidClient() : Boolean
+isSessionIDvalidServer() : Boolean
1
1
1
1
1
0..*
SessionSync
+getSingleton() : SessionSync
+synchSessions(in senderHID : String, in receiverHID : String) : String
+createRemoteSesisonServer(in data : String, in s : String, in d : String) : Boolean
+createRemoteSessionClient(in data : String, in s : String, in d : String) : Boolean
SessionHIDs
-clientHID : String
-serverHID : String
1
TheadsManager
-sessionSynchronization : SessionSynchronization
-sessionCleanUp : SessionCleanUp
+run()
+stopThreads()
HID Manager
0..*
Session
-sessionID : String
-interlocutorHID : String
-creationTime
-lastAccessedTime
-expirationTime
-timeToLive
-paramsList : List
-sessionDelay : long
-sessionDataPath : String
+saveSession()
+updateTimingInfo()
+getSessionID() : String
+sesSessionID(in id : String)
+getInterlocutorHID() : String
+setInterlocutorHID(in hid : String)
+getCreationTime() : Long
+setCreationTime(in time : Long)
+getLastAccessedTime() : Long
+setLastAccesedTime(in time : Long)
+getExpirationTime() : Long
+setExpirationTime(in time : Long)
+getTimeToLive() : Long
+setTimeToLive(in time : Long)
+getParamsList : List()
+setParamsList(in paramList : List)
1
1
1..*
SessionCleanUp
+run()
+stopThread()
1..*
SessionSynchronization
+run()
+stopThread()

Figure 21: Session Manager Class Diagram

## HID Manager Component

### Purpose

The HID Manager is responsible for managing the Hydra Identifiers (HIDs) and their correspondences with service endpoints.

Main **Functionalities**

- Creating, updating and deleting HIDs.
- Maintaining the match between HIDs and HID related information such as service endpoint, description or cryptographic information.
- Providing the different Hydra Ids associated to local Network Manager.

**Description**

The HID Manager provides a unique identifier for each device (physical or semantic) service, where its scope is the Internet-of-Things network, allowing identifying it at any time and addressing it from a logical point of view.

