Universal Integration of the Internet of Things through an IPv6-based Service Oriented Architecture enabling heterogeneous components interoperability

Grant agreement for: Collaborative project
Grant agreement no.: 288445
Start date of project: January 1st, 2011 (36 months duration)

**Deliverable D3.2**
Smart Routing Mechanisms Design

<table>
<thead>
<tr>
<th>Contract Due Date</th>
<th>30/06/2013</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submission Date</td>
<td>30/06/2013</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Responsible Partner</td>
<td>University of Luxembourg</td>
</tr>
<tr>
<td>Author List</td>
<td>F. Melakessou, T. Cholez, J. François, M.R. Palattella</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
<tr>
<td>Keywords</td>
<td>Internet of Things, IPv6, Routing, QoS, Software-Defined Networking, Content Centric Networking</td>
</tr>
</tbody>
</table>

Project Coordinator: Mandat International (MI)
Sébastien Ziegler sziegler@mandint.org
# Table of Contents

**Executive summary** ............................................................................................................................................. 7  
  General overview .................................................................................................................................................. 7  
  Selected solution .................................................................................................................................................. 7  

1 **Overview** ....................................................................................................................................................... 8  
  1.1 Purpose and scope of the document ............................................................................................................. 8  
  1.2 Document outline ......................................................................................................................................... 9  

2 **Introduction** ..................................................................................................................................................... 10  
  2.1 Routing fundamentals ................................................................................................................................. 10  
  2.2 Best Effort and Quality of Service ............................................................................................................. 11  
  2.3 Smart routing definition ............................................................................................................................. 12  
  2.4 Generic scenario ......................................................................................................................................... 14  

3 **Enabling Technologies** ................................................................................................................................. 16  
  3.1 IPv6 traffic class and flow label ................................................................................................................. 16  
  3.2 Content Packet filtering ............................................................................................................................. 18  
    3.2.1 Existing Packet Analyzers .................................................................................................................... 18  
    3.2.2 Technological choices ............................................................................................................................ 23  
    3.2.3 xtables .................................................................................................................................................. 24  
    3.2.4 NPTv6 / RFC6296 ................................................................................................................................ 29  
    3.2.5 Conclusion ........................................................................................................................................... 29  
  3.3 CCN Based Smart Routing ............................................................................................................................ 30  
    3.3.1 CCN-based IoT ....................................................................................................................................... 30  
    3.3.2 CCN basics ............................................................................................................................................ 31  
    3.3.3 Problem definition ................................................................................................................................ 33  
    3.3.4 High-level Smart Routing .................................................................................................................... 35  
    3.3.5 Low-level smart routing ....................................................................................................................... 37  
    3.3.6 Evaluation ............................................................................................................................................ 39  
    3.3.7 Conclusion ........................................................................................................................................... 41  
  3.4 Openflow Based Smart Routing .................................................................................................................... 43  
    3.4.1 Introducing Software-Defined Networking (SDN) ........................................................................... 43  
    3.4.2 OpenFlow: an overview ........................................................................................................................ 44  
    3.4.3 Feasibility study for using IPv6 Flow label with Openflow .................................................................. 47  
  3.5 RPL Based Smart Routing ............................................................................................................................. 49  
    3.5.1 RPL Topology Formation ....................................................................................................................... 49  
    3.5.2 RPL Control Messages and Metrics ...................................................................................................... 51
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3.2</td>
<td>Smart Routing Mechanisms Design</td>
<td></td>
</tr>
<tr>
<td>3.5.3</td>
<td>The Objective Function</td>
<td>51</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>3.6</td>
<td>ARCs Based Smart Routing</td>
<td>53</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Definitions</td>
<td>53</td>
</tr>
<tr>
<td>3.6.2</td>
<td>ARC Algorithm</td>
<td>55</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Example: building ARCs on a Simple Network Topology</td>
<td>57</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Simulation Environment</td>
<td>61</td>
</tr>
<tr>
<td>3.6.5</td>
<td>ARC Simulation on NARVAL</td>
<td>64</td>
</tr>
<tr>
<td>3.6.6</td>
<td>Comparison between ARC and RPL</td>
<td>68</td>
</tr>
<tr>
<td>3.6.7</td>
<td>Conclusion</td>
<td>71</td>
</tr>
<tr>
<td>4</td>
<td>Illustrative use cases</td>
<td>72</td>
</tr>
<tr>
<td>4.1</td>
<td>Building Maintenance Process</td>
<td>72</td>
</tr>
<tr>
<td>4.2</td>
<td>Safety Alert with QoS</td>
<td>73</td>
</tr>
<tr>
<td>4.3</td>
<td>The Smart Office</td>
<td>74</td>
</tr>
<tr>
<td>5</td>
<td>Conclusion</td>
<td>75</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>76</td>
</tr>
</tbody>
</table>
D3.2 Smart Routing Mechanisms Design

List of Tables

Table 1 ARCs’ information ........................................................................................................ 54
Table 2 Application of NARVAL_R_ARC ............................................................................... 64
Table 3 Application of NARVAL_R_CursorARC ................................................................. 65
Table 4 Application of NARVAL_R_RoutingARC for the destination node 12 .................... 66
Table 5 Application of NARVAL_R_RoutingARC for the destination node 14 ....................... 67
Table 6 Predecessor vector (RPL) ........................................................................................ 69

List of Figures

Figure 1 Network physical architecture .................................................................................. 10
Figure 2 IoT6 Network Structure .......................................................................................... 10
Figure 3 Example of High level smart routing ....................................................................... 13
Figure 4 Example of Low level smart routing ....................................................................... 13
Figure 5 Low-level and High-level smart routing in a generic scenario ................................. 15
Figure 6 Libpcap packet capture process .............................................................................. 19
Figure 7 Wireshark graphical user interface .......................................................................... 20
Figure 8 Architecture of the Linux-based Content Switch ....................................................... 22
Figure 9 Main operations of content switch .......................................................................... 23
Figure 10 Netfilter Packet Flow (image under Creative Commons Attribution-Share Alike 3.0 Unported, details at http://en.wikipedia.org/wiki/File:Netfilter-packet-flow.svg) ................................................................. 28
Figure 11 CCN Forwarding Engine ........................................................................................ 32
Figure 12 CCN Messages ....................................................................................................... 33
Figure 13 Hierarchical CCN Naming ..................................................................................... 33
Figure 14 Common IoT network ............................................................................................ 34
Figure 15 CCN-based IoT network ........................................................................................ 34
Figure 16 Sampling optimization problem ............................................................................ 35
Figure 17 On demand notification (pull) ................................................................................ 35
Figure 18 Subscription based mechanism (push) .................................................................... 37
Figure 19 Optimal forwarding strategy .................................................................................. 37
Figure 20 GCD-based optimization ....................................................................................... 38
Figure 21 CCN Smart Forwarding ........................................................................................ 39
Figure 22 Impact of the number of consumers ...................................................................... 40
Figure 23 Impact of the number of counters ......................................................................... 41
D3.2 Smart Routing Mechanisms Design