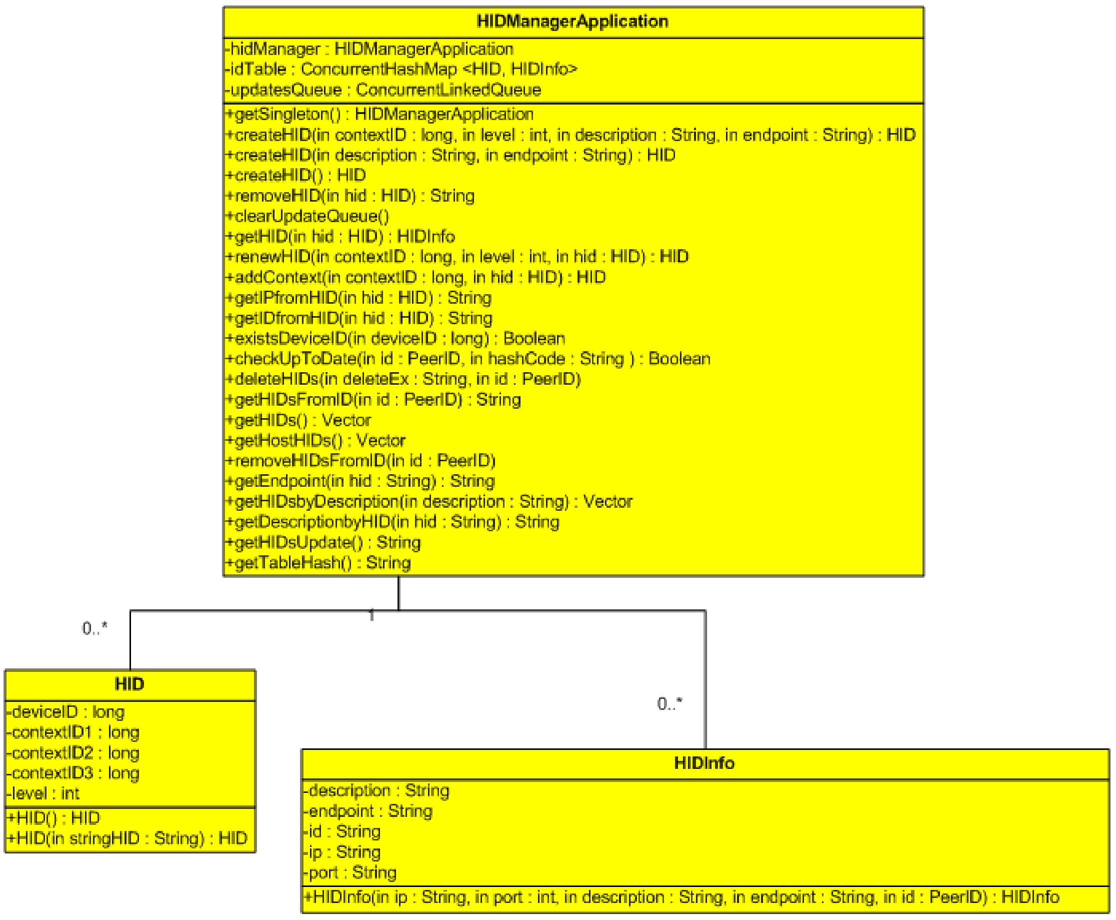

HIDManagerApplication
-hidManager : HIDManagerApplication
-idTable : ConcurrentHashMap <HID, HIDInfo>
-updatesQueue : ConcurrentLinkedQueue
+getSingleton() : HIDManagerApplication
+createHID(in contextID : long, in level : int, in description : String, in endpoint : String) : HID
+createHID(in description : String, in endpoint : String) : HID
+createHID() : HID
+removeHID(in hid : HID) : String
+clearUpdateQueue()
+getHID(in hid : HID) : HIDInfo
+renewHID(in contextID : long, in level : int, in hid : HID) : HID
+addContext(in contextID : long, in hid : HID) : HID
+getIPfromHID(in hid : HID) : String
+getIDfromHID(in hid : HID) : String
+existsDeviceID(in deviceID : long) : Boolean
+checkUpToDate(in id : PeerID, in hashCode : String ) : Boolean
+deleteHIDs(in deleteEx : String, in id : PeerID)
+getHIDsFromID(in id : PeerID) : String
+getHIDs() : Vector
+getHostHIDs() : Vector
+removeHIDsFromID(in id : PeerID)
+getEndpoint(in hid : String) : String
+getHIDsbyDescription(in description : String) : Vector
+getDescriptionbyHID(in hid : String) : String
+getHIDsUpdate() : String
+getTableHash() : String
1
0..*
0..*
HID
-deviceID : long
-contextID1 : long
-contextID2 : long
-contextID3 : long
-level : int
+HID() : HID
+HID(in stringHID : String) : HID
HIDInfo
-description : String
-endpoint : String
-id : String
-ip : String
-port : String
+HIDInfo(in ip : String, in port : int, in description : String, in endpoint : String, in id : PeerID) : HIDInfo

Figure 22: Identity Manager Class Diagram

Then the HID Manager owns and manages the *idTable*, a data structure dedicated to store the matching between logical and physical identifiers. Concretely, it associates every HID to the address of the Network Manager hosting them. The HID Manager is responsible of maintaining those Hydra Identifiers which been defined to identify each entity in a given context.

The *idTable* contains both local and remote HIDs. The remote HIDs (HID hosted in other Network Managers) are updated through the Network Manager P2P discovery process performed by the Backbone Manager. This *idTable* can be seen as a distributed DNS table, but with HIDs instead of domain names. The Identity Manager also provides several interfaces for the Backbone Manager in order to update the *idTable*. Figure 22 displays a class diagram as the static definition of the HID Manager.

**Time Manager Component**

**Purpose**

It is responsible for synchronizing the nodes of the application in the network with referential time.

**Main Functionalities**

- Providing time synchronization inside LinkSmart middleware

**Description**

The objective of the Time Manager is to ensure time synchronization among the Internet-of-Things network. This synchronization is achieved through two different processes:

- **Connection with NTP server** where the Time Manager will request the referential time to a third party (if it has an external connection).
- **Synchronization with other Time Managers (JXTA pipes)** where the Time Manager will use the Backbone Manager to broadcast the referential time to the other Network Managers that will use it to set their internal clock.

The class diagram in Figure 23 shows the static definition of the Time Manager.

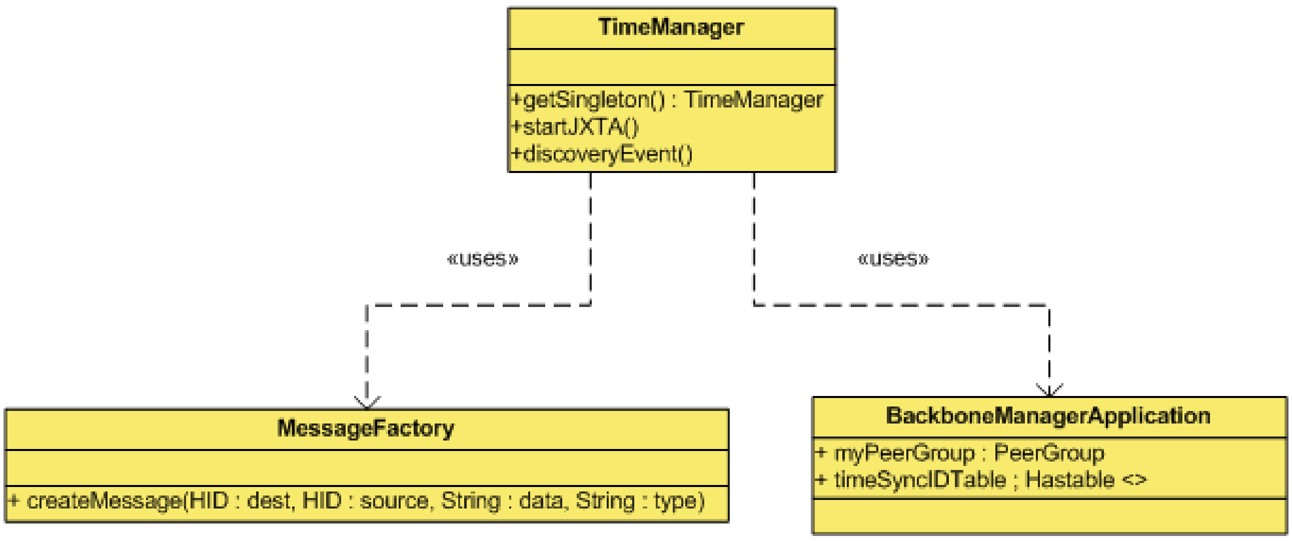

TimeManager
+getSingleton() : TimeManager
+startJXTA()
+discoveryEvent()
«uses»
«uses»
MessageFactory
+ createMessage(HID : dest, HID : source, String : data, String : type)
BackboneManagerApplication
+ myPeerGroup : PeerGroup
+ timeSyncIDTable ; Hastable <>

Figure 23: Time Manager Class Diagram

**Backbone Manager Component**

**Purpose**

It is responsible for supporting the Device-To-Device communication through a peer-to-peer architecture and discovering Hydra Ids over the Internet-of-Things network.

**Main Functionalities**

- Providing peer-to-peer based communication to the Routing Manager for exchange of SOAP messages between Network Managers (service invocations).
- Updating the idTable of the HID Manager and announce its content to other Network Managers (HID exchange)
- Providing a transport mechanism over the overlay P2P network for multimedia content exchange.
- Providing time synchronization mechanism to the Time Manager.

**Description**

The main objective of this manager is to build the overlay Hydra network, independent of network addressing and protocols. The Backbone Manager relies on JXTA P2P platform in order to build the overlay network. Every Network Manager appears in the network as a peer (or node), belonging to the peer group, a password protected peer group. Inside this group, the information from *idTable* is distributed between Network Managers through JXTA multicast sockets. Thus, each HID Manager keeps inside the *idTable* an updated list of every HID in the network.

This overlay network allows every manager to communicate with a specific HID transparently without knowing the IP address where it resides. But also allows interconnecting networks separated by NATs or firewalls through JXTA protocols.

The Backbone Manager takes advantage of the peer-to-peer architecture used inside this overlay network to offer the following services:

- Creation of node groups
- Broadcast of specific information to other Network Managers (i.e. idTable status, synchronization time)
- Discovery of other Network Managers
- Communication over JXTA pipes. Combined with the SOAP Tunnel and the HID addressing mechanism, this communication mechanism can be used for remote service invocation, independently of the endpoint where the service is located.
- Communication over JXTA sockets for multimedia content exchange.

The class diagram in Figure 24 shows the static definition of the Backbone Manager.

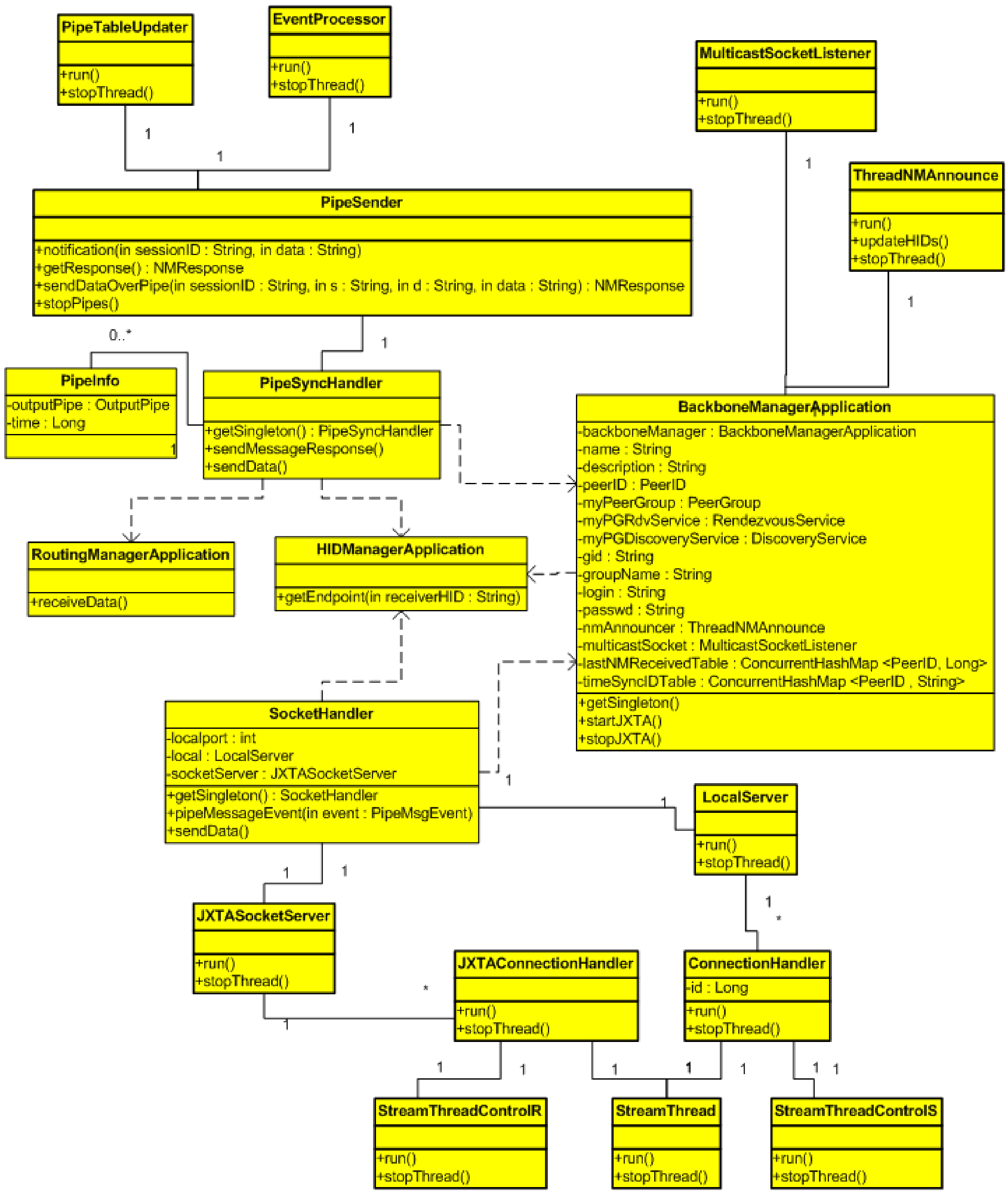


Figure 24: Backbone Manager Class Diagram

**SOAP Tunneling Component**

**Purpose**

The SOAP Tunneling for providing an interface for transparent service invocation and SOAP message transport over the LinkSmart P2P overlay network. Using this component interface, any application or manager is able to invoke services from devices located in other networks (hosted by other Network Managers) transparently.