Figure 24 Software-Defined Networking Architecture ................................................................. 43
Figure 25 OpenFlow Architecture: separating data plane from control plane .......................... 44
Figure 26 Architecture of an OpenFlow network ........................................................................ 45
Figure 27 Handling of incoming packets in an OpenFlow switch .............................................. 46
Figure 28 OpenFlow-based smart routing in IPv6 network, using Ipv6 Flow labels ................. 48
Figure 29 RPL encapsulation ........................................................................................................ 51
Figure 30 Routing graph view ........................................................................................................ 54
Figure 31 Arc View representation ............................................................................................... 55
Figure 32 Physical topology ........................................................................................................... 57
Figure 33 Comparison among the final forwarding topology built by different routing protocols ..... 61
Figure 34 NARVAL module ........................................................................................................... 62
Figure 35 Screenshots of the NARVAL functionalities on Windows ........................................... 62
Figure 36 Network Topology ......................................................................................................... 64
Figure 37 Multiple paths generated by the ARC algorithm for a communication between the node 12 and the root node 1 .................................................................................. 67
Figure 38 Multiple paths generated by the ARC algorithm for a communication between the node 14 and the root node 1 .................................................................................. 68
Figure 39 RPL topology for the source node 1 ........................................................................... 70
Figure 40 Path between the node 14 and the root node 1 ............................................................ 70

<table>
<thead>
<tr>
<th>List of Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>6LoWPAN</td>
</tr>
<tr>
<td>API</td>
</tr>
<tr>
<td>ARC</td>
</tr>
<tr>
<td>CCN</td>
</tr>
<tr>
<td>DAG</td>
</tr>
<tr>
<td>GCD</td>
</tr>
<tr>
<td>ICN</td>
</tr>
<tr>
<td>IoT</td>
</tr>
<tr>
<td>IPv6</td>
</tr>
</tbody>
</table>
### D3.2 Smart Routing Mechanisms Design

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>Libpcap</td>
<td>Library Packet Capture</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>oLAF</td>
<td>open Lowest ARC First</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality-of-Service</td>
</tr>
<tr>
<td>RPL</td>
<td>Routing Protocol for Low power and Lossy Networks</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Networking</td>
</tr>
<tr>
<td>SPF</td>
<td>Shortest Path First</td>
</tr>
<tr>
<td>TC</td>
<td>Traffic Class</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
</tbody>
</table>
Executive summary

**General overview**

This deliverable D3.2 deals with the smart routing functionalities investigated in the context of the IoT6 project with the objective to select the most relevant approach for further implementation and test within the real IoT6 architecture. This document is therefore the result of Task T3.2, focused on smart routing mechanisms, but it also encompasses part of Task T2.2 which subtopic on the QoS has been moved to T3.2 because of the relevance of both smart routing and QoS topics.

We first propose an extensive definition of smart routing which can be summarized by the following statement: smart routing is a routing mechanism embedding some intelligence for optimizing information flow. More concretely, and based on the use cases addressed by IoT6 (which are described at the end of the document), smart routing will mainly consist in sending a message to the most appropriate destination host(s) based on the meaning of the carried information.

We then describe several technologies that can enable smart routing features at a larger sense. Most of the technologies we consider are still at the design stage and proposed for research purposes only, in particular:

- **Content-Centric Networking (CCN):** It is a recent routing paradigm based on named data instead of named host. We describe how the CCN paradigm can be adapted to sensors’ communications for the IoT world. We propose some optimizations and we evaluate them.
- **Software Define Networking through OpenFlow:** It is also a new routing paradigm where the different network devices are programmable and receive instructions from a central manager. Here, we describe the technology and how it could use IPv6 features to enable smart routing.
- **Routing Protocol for Low power and Lossy Networks (RPL):** It is the routing protocol associated with 6LoWPAN in WSNs. We describe it and more precisely how it can be used for smart routing by properly defining metrics and object functions.
- **Available Routing Construct (ARC):** It is a new proposition to build routing graphs in WSNs proposed by Cisco with the particularity to automatically build backup paths to allow fast recovery in case of network failures. We describe the algorithm and we evaluate its efficiency through simulations.

**Selected solution**

However, we also considered more classic approaches to implement smart routing based on Content Packet filtering techniques. In particular, we show how xtables (ip6tables) in conjunction with the u32 and string modules can be used to provide the IoT6 architecture with easily implementable and reliable smart routing features. In particular, we provide a detailed analysis of the definition of rules to be made to process 6LoWPAN / IPv6 packets while enabling smart routing functionalities, as well as advices for the proper integration within the IoT6 architecture.

---

1IoT6 – http://www.iot6.eu
1 Overview

1.1 Purpose and scope of the document

The IoT6 research project aims at researching and exploiting the potential of IPv6 and IPv6-related technologies to develop an Open Service-oriented architecture overcoming the current Internet of Things fragmentation.

This document is the result of Task T3.2, focused on smart routing mechanisms. As outlined in the proposal, smart routing aims at redirecting incoming data flows to different IPv6 addressed servers according to the packet content and some pattern recognitions. A common example is the temperature value that a sensor emits on the network. The corresponding data packet might be forwarded to distinct servers depending on the temperature value. For instance, a very high temperature in a room can be considered as abnormal and can be sent to a priority server in order to trigger the necessary actions to reduce the temperature like sending a message to the building maintenance, switching on the air conditioning system or even automatically triggering fire protection systems.

The first objective of this deliverable is to precisely and formally define what is smart routing with a larger view than the sole scope of IoT6 architecture and use cases, in order to have a broader impact on the IoT community.