**Main Functionalities**

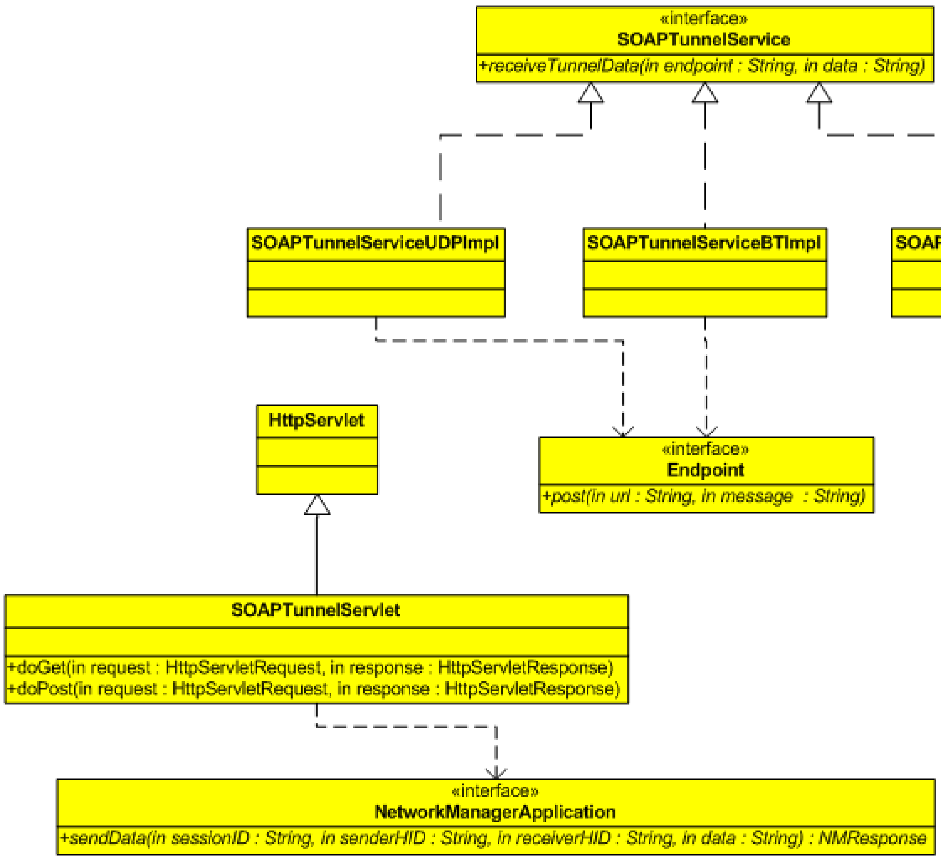
The SOAP Tunnel component provides the following functionalities:

- Invoke device services over the LinkSmart P2P overlay network.
- Allows devices to provide services even if they are located behind firewalls or NATs, in different networks.

**Description**

This component is divided in two parts: Client Side and Server Side. The client side, acting as a servlet (extends HttpServlet), is responsible for receiving the service invocation requests from managers and applications and forwarding the generated SOAP message to the Network Manager in order to be sent over the Hydra Network to the destination Network Manager where the desired service has been registered (which is identified by the HID provided by the service client).

The server side is responsible for receiving the SOAP message from client requests and posting the message to the correct destination endpoint where the service is running. It performs the translation between the HID of the service and the real endpoint, which is unknown by the application which is requesting the invocation (as it is in another network and direct invocation is not possible) but is known in the Network Manager where the service was registered.


«interface»
SOAPTunnelService
+receiveTunnelData(in endpoint : String, in data : String)
SOAPTunnelServiceUDPImpl
SOAPTunnelServiceBTImpl
SOAPTunnelServiceImpl
HttpServlet
«interface»
Endpoint
+post(in url : String, in message : String)
SOAPTunnelServlet
+doGet(in request : HttpServletRequest, in response : HttpServletResponse)
+doPost(in request : HttpServletRequest, in response : HttpServletResponse)
«interface»
NetworkManagerApplication
+sendData(in sessionID : String, in senderHID : String, in receiverHID : String, in data : String) : NMResponse
Figure 25: SOAP Tunnel Class Diagram

## 6.3 Virtual identities

Virtualization has been serving as a major guideline for the security meta-model of the LinkSmart middleware. In contrast to the understanding of virtualization as a way to abstract operation systems and platforms from the underlying hardware (such as for virtualization products like VMware, XEN or QEmu) a broad definition of virtualization which also follows the (SECURIST Advisory Board 2007) FP7 Roadmapping is used:

*A method to create logical entities that are present within a certain scope (e. g., an application) but do not need to physically exist in the same form.*

This notion of virtualization targets not only at abstractions from hardware (referring to virtual devices) but also at decoupling the mechanisms addressing and identification, thereby introducing virtual users and virtual services which refer to addressable entities whose actual identity cannot be assigned to their addresses by default. For example, several devices can be combined to a logical representation that expose the functionalities offered by the physical devices but appear in LinkSmart as one "virtual" device.

Virtualization has been realized in LinkSmart by the concept of a semantic-free addressing layer using Hydra IDs (HID) to address devices and services. As HIDs neither do provide any information about the entity they represent nor can be assumed to be persistent, they allow addressing entities without tracking them – thereby enabling the creation of "virtual" entities. An HID is a number of 4 blocks of 32 bit each. The first 3 blocks are for context information and the last block can be chosen at random by the Network Manager assigned to the device. Further, the Network Manager provides methods for devices to refresh the HID so a LinkSmart device has the option to change its HID address at any time. As a consequence, HID addresses serve only the purpose of addressing but cannot work as identifiers in order to recognize known devices. This is intended as a clear separation of addressing and identification and helps to overcome privacy problems which occur from the usage of static addresses.

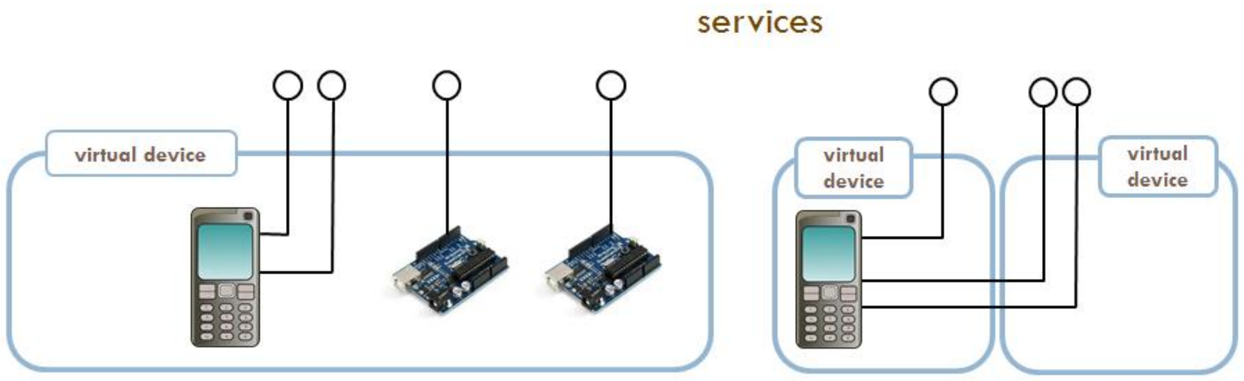


Figure 26 Virtualization of devices

As mentioned, devices can choose to change their HIDs. This may occur for two reasons. One, if the device wants to gain a new address to make tracing harder. As the HID of a node remains unchanged until it is in the same context it may happen that a big amount of traffic is generated connected to that device. This traffic could lead to information leaking (G. Danezis 2006), which can be avoided if addresses are changed regularly. The partitioning of traffic, through a change of address, is an operation which is solely decided by the communicating entity. Other case when devices could change their address is linked to the topic of opportunistic communication. When a device switches from one radio technology to another it gets registered by the local Network Manager anew and would receive a new HID. Although this may be of advantage sometimes, it could also be the requirement of the node to keep its previous HID. This could be achieved by using cryptographic tickets, which are exchanged to re-authenticate to the Network Manager. This will be examined by Task 8.5 "Security management".

## 6.4 Context hierarchies

It may be requirement of a device to keep its privacy and anonymity. On the other side it may also be the wish of the network operator to conceal the devices in his responsibility. For example a manufacturing site owner does not want to show the number of machines in the building or the size of the infrastructure. The solution is to have a kind of hierarchy in the system. The way to provide a hierarchy in LinkSmart is through the usage of contexts. The 4 blocks of the address all stand for different levels of context. For each level there should be one entity responsible for managing it. This node controls what information is allowed out of the context and what outside entities should learn about it. To provide this mechanism it has some policies which are applied after authentication. As these services are probably located at the edge of the network a Border Network Manager (B-NM) could be responsible for them. The B-NM can be reached by calling the root context address or through redirection when a prohibited device is intended to be accessed.

This also makes a better scaling of the network possible as not all nodes are known globally. The Network Managers in each level only have to know their direct neighbors and their B-NM. To find a remote device a query can be sent up each level until the device can be found but this is going to be explained in more detail in chapter (6.7).

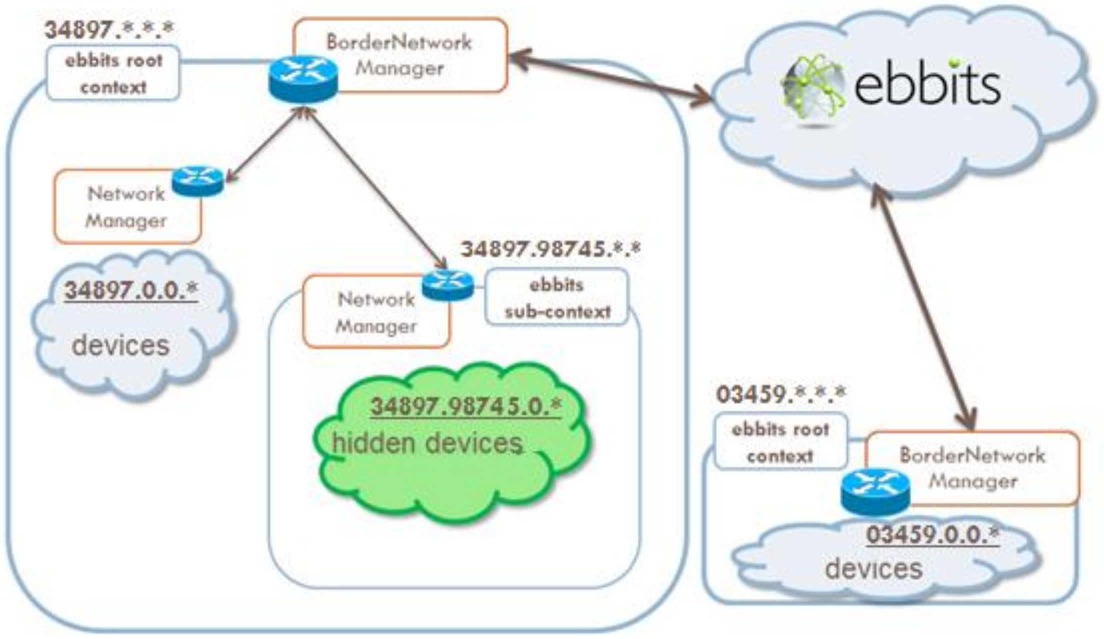


Figure 27 Context hierarchy in ebbits (address responsibilities of B-NMs and address ranges of devices)