By nature, smart routing is closely related to the network routing at a lower level, which is essential for reaching the different hosts like for example the servers which are responsible to deal with the temperature values in the previous example.

Therefore guaranteeing an efficient and reliable routing is also essential. For example, a command that will trigger a fire protection system has to reach different emergency destinations as fast as possible and in a reliable way. This needs to differentiate and prioritize (to avoid any drop) certain messages. Such an approach is part of the QoS which was initially planned to be addressed in Task T2.2 and Deliverable D2.2. However, this can also be seen as a smart way to route messages in the network. Therefore, QoS approaches are integrated to the current deliverable by considering two kinds of smart routing:

- High level smart routing where a smart router can decide to change the destination of a message based on its content
- Low level smart routing which addresses performance issues by leveraging QoS techniques.

Hence, Task T2.2 and Deliverable D2.2 are now only dedicated to security and privacy which is more coherent than having smart routing and QoS in separate documents.

In the scope of smart routing, decisions are taken regarding the content of a packet. The term content refers either to specific IPv6 header fields, like the Traffic Class (TC) or the Flow Label, or the specific payload.

This deliverable describes several technologies that can enable smart routing. Firstly, a well-supported technology (traffic filtering with iptables) has been investigated in order to ease its integration in the scope of WP5 and reduce the risk of non-delivery of a working smart routing technology. Secondly, OpenFlow, as a skyrocketing technology, is also considered for high level smart routing. For low level smart routing, we researched on Available Routing Construct (ARC) in cooperation with CISCO which has developed this new routing approach highly tolerant to network and node failures. Finally, using Content Centric Networking (CCN) for smart routing has been explored. Being one implementation of Information Centric Networking (ICN), it is naturally well adapted to the high level smart routing because this routing paradigm is driven by the content itself. In addition, we propose optimization in order
D3.2 Smart Routing Mechanisms Design

to enhance QoS in CCN when deployed within a WSN.

1.2 Document outline

Since smart routing is related to network routing, the Introduction in Chapter 2 firstly reviews the fundamentals of this topic and also introduces common QoS mechanisms. Then, a definition of “smart routing” is given and illustrated through a small use case and a generic scenario.

After introducing IPv6 features that can be useful for smart routing, Chapter 3 presents the different enabling technologies:

- Content packet filtering (traffic filtering): after a review of the state-of-the-art approaches, details about leveraging xtables/iptables for IPv6 based smart routing are given. In particular, integration constraints have been carefully considered for avoiding useless overhead in WP5,

- CCN: as a recent paradigm, necessary background is given to the reader to show how CCN can be used in the context of smart routing for the IoT world. Optimizations are then proposed and evaluated,

- OpenFlow: based on a review of OpenFlow usable features, we propose an adaptation of smart routing based on these features,

- ARCs: to show enhancement provided by this new routing solution, a simulator was specifically designed and implemented.

Chapter 4 highlights the usability of the smart routing for IoT6 use cases, in particular to those which have been finally selected in WP7 for further investigation and demonstration. Finally, a general Conclusion is given in Chapter 5.
D3.2 Smart Routing Mechanisms Design

2 Introduction

2.1 Routing fundamentals

Routing represents the process of selecting paths in a network where data traffic will be sent. In communication networks, packet forwarding enables the transit of packets from their sources towards their destinations through intermediate nodes such as routers, bridges, gateways, firewalls or switches. Routing decisions are usually based on routing tables locally stored in each intermediate node. In fact, routers maintain a record of routes to various network destinations. Routing algorithms aim at building and maintaining these routing tables.

As shown in Figure 1 and Figure 2, in the IoT6 network architecture, we can identify three distinct network domains:

- IPv6 Sensor Network (or SensorNet, DeviceNet) interconnecting smart objects mainly through wireless technologies (Wireless Sensor Network, i.e. 802.15.4 and wifi) but also sometimes through wired technologies (i.e. X10, BACnet, KNX)
- IPv6 Local Network (or LocalNet, ServiceNet)
- Internet Wide-Area interconnecting smart objects and smart servers (they may be also more locally connected through the ServiceNet)

Each domain should be able to route and forward packets in order to enable data communications between any components of the IoT6 architecture.
Different routing strategies have been proposed either based on static or dynamic routes. In particular, different dynamic route updates have been proposed like the distance vector, the link state of the path vector strategy.

Static routing is generally applied for small networks. In that case, routing tables are manually assigned. When the network size grows and becomes dynamic, it becomes quickly impossible to fill and update all routing tables in an efficient way in terms of time and efforts. Thus routing algorithms aim at automating the construction and the maintenance of these routing tables in respect with specific information exchanges. As a consequence, after a cost optimization based on specific metrics defined by network administrators (hop distance, bandwidth, load, delay, jitter, MTU, reliability, etc.), each router is capable to select the best next hop (router) where to send each data packet in order to reach its destination in an optimal way.

Distance vector and link state routing are used for intra-domain communications. Distance vector algorithms are based on the Bellman-Ford [31]. An ID and a cost are assigned to each physical link of a network. These algorithms finally provide for each couple of distinct node of a network the path with the lowest total cost. Link-state algorithms are based on a shared global vision of the network topology. In that case, each network entity floods the network with its local neighbouring information. Thus each node can build a complete graph of the network topology where the path between itself and any other network node can be performed in respect to the shortest path algorithm such as the Dijkstra’s[32].

The path vector routing protocol like BGP (Border Gateway Protocol) [33] is used for inter-domain communications. It is similar to distance vector routing. In fact, a speaking node is assigned to each autonomous system. It is responsible for the creation and maintenance of inter-domain routing tables.

### 2.2 Best Effort and Quality of Service

The dynamic routing mechanisms described in Sec. 2.1 are already able to take decision on current network situations. In fact, they can automatically readapt the routing tables according to new events, like link failure, through a mechanism of announcements.

However routing decisions are taken based on pure network characteristics, in particular the network topology and link costs, which are related to performance metrics. This routing approach that mainly aims to simply forward the packet with a minimal cost based on performance metrics, is known as best effort which also implies that no guarantee is given on the final “end-to-end” delivery. The latter does not apply any distinction between the natures of packets to be transmitted.