An open issue remains: how is a context defined and what address does it get? As HIDs are used for addressing it is necessary to keep them unique over the system. As everybody may connect to the ebbits network it is hard to ensure that nodes do not take on the address of others. What can be done is to bind addresses to certificates. The certificate could be a proof of trust to other entities ensuring them that the B-NM is really in charge for the context and so for the address. The distribution of root contexts could either be done through the approval of the whole network (similar to OpenPGP) or through a central authority (like PKI). If the B-NM receives the certificate putting it into charge for the root context it can start manage its sub contexts. Sub contexts can be defined any way: based on location, organization unit, etc. How the sub contexts are defined is the responsibility of the root B-NM.

## 6.5 Accountability

In a business environment it is essential to have AAA (authentication, authorization and accounting) capabilities. If there are identities to build upon it is possible to create policies of allowed methods and have logging about operations a user has done. On the other hand as ebbits is an open environment, aimed at providing services to everybody, it cannot be assumed that everybody signs a service contract allowing organizations to check their identity. To fulfill these two aspects some kind of trade off has to be made between static identities and anonymity. One possibility to keep privacy is to have AAA based on roles like operational and support staff, technicians or service team as in the manufacturing environment (A. Pfitzmann et al. 2008). This could be reached by having only one account with a given role so everybody would have to use that username and password. Other solution would be the usage of certificates describing different roles but this would still require some method to hide the identity of the person who presented the signature.

However in the case of using roles the logs about an accident would not contain the identity of the specific person making it impossible to account someone for it. The proposed solution for this is the use of pseudonyms. A pseudonym is a chosen identity which is only known to some kind of identity authority and the person owning the pseudonym. This way, if someone sees the transactions he is not able to bind the actions to identities. But using the same pseudonym continually can make someone traceable which is against privacy regulations. If someone would change the pseudonym regularly it were possible to overcome this problem; on the other side the identity authority has to know about the changes making the process too complicated for practical use.

We propose the use of zero-knowledge proofs (ZKP) combined with roles to overcome this problem (S. Goldwasser et al. 1985). With ZKPs the only information leaked about an entity is their role. Still, through the proofs which are provided at each interaction, if there is some need to account someone for an action the identity authority can open them and reveal the real identity. To be able to use ZKPs they have to be further investigated and a network protocol has to be created based on them. This will be done in Task 8.5 Security management

## 6.6 Service discovery

As mentioned earlier ebbits is an open environment so services and devices have to be discovered by entities. The discovery has to overcome two problems. One is the search over contexts outside the direct neighborhood of the device. To find a given service the context owner has to be queried first. If it does not restrict access to the devices, the query can directly be forwarded. However in the opposite case the Border Network Manager may ask the querying device to authenticate itself before allowing the search in its context.

A special case that needs to be handled in the service discovery are the ebbits "infrastructure" services, such as the DAC (Device Application Catalogue, See chapter 6.1), the Ontology Manager et c. Basically there are two types of "infrastructure" services:

- Distributed services, such as the DAC, that exist in all different contexts. I.e. there are multiple instances of the service spread in the network.
- Centralized services which usually exist only in one place, such as the ontology manager.

The other difficulty to overcome is related to privacy issues. If a device always responds to the same type of queries it is unavoidable to get identifiable. For example, if we assume that a mobile phone has some specific services, we can search it by querying for those given services. From this information it would be possible to track the owner of this phone by simply querying for the same attributes regularly. This can be solved if there are policies on the mobile phone which restrict answering to queries without authenticating the searching party first. However, even if the policies on the device are set not to answer to everybody, the fact that a device is answering to a set of search parameters - made by someone to whom the node does answer - makes it obvious who it is. For this reason even the search parameters have to be hidden. This is going to be further investigated in Task 8.5 Security management.

## 6.7 Routing

As described in chapter 6.4 we will introduce Border NetworkManagers (B-NM) to create sub networks for performance and security reasons. The B-NM acts as a router in-between ebbits P2P networks forwarding calls to the B-NM that can route the message forward.

Since we already have the base line Hydra network manager functionality with P2P networking and a synchronised HID table we propose to use this functionality to create the B-NM implementation and routing mechanism. The Border Network managers create their own P2P network where they exchange routing information, i.e. which root context they handle. Using tP2P in this case as well will make it possible to bridge firewalls etc and create a private network, see Figure 28.

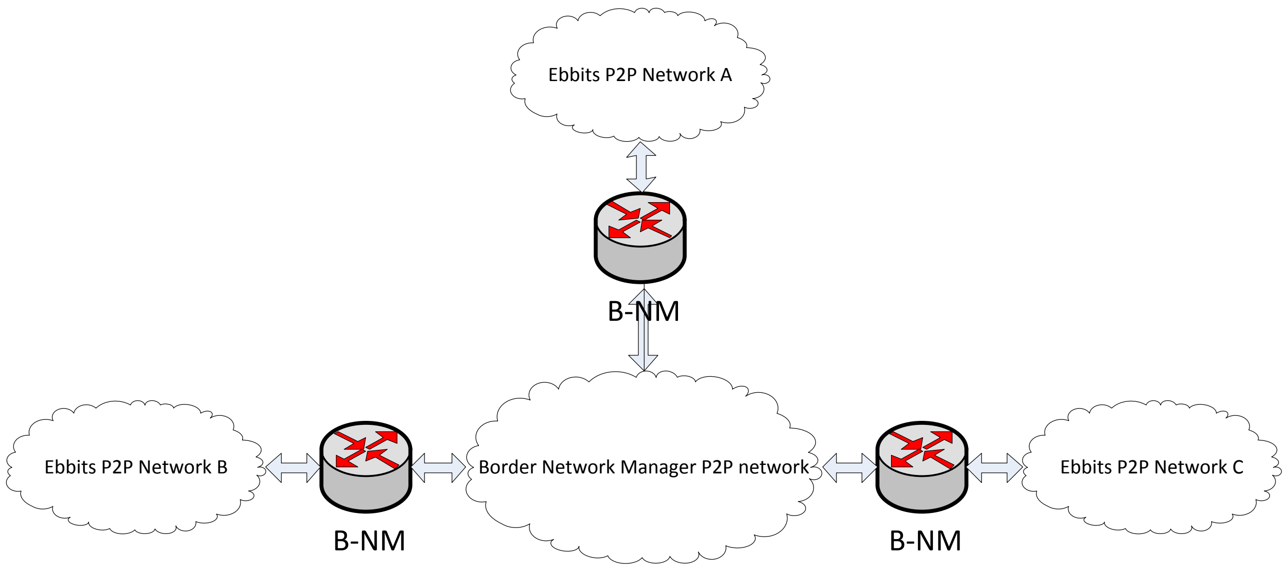


Figure 28 Border Network Manager P2P Network

The information necessary for the routing needs to be synchronised over the P2P network. The information needed in the routing table is quite limited, it would basically consist which ebbits root context is handled by which B-NM.

| ***B-NM Node ID*** | ***Root Context*** |
|---|---|
| ***A*** | 3436.*.*.* |
| ***A*** | 3438.*.*.* |
| ***B*** | 2334.*.*.* |
| ***C*** | 676.*.*.* |

Table 2 Example of routing table for B-NM

The routing algorithm needed for the B-NMs is quite straightforward and simple. When a call arrives from its inside P2P network that need to be routed it will check to see if the address matches any of the known root contexts. For example if a call is made in ebbits P2P network A for an address of 676.989.9881.9 the B-NM will identify that this call should go to ebbits P2P network C since the contexts match. It will then forward the call to the B-NM C.

If a Call arrives from the B-NM network the B-NM will forward the call onto the normal local ebbits P2P network using the standard LinkSmart routing mechanism of course taking into account security considerations and policies.

This will allow for very flexible solutions, from the routing point of view, where individual sub networks can choose how much information they want to disclose and controlling the calls going in and out from the sub network. It would also allow having B-NM within each other, i.e. networks can be arbitrarily being divided into sub-networks for security or practical reasons.

## 6.8 Connections to cloud (External data streams/services)

With recent developments in the area of cloud-enabled services for data storage and processing, a number of research projects, companies and amateurs are already looking at how public web-based/cloud technologies can be used in conjunction with Internet-of-Things applications and devices.

The above cloud-enabled services, such as Pachube (http://pachube.com/), Sen.Se (http://open.sen.se/) or TalesOfThings (http://talesofthings.com/) allow the upload of physical-data through open APIs and provide customizable services to enable processing and display of data. Pachube, for instance, allows the definition of public "data feeds". Each feed, beyond a title and a short description, can include a number of information provided by the feed creator, including the description of geographical position (latitude/longitude/altitude) of the data source, its exposure (indoor/outdoor) and its disposition (fixed or mobile). Feeds contain one or more "data streams", each characterized by a data type descriptor, a physical unit and one or more "tags" describing the data stream. In addition to these services, which are focused on the IoT paradigm, a number of demonstrations (e.g. http://bubblino.com/) have also been made by leveraging on popular micro-blogging platforms such as twitter and by enabling smart objects to publish short messages or react to some keywords published by users.

It is important to mention that integration to these kinds of service cannot be taken lightly in ebbits use-cases. These services are in fact based on the "open data" paradigm, meaning any user can freely publish data and access it. This feature, despite enabling a lot of interesting applications (including world-scale monitoring) makes these services not suitable for ebbits scenarios where data cannot be made fully public. Two major issues should be taken into account. On the one hand, in fact, it is impossible to enforce privacy and confidentiality requirements for published data e.g. it is not possible to control access to a sensor reading, once it has been uploaded. On the other hand, these services do not provide means to enforce trust with data sources.

As a result, public cloud-based services may be adopted in ebbits project only in specific use cases. Nevertheless it has been decided to outline possible integration strategies with these services for mainly two reasons: in first place, future developments in these technologies might solve the aforementioned issues within the duration of the project; in second place, some similarities exist in the approach which must be followed to integrate external private cloud-based services, which instead might be of interest for the project in evolutionary scenarios.

According to current ebbits architecture, proxies publishing or exploiting data from external public cloud-based services, if needed, must be currently kept at the edge of the ebbits architecture. Such proxies could be considered as gateways exposing the interaction with external sub-systems.

Concerning exploitation of physical-world data *FROM* the cloud, given that security/trust/privacy issues are solved externally, external data feeds could be integrated in ebbits by developing a custom version of the PWAL. Such PWAL, leveraging the open APIs from these services and a stable Internet connection, should just enrich data from external feeds with all the needed meta-data as described in deliverables D8.2 and D5.3.1.

As external services are different and use different approaches, it is difficult to characterize the specifications of this PWAL version. Nevertheless is it possible to foresee different integration strategies ranging from approaches leveraging manual configuration from the user (e.g. the user specifies that data-stream X in feed Y must be polled every hour and contains the amount of energy consumed by the whole building), to approaches leveraging meta-data already available in external services (e.g. the data type or the position in Pachube).