In reality, all applications do not need the same level of Quality of Service (QoS). This has led to the introduction of QoS routing mechanisms that aim to differentiate the packet routing process based on the specific requirements of the application they support. For example, Voice-over-IP (VoIP) requires low delays. More specifically, the objective of QoS mechanisms is to optimize the network resource usage to guarantee the specified quality of service (in term of throughput, delay, error rate, etc.), for different classes of applications and services.

In IPv4, there are two main approaches for enabling QoS. A first method consists of reserving the necessary resources in advance such that performances are guaranteed. The best example is RSVP [3] but such approaches generate a lot of overhead due to the management of a reservation itself. In contrast, DiffServ [4] is qualified as traffic engineering by creating class of traffic using some attributes of packets as properties. Once the different classes are created, a router can process the different class differently. For example, there can be a highest priority class whose packets will be forwarded first.
2.3 Smart routing definition

In this section we provide a clear definition of what “Smart routing” means in the context of the IoT6 project.

Smart Routing can be defined as a routing mechanism embedding some intelligence for optimizing information flow. Based on how and where the intelligence is embedded, it is possible to identify:

- **Low-level Smart Routing**
- **High-level Smart Routing**

The Low-Level Smart Routing is based on the classical approaches, Best Effort and QoS, introduced in Sec. 2.2. As already specified, the objective of the Best Effort is to provide good performances for all communications; while the aim of the QoS mechanisms is to provide a different level of performance for different applications.

In addition, IoT6 envisions a more advanced smart routing that can automatically identify the more suitable destination to forward information embedded in the packet data. Unlike Best Effort or QoS approaches, this routing mechanism implies that the destination (IP address) can be modified by the smart routing process itself. Due to the fact that it operates at a higher level by considering applications' needs, it has been defined as high level smart routing.

In directed communication models, the destination IP address of a message is specified when it is sent. It can be more transparent through the use of a name and a server address, or a group identifier (multicast). In any case, the responsibility to forward the data packet to a specific destination (a group or a single host) is assigned to the sender. With the suggested high-level smart routing, a smart router affects dynamically the destination of a packet based on the embedded information (packet headers and/or payload). This high level smart routing is a critical component which improper use can lead to security issues like the Denial of Service of the communications that would never reach the destination. Therefore, great care must be taken that any changes to routing tables is appropriate and carried out only with the right authorization.

Within the IoT6 scenarios, it may be possible that the sender (for instance a smart object such as an IPv6 sensor) does not know a priori the destination, which can change according to the data content (normal server, priority server, etc.).

The dissemination of sensor information is a classic example where high-level smart routing can be applied. In fact, related to IoT is the dissemination of sensor information. In many cases, the treatment is different regarding the information. For example, if the sensor value (e.g., a temperature value) fluctuates within a normal range of values, this information is reported to the monitoring system. However, if the sensor monitors abnormal values, this information may also be directly sent to another system, like an alarm or a protection system to reduce the potential damages. In fact, this abnormal value is probably a sign of a malfunctioning. For example, a high temperature value in a building may trigger automatically fire protection systems.

Figure 3 shows how the high-level smart routing operates in the network scenario described above. The smart routing decisions (i.e., the final destination(s) of the message sent by the sensor) can be taken, as illustrated in the Figure 3, by a specific device, or by an entity co-located with other equipment (e.g., the IoT6 smart board).

Finally, Figure 4 highlights in a single-network infrastructure scenario how the low-level smart routing takes decisions through the classical network routing approach, where each intermediate node forwards the message according to a routing table and the specified destination address without modifying the latter.
We now provide a formal definition of smart routing that works for both low-level and high-level. Let us assume that:

- The data to be transmitted, denoted as $D$, is characterized by a set of attributes $\text{Attr}_D = \{D_1, D_2, \ldots, D_n\}$,
- The Network, denoted as $N$, is characterized by a set of attributes $\text{Attr}_N = \{N_1, N_2, \ldots, N_m\}$
- The forwarding strategy $S$ considers both the system (i.e., queue level), and network level (i.e., next hop or final destination).
- The smart routing is the process that defines the forwarding strategy $S$ of $D$ within $N$ based on the data attributes $\text{Attr}_D$ and the network attributes $\text{Attr}_N$. It can be symbolized as a function $F$ where $F(\text{Attr}_D, \text{Attr}_N)$ returns a forwarding strategy.

This generic definition covers all the previous described use cases. For example, in the low-level smart routing, based on the best effort strategy, the IP address is considered as the data attribute, and the next-hop is the forwarding strategy in standard associated algorithm. The QoS based low-level smart routing considers multiple network attributes (number of hops, bandwidth, delay, error rate, etc.), in order to operate traffic classification. The forwarding
D3.2 Smart Routing Mechanisms Design

strategy also operates at the system side by prioritizing data transmission. Finally, high-level smart routing evaluates multiple application-specific attributes at the data level to change the final destination in the forwarding strategy.

This deliverable addresses smart routing from these different perspectives. Because classical Best Effort solutions and QoS approaches have already been developed and evaluated [5], the effort in the context of the IoT6 project was oriented mainly towards the adaptation and evaluation of new paradigms for applying high-level smart routing. However, we also propose enhancement for low-level smart routing as well:

- **Packet filtering**: Using standard packet filtering approaches, it is easily possible to modify packets, including the different header fields. This is also a potential solution for high-level smart routing. The easiest solution consists of using an explicit agent or proxy at a known address to which the sender always delivers data packets [23]. Then this agent is given the proper credentials to edit the destination address field and takes the proper decisions about the location where to distribute the message.

- **CCN**: Content-based communication is a communication service where the flow of messages from senders to receivers is driven by the content of the messages, rather than by explicit addresses assigned by senders and attached to the messages. By design, this is thus a good candidate for high-level smart routing since the final destination machines is not defined by the sender at the beginning but automatically assigned by the routing itself. In this deliverable we introduce a novel CCN-based communication scheme for Sensornets which optimizes the message overhead for such low power environment. Since CCN can be qualified as both high-level (when running over IP) and low-level (when natively supported) smart routing. In this deliverable we envision CCN as a low-level approach in the Sensornet up to the Localnet but not beyond due to the lack of large scale support of such disruptive technologies.

- **Software Defined Networking with a strong focus on OpenFlow[25]**: Such a paradigm decouples data plane access and control plane access. This basically allows a decentralized controller to take decisions about routing instead of routers themselves. Such decisions can be taken by considering the attributes of the traffic flows.