Concerning instead upload of physical-world data to the cloud, the issue is less complicated. Any ebbits-enabled application running on a device provided with stable Internet connectivity could include a custom software module provided with hooks to access external services APIs to upload data from ebbits according to its own implementation. Policies concerning access to data, privacy, etc. in this case can be enforced through the usual features of the ebbits platform.

# 7. Conclusions and future plans

As described in this deliverable we can reuse the networking part of the base line technology from the Hydra project. But in order to adapt it to the needs of ebbits we see that we need to add functionality in the following areas:

- Opportunistic networking:
  We need to be able to cope with more unreliable environments where communications will at times fail. This is fundamental in the ebbits platform because we are dealing both with harsh communication environments as well as traceability so there is need to guarantee the delivery of events and data.
- Id management and routing:
  For both security and performance reasons the Hydra network manager needs to be extended with routing and a addressing scheme that supports this routing mechanism. In ebbits we need to handle huge amount of devices and sites which makes it impossible to use the Hydra approach of synchronising all ID over the P2P network. The security aspect of the routing and addressing mechanism need to be addressed as well. This will be done by extending the Hydra network manager and by creating a Border Network Manager which is responsible of routing in-between different ebbits networks.
- Wireless sensor networks:
  We need to extend the device support in LinkSmart to cover WSN based networks to deal with the specific problems of those, such as latency, memory- and computational constraints.

For the next iteration we will be further extending the base line components and test different aspects of the extensions. The testing of the routing and addressing schemes are extremely important since it will determine the performance and reliability of the system. The same is applicable for the opportunistic networking components which need to be thoroughly tested in order to make ebbits meet its requirements.

# 8. List of figures

# 9. References

(Ahlsen et al. 2011) Ahlsen, M., Kool, P., Rosengren, P. and Kostelnik, P. (2011). "D7.3.1 Implementation of Data and Event Management models 1." ebbits Project Deliverable, Rep. No: D7.3.1, FP7 project 257852 - ebbits, February 2011.

(Berggren et al. 2005) Berggren, F.; Bria, A.; Badia, L.; Karla, I.; Litjens, R.; Magnusson, P.; Meago, F.; Haitao Tang; Veronesi, R.; , "Multi-radio resource management for ambient networks," *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on* , vol.2, no., pp.942-946 Vol. 2, 11-14 Sept. 2005.

(Contiki 2010) The Contiki home page http://www.sics.se/Contiki (accessed 2011-05-21).

(G. Danezis 2006), "Introducing Traffic Analysis", 2006

(Esper) Esper - Complex Event Processing.", http://esper.codehaus.org/, retrieved 02/2011.

(Etzion et al. 2010) Etzion O., Niblett, P. Event Processing in Action. ISBN: 9781935182214, pp. 384. Manning Publications, 2010.

(Ferrus et al. 2010) Ferrus, R.; Sallent, O.; Agusti, R.; , "Interworking in heterogeneous wireless networks: Comprehensive framework and future trends," *Wireless Communications, IEEE* , vol.17, no.2, pp.22-31, April 2010.

(S. Goldwasser, S. Micali and Ch. Rackoff 1985), "The KnowIedge Complexity of Interactive Proof Systems", 1985

(Gustafsson et al. 2003) Gustafsson, E.; Jonsson, A.; , "Always best connected," *Wireless Communications, IEEE* , vol.10, no.1, pp. 49- 55, Feb. 2003

(IEEE 2009) IEEE; "IEEE Standard for Local and Metropolitan Area Networks- Part 21: Media Independent Handover," *IEEE Std 802.21-2008* , vol., no., pp.c1-301, Jan. 21 2009.

(Kostelnik et al. 2011) Kostelnik, P., Hreno, J., Ahlsen, M., Alcaraz, G., Pastrone, C., Spirito, M., Madsen, T. N., Checcozzo, R., Paralič, M. and Furdík, K. (2011). "D7.2 Event and data structures, taxonomies and ontologies." ebbits Project Deliverable, Rep. No: D7.2, FP7 project 257852 - ebbits, February 2011.

(Lardies 2009) Lardies, F. M. (2009). "Deploying Pervasive Web Services over a P2P Overlay". **in** 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, IEEE Computer Society. 2009.

(Mansor and Wan 2010) Mansor, S.; Tat-Chee Wan; , "Mobility Management in Heterogeneous Wireless Access Network with IEEE 802.21 Services," *Computer and Network Technology (ICCNT), 2010 Second International Conference on* , vol., no., pp.110-114, 23-25 April 2010.

(SecurIST Advisory Board 2007). "7recommendations for a security and dependability research framework: from security and dependability by central command and control to security and dependability by empowerment.", 2007

(A. Naveed and S. S. Kanhere 2006), "Security Vulnerabilities in Channel Assignment of Multi-Radio Multi-Channel Wireless Mesh Networks," *IEEE Globecom 2006*, pp. 1-5, Nov. 2006.

(A. Pfitzmann and U. L. D. Kiel 2008), "Pseudonymity , and Identity Management – A Consolidated Proposal for Terminology", 2008

(Shen 2010) Shen, H. (2010). Content-Based Publish/Subscribe Systems. In X. Shen et al. (eds.), Handbook of Peer-to-Peer Networking, Springer Science + Business Media, pp. 1333-1366.

(Soares and Rocha 2009) Soares, J.M. and Rocha, R.M., "CHARON: Routing in low-density opportunistic wireless sensor networks," in Wireless Days (WD), 2009 2nd IFIP, dec. 2009, pp. 1 –5.

(Soares et al. 2011) Soares, Jorge M. and Franceschinis, Mirko and Rocha, Rui M. And Zhang, Wansheng and Spirito, Maurizio A., "Opportunistic data collection in sparse wireless sensor networks," EURASIP J. Wirel. Commun. Netw., vol. 2011, pp. 6:1–6:20, January 2011.