- **ARC [30]**: It is a recent proposition from Cisco which extends the standard best effort approach by guaranteeing a backup path for data in case of failure. It is thus dedicated to low level smart routing. The added “smart” value here being the costless construction of a backup path for all the links.

Depending on the solution, the decision leading to the smart routing can take place either within the Sensornet or the Localnet. More precisely, ARC and CCN will affect routing at the Sensornet level, SDN at the Localnet level and Packet filtering at the Smart Gateway level.

### 2.4 Generic scenario

In order to illustrate in a very generic way how the smart routing acts, we can consider the following configuration:

- An information producer $P$, e.g., a sensor
- At least two consumers, $C_1$ and $C_2$, which are interested into the values provided by $P$, according to different specific criteria. Note that the number of consumers can be larger than 2.

Figure 5 shows how the two smart routing levels operate in such network scenario.
D3.2 Smart Routing Mechanisms Design

- The low-level smart routing level operates at the network level or below by deciding on the forwarding paths between a producer and its consumers in order to reach the best or acceptable performances
- The high level smart routing implies the use of a specific smart router device, $SR$. The data is initially sent from the producer to $SR$. Then, $SR$ decides who the final recipient of data is.

As shown in the figure, both approaches are not exclusive and so can be used in combination.

*Figure 5 Low-level and High-level smart routing in a generic scenario*
3 Enabling Technologies

This section presents the different enabling technologies that are good candidates for smart routing in the context of IoT6.

Smart routing is used to better distribute the workloads and traffic within half-gateways of the IoT6 architecture. In general, a rule-based routing mechanism based on a combination of configurable conditions and predefined set of criteria is used to make decisions. The workloads are forwarded to the best fitting server for processing.

Smart routing systems have the ability to route traffic depending on a set of pre-defined rules set by the administrator.

This is different from routing in traditional networks where decisions are done in respect with destination address. In the smart routing approach, decisions are made according to additional data attributes such as sensed value, location, time, sensor id, energy level, etc.

3.1 IPv6 traffic class and flow label

Traffic Class (TC) is an 8 bits field in the IPv6 header which has been designed to define traffic classes. Although this could have been a natural candidate for smart routing by differentiating packets, this should be avoided because it is already reserved to be used for DiffServ. In addition, a field may be modified by intermediate nodes in the network. Hence, assuming a packet is marked using this field and regarding the content, it would be logical not to change it.

However, it is still possible to use our smart routing approach in conjunction with DiffServ if we assume that smart routing rules are defined coherently to the definition of DiffServ traffic priorities.

As defined by the RFC 6437 [6], another field called flow label, should not be modified once the sender sets it. A flow is defined as a sequence of packets sent from a particular source to a particular (unicast or multicast) destination for which the source desires special handling by the intervening routers (priority control). A source can use the 20-bits flow label field in the IPv6 header to discriminate all packets of a flow and so then applying a specific process and in particular for QoS. Therefore, using the flow label is usable for smart routing purposes. At least, even if not supported at the Internet scale, flow label can be used to implement smart routing in the Sensornet and in the Localnet.

However, smart routing as envisioned in section 2 and in the context of IoT6, consists of taking routing decision based on information content and therefore payload, which is not by default in the IPv6 flow label. Two options can be thus considered with their advantages and drawbacks:

- The smart routing decisions are taken by inspecting the packet payload
  - Advantage: No need for the source of messages to translate the content into a flow label
  - Drawback: Risk of fragmentation which will thus prevent proper parsing, computation overhead induced when processing the payload because deep packet inspection techniques are needed

- The smart routing decisions are taken by inspecting the IPv6 flow label
  - Advantage: Even fragmented, each packet will be marked, low computation overhead
  - Drawback: The source of message has to calculate a flow label based on the
D3.2 Smart Routing Mechanisms Design

content of the message

While the zero flow label (0x00000) is normally reserved, this still lets $2^{20} - 1$ usable tags for QoS differentiation: from 0x00001 to 0x7FFFFF. It corresponds to more than one million which should be enough for most of smart routing scenarios. In addition, associating to the content a flow label will speed up the packet analysis as this limits the analysis to headers up to layer 3.

Therefore, smart routing assumes that the source node of information that wants to leverage smart routing capabilities has to apply a function which computes the corresponding flow label of the content of the message. It mainly corresponds to the message priority. Otherwise, the flow label field should be set as 0.

For completeness, content-based filtering (application layer) is still considered in the following sections.
3.2 Content Packet filtering

Packet filtering consists of analyzing every transmitted packet for applying a specific treatment based on rules. Such techniques are usually used for security purposes (firewall) but also for routing purposes for example by performing NAT through packet modifications. In the past, packet filtering was almost limited up to layer 3 like IP which thus limits the analysis to the headers of the corresponding protocols. Recent advances in packet filtering enable further analysis by also being able to analyze the upper layers as well. More sophisticated techniques that parse the payload of packet might be possible either for security or routing purposes. Therefore, smart routing naturally may use such an approach. This section will firstly describe the main existing technologies before selecting one that fits the IoT requirements.

3.2.1 Existing Packet Analyzers

3.2.1.1 Libpcap

Libpcap (Library packet capture) is an open-source and portable user-level packet capture and filtering library, included in many packet network analysers such as Tcpcap, Dsniff, Kismet, Snort or Wireshark. It was developed by the tcpdump.org group. The processing of packet capture consists of collecting data transmitted over a network, either for incoming or outgoing packets as shown in Figure 6. It enables low-level network monitoring, with applications such as network statistics gathering, network traffic analysis, security monitoring, etc. It provides an Application Programming Interface (API) that permits to capture network traffic. Libpcap is available for all Unix-like systems. Another version is also available for Windows operating systems (WinPcap).

http://www.tcpdump.org/
3.2.1.2 Wireshark/Tshark

Wireshark\(^3\) and Tshark are the well-used network packet analyzers which propose respectively a graphical user interface or a command line interface. Relying on Libpcap, they permit to capture data packets travelling through a network. Wireshark also enables to display and interactively browse data traffic information captured on a computer network. Wireshark is available for Unix and Windows environments. It captures live traffic composed by data packets crossing a network interface. It also displays packets with very detailed protocol information as shown in Figure 7. It is possible to save traffic traces for later processing and analysis. It is interoperable with a lot of other capture programs. Filtering features only enable to select defined packet of interest. Many criteria can be chosen combined with specific colorization of the displayed packets based on filters. Many statistics can be easily retrieved from traffic traces.

\(^{3}\) http://www.wireshark.org/
D3.2 Smart Routing Mechanisms Design

Here is a non-exhaustive list of features:

- Deep inspection of hundreds of protocols such as IPv6, etc.
- Live capture and offline analysis.
- Standard three-pane packet browser.
- Multi-platform (Windows, Linux, OS X, Solaris, FreeBSD, NetBSD, etc.).
- Compatible with many different capture file formats: tcpdump (libpcap), Pcap NG, NetScreen snoop, etc.
- Live data can be read from Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, etc.
- Decryption support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP and WPA/WPA2.

3.2.1.3 Netfilter

Netfilter\(^4\) is a packet filtering framework inside Linux 2.4.x and later kernel series. iptable is based on this framework. Netfilter enables packet filtering, network address and port translation (respectively NAP and NAT) and other low-level packet processing. It is based on a natural evolution and improvement of the tools ipchains (Linux 2.2.x) and ipfwadm(Linux

\(^4\) www.netfilter.org
D3.2 Smart Routing Mechanisms Design

Netfilter is a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack.

Netfilter enables to:

- Build internet firewalls based on stateless and stateful packet filtering (IPv4 and IPv6).
- Deploy highly available stateless and stateful firewall clusters.
- Use NAT to implement transparent proxies.
- Aid tc and iproute2 tools to build sophisticated QoS and policy routers.

3.2.1.4 IPtables

IPtables is a generic table structure for the definition of rule sets. Each rule within an IP table consists of a number of classifiers (iptables matches) and one connected action (iptables target). It refers to the kernel-level components.

IPtables is the userspace command line program used to configure the Linux 2.4.x and later IPv4 and IPv6 packet filtering rule set. It is targeted for system administrators. IPtables can be used for Network Address Translation (NAT). IP6tables is used for configuring the IPv6 packet filter.

The main features are:

- Listing the contents of the packet filter rule set
- Adding/removing/modifying rules in the packet filter rule set
- Listing/zeroing per-rule counters of the packet filter rule set

3.2.1.5 L7-filter

L7-filter is a new packet classifier for the Linux kernel. Unlike other classifiers, it doesn't just look at simple values such as port numbers. Instead, it does regular expression matching on the application layer data to determine what protocols are being used.

L7-filter is a Netfilter match module which classifies packets based on application layer (OSI layer 7) data. This means that it is able to classify packets as HTTP, FTP, Gnucleus, Kazaa, etc., regardless of ports. It complements existing matches that classify based on port numbers, packet length, TOS bits, etc.

QOS-L7 package is Coyote Linux & BrazilFW implementation of L7-filter capabilities. L7-filter support is already built in standard Coyote Linux and BrazilFW distributions. This package simplifies L7-filter configuration and integrates L7-filtering capabilities to existing QoS classes. It uses web-based configuration and management capabilities provided by webadmin interface. Coyote (BrazilFW) QOS class configuration and manual QOS class configurations are supported.

---

5 http://l7-filter.sourceforge.net/

6 http://dolly.czi.cz/coyote/packages/qosl7.asp
3.2.1.6 Linux-based Content Switch

The Linux-based Content switch (LCS) [34] depicted in Figure 8 is based on the Linux 2.2-16 kernel and the related LVS package. LVS is a Layer 4 load balancer which forwards the incoming request to the real server by examining the IP addresses and port numbers using some existing schedule algorithm. LVS source code is modified and extended with new content switching functions. LCS examines the content of the request, e.g., URL in HTTP header and XML payload, besides its IP address and port number, and forwards the request to the real servers based on the predefined content switching rules. Content switch rules are expressed in term of a set of simple conditional statements. The process is illustrated in Figure 9.

![Architecture of the Linux-based Content Switch](image)

*Figure 8 Architecture of the Linux-based Content Switch*
Content switch rule has a simple syntax: 

$$\text{if (condition) \{action1\} [else \{action2\}]$$

The rules are typically expressed in terms of content pattern or conditions that cover the class of packets to be matched, and its associated actions. In the existing products, there are two basic ways that rules are specified:

- Using directly the command line interface. The syntax are typically similar to the CISCO Access Control List (ACL) convention.
- Using a special language to specify the pattern and describe the action of the service. The rule set is then translated and downloaded into the content switch. A network classification language can be used to classify the incoming traffic and describe the action of the packet: 

  Rule <name of the rule> {predicate} {action_method()}. The predicate part of the rule is a Boolean expression that describes the condition.

For example, to reroute a packet to a specific server when the embedded data is deviant regarding the mean, a rule will be formulated as:

If (data>${\sigma}\ast\text{mean data}$) \{routeTo(serverAlarm);\}

Rule matching algorithm directly affects the performance of the Content switch. It is related to the packet classification techniques. In a layer 3 switching, the switch only considers the IP address and the port number of the packet which are in the fixed field. So the rule matching process can be easily sped up by using a hash function. In Content switch, higher layer content information is needed. This information such as URL, HTTP header or XML tag are not from the fixed fields and have different length, so it is hard to build a hash data structure to speed the searching process.

### 3.2.2 Technological choices

As highlighted in section 4.3.1, several technologies may be used in the context of smart routing. However, the Linux-based content switch is obsolete and L-7 filter is highly recent with so some lack of maturity. In contrast, netfilter/iptables is widely supported and integrated within most of Linux kernels and distribution. It is also widely used and well documented. It is thus the best choice to ease the integration of smart routing within the smart board defined in WP5. In particular, iptables-based smart routing has been successfully tested on the smart
D3.2 Smart Routing Mechanisms Design

board.

Smart routing is performed in two steps when a packet arrives:

- Retrieving the *label*
- Forwarding the packet towards the right destination regarding the *label*

Such functions are realized through the configuration of iptables. In IPv4, this is very easy as this can be interpreted as NAT using the dedicated table. However, one objective for IPv6 is to get rid of such mechanisms which have mainly been designed to compensate the limited number of IPv4 addresses.

Therefore, ip6tables, which is the IPv6 version of iptables, does not support similar functionalities by default. In the following paragraph, this problem is addressed by xtables extensions to support raw destination NAT.

For each of them, the functionalities will be assessed in terms of communication support but also in terms of integration and operational deployment.

### 3.2.3 xtables

xtables is a generic name for the kernel module containing the shared code of iptables for different protocols like ipv4 or ipv6. While it was overviewed in section 3.2.1.4, this section introduces more details, in particular regarding the context of smart routing.

Regarding IoT6, the focus is on ipv6 but the basic functioning scheme is the same. The packets are forwarded from one chain to another:

- **PREROUTING**: initial chain, no decision about where to deliver the packet has been taken
- **INPUT**: the packet will be delivered to the current host
- **FORWARD**: the packet will be forwarded to the network but routing decision is not taken yet
- **POSTROUTING**: the routing has been taken before forwarding the packet on the network
- **OUTPUT**: the packet has been created by the current host itself

In addition different tables can be applied to one or more of these chains. The default one is:

- **Filter**: useful to apply packet filtering like ACCEPT or DROP
- **Mangle**: useful to modify packet
- **Raw**: mainly used to add some tags on packet before being processed by other tables which can interpret them like NOTRACK to discard connection tracking
- **Security**: mandatory access control purposes

As highlighted before, a *Nat* table exists for IPv4, but not for IPv6, which provides NAT related functionalities (IP masquerading, port translation, etc).

It is also possible for a user to define a specific chain. For details, the reader is encouraged to visit [http://www.netfilter.org](http://www.netfilter.org). Figure 10 illustrates the path that a packet follows.

High level smart routing, as described before, decides what is the most suitable of the packet destination based on the IPv6 payload or on the flow label information if the latter has been properly set as explained in the previous section. Hence, this basically corresponds to modify the destination IP address of a packet. However, the *mangle* table does not allow such modifications as this is usually done in the context of the *NAT* table. Therefore, for doing
IPv6 destination address modification, extensions are needed.

Xtables-addons\(^7\) is a set of extensions for iptables which appeared in 2008 and has replaced the previous patch mechanisms of netfilter (patch-o-matic). An interesting extension is the RAWDNAT target which allows making modifications when using the raw table. It simply allows modifying the destination IP addresses. For example, to reroute the traffic to A towards B, the rule should be:

```
ip6tables -t raw --I PREROUTING --d A --j RAWDNAT --to-destination B
```

We have chosen the PREROUTING chain in order to reroute the traffic even if it was initially targeted to the machine where the command is executed. In such a case, POSTROUTING or FORWARD would have received the packet.

In addition, the machine where this command is executed has to work as a router, which can be qualified as smart. Usually, default policy does not allow a machine to act like that and it thus cannot forward any packet. The easiest way to correct that is to change the default policy:

```
ip6tables -P FORWARD ACCEPT
```

However, a more cautious mode keeping the default policy as DROP is often considered as better. This implies that packet forwarding has to be enabled under certain conditions. For instance, assuming that A is a host that can use the smart routing functionalities, the filter table can be updated accordingly:

```
ip6tables -t filter --I FORWARD --s A -j ACCEPT
```

A can also represent an entire subnetwork. Other kinds of constraints can be specified. Another standard approach is to forward only packets arriving on a specific interface. For instance, assuming that connected equipment which can request smart routing are connected through the interface devA, the rule has to be specified as follows:

```
ip6tables -t filter --I FORWARD --s A -j ACCEPT
```

For additional options, the manual pages of ip6tables are well documented.

Moreover, forwarding can be also disabled by the system, which is the case in most of the Linux distributions. It is saved as a flag in the kernel which can be dynamically modified by the following command:

```
echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
```

This sets 1 (true) for IPv6 forwarding. The mentioned path is the standard one used in many Linux distributions but can be subject to change in specific Linux distributions. It is also important that this flag remains set even if the system or the network is restarted. An option is to use the sysctl command:

```
sysctl -w net.ipv6.conf.all.forwarding=1
```

or by editing its configuration file /etc/sysctl.conf:

```
net.ipv6.conf.all.forwarding=1
```

Using all configures all interfaces but a fine-tuned configuration can specify the network interface.

\(^7\) http://xtables-addons.sourceforge.net/
D3.2 Smart Routing Mechanisms Design

3.2.3.1 u32 module

The previous details show how forwarding packets while modifying the destination is possible. However, the smart routing decisions are taken according to the information carried by the packet (either in the headers’ fields or in the payload). A first solution is to use the u32 match options of ip6tables. It allows extracting 4 bytes of the packet. The general syntax is –m u32 --u32 “Start&Mask=Range” where:

- Start is the position offset in the packet bytes (starting at the beginning of the network layer)
- Mask is a mask to apply on the extracted value, which is thus symbolized by the “&” symbol
- Range is the value to verify

Hence if the test (=) is true, the rule is triggered.

The flow label is starting from the bit 12 to the bit 31. Therefore, it is completely enclosed in the first 4 bytes (Start = 0) and the mask to extract only the flow label is 0x00 0F FF FF in hex format which can be written in a compressed way 0xFFFFF.

Assumes that for every flow label, the smart routing strategy, smart(label), returns a specific address and that this strategy is applied to every packet sent to a machine A, the ip6tables smart rules has to follow the following pattern:

```
ip6tables –m u32 --u32 “0&0xFFFFF=label” –t raw --I PREROUTING --d A –j RAWDNAT
--to-destination smart(label)
```

Obviously, functions cannot be applied in ip6tables rules and the previous rule has to be considered as an abstraction. The smart strategy has to be defined prior and translates into several ip6tables rules either manually or automatically by a script. ip6tables can be also dynamically added, removed or updated.

As a final example, assuming that the server B will answer the request with the flow label 0xBB and C the request with the flow label 0xCC, the rules to create are:

```
ip6tables –m u32 --u32 “0&0xFFFFF=0xBB” –t raw --I PREROUTING --d A –j RAWDNAT
--to-destination BB

ip6tables –m u32 --u32 “0&0xFFFFF=0xCC” –t raw --I PREROUTING --d A –j RAWDNAT
--to-destination CC
```

Considering now the case where the flow label was not properly assigned, the objective is to extract valuable content in the payload itself. Using the u32 match is still possible however this needs:

- Further check regarding potential optional header extensions has to be made also with u32
- Multiple u32 checks if the content is more than 4 bytes

u32 is flexible enough to address these issues because it allows us to combine several tests like -m u32 --u32 “Start&Mask=Range&&Start&Mask=Range&&…”.

More advanced examples with bit shifting or use of previously extracted value can be found at: http://www.netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO-3.html#ss3.21

3.2.3.2 string module

Another option for an easy content based filtering is to use the string module. It aims at
searching for a specific string in the packet with the following options:

- the starting position offset
- the ending position offset
- the string to search (either in text or hexadecimal format)
- the search algorithm
  - bm: Boyer-Moore [7]
  - kmp: Knuth-Pratt-Morris [8]

Assuming the previous example with the strategy `smart(label)`, the `string` module can be still leveraged for flow label analysis:

```
ip6tables --m string --from 2 --to 4 --hex-string "[label]" --algobm --t raw --I PREROUTING --d A --j RAWDNAT --to-destination smart(label)
```

The | symbol is necessary to delimits the hex string like “[0A]”. As illustrated, we consider only the flow label from bits from 15 to 31 in order to not do string comparison with the end of the traffic class field. It is one drawback of this method. Also string lookup is more computational. Therefore, the `string` module can be used for tests but preference should be given to `u32`. However, it can be easily used for looking for long text label in the payload as for example:

```
ip6tables --m string --from 48 --string "Here is my very long label that the smart router has to match" --algobm --t raw --I PREROUTING --d A --j RAWDNAT --to-destination B
```

In this example, the smart decision is taken based on the payload of the IPv6 packet starting at the 48th bytes. No end offset is given meaning that the full payload is analyzed.

### 3.2.3.3 Integration

For integration and operational purposes, xtables solution is a good candidate for the following reasons. Firstly, netfilter/iptables is a long running project which is implemented for Linux kernels from 2.4. There is an active open source community. The latest version of iptables is 1.4.17. Many famous distributions integrate netfilters and iptables within their packaging which ensures easy updating even if it is not the latest version. In fact, u32 is supported until version 1.2.8 (2003). Secondly, xtables-addons development started in 2008 and is now included in many Linux distributions. Compilation is also an easy step as both iptables and xtables are written in C using common libraries. More especially, the RAWDNAT functionality was released in April 2009 under the version 1.5 followed by several updates including bug fixes. However, this functionality is now stable as the last major fix was in July 2010 (v1.28).
3.2.4 NPTv6 / RFC6296

It is also important to note that the community has also pointed out the lack of a NAT support for IPv6, which can be still needed for specific purposes, even in IPv6. Due to that, RFC6296 [35] proposes a mechanism for IPv6 prefix translation. After some testing, existing implementations (MAP66, NFNAT66) are partial or not mature enough to provide the necessary functionalities for smart routing and to be integrated into the smart board without huge efforts, unlike the iptables solution.

3.2.5 Conclusion

Iptables/xtables clearly enables the high level smart routing envisioned in IoT6. As highlighted, it has been successfully tested on the smart board. However, one key aspect is to encode the smart routing label into a proper location in the packet. While the flow label was designed for such QoS functionalities, using this header seems nowadays unsuitable in Internet as this will be probably altered (usually set to 0) by the routers of common ISP.

Therefore, using flow label should be mainly considered for local scenarios (Sensornet / Localnet) while smart routing which will traverse Internet should be based either on the content or on the TC field. Using the TC field could be a better option to speed up the packet analysis but it may already be reserved for "more traditional" QoS technologies like DiffServ, while flow label should not interfere with those. Obviously, changing the way packets are routed at the Internet level is beyond the scope of the project.

---

8 http://sourceforge.net/projects/map66/
9 http://sourceforge.net/projects/nfnat66/
### 3.3 CCN Based Smart Routing

A Content-based network is a communication network that features a new advanced communication model where messages are not given explicit destination addresses, and where the destinations of a message are determined by matching the content of the message against specific predicates (“interests”) declared by nodes. Routing in a content-based network aims at propagating these predicates to create a path for the return data and the necessary topological information in order to maintain loop-free and possibly minimal forwarding paths for messages [18]. Content-based communication is a communication service whereby the flow of messages from senders to receivers is driven by the content of the messages, rather than by explicit addresses assigned by senders and attached to the messages. Using a content-based communication service, receivers declare their interests by means of *selection predicates*, while senders simply publish messages announcing the reachable contents in a first time and then simply answer to data requests. The service consists of delivering to any and all receivers each message that matches the selection predicates declared by those receivers. The best way to provide a content-based communication service is as a datagram-based, connectionless service, through a content based network.

Therefore, content-based routing is a natural candidate for smart routing, which objective is to take routing decision based on the message content. In this section, we leverage CCN, one of the most advanced and usable technology for content-based routing. However, this arises some issues. Firstly, such a routing scheme is mainly designed for content retrieval systems and so envisions on-demand delivery mechanisms while smart routing applications are more focused on push mode dissemination where producer forwards updated information to different consumers as envisioned in the generic scenario of section 2.4. Secondly, such a scenario, where data is pushed, corresponds in majority to WSNs (Wireless Sensor Networks) or other low power devices. Therefore, guaranteeing a proper service with minimal overhead (computational and networking resources) is essential.

The IoT could benefit from CCN through the following relevant functionalities:

- **High-level smart routing**
  - Native pull-based communication scheme of CCN
  - A new push-based communication scheme for CCN in order to ease the diffusion of event notification, which is more adapted to the envisioned usage of smart routing

- **Low-level smart routing**
  - A strategy for forwarding sampled values with a minimal overhead

This section firstly motivates the use of content centric networking regarding the general context of IoT before describing CCN.

This work was also accepted for being presented in the Network of Future (NoF) conference in October 2013 [53].

#### 3.3.1 CCN-based IoT

IoT aims at enabling smart services through the interconnection of various things having computational resources. Smart applications usually rely on contextual information like the geographical location or other physical sensors. For example, temperature sensors may automatically trigger alarms or presence sensors can detect physical intrusion, etc. Therefore, a core component of IoT is the sensors or in a more general way information producers. WSNs (Wireless Sensor Network) are usually connected to Internet through gateways [9,10].

The information might be produced in two different modes: either on demand when another
D3.2 Smart Routing Mechanisms Design

entity requests it or proactively to a set of subscribers, which in the most general form is a broadcast diffusion. In addition, information sent may be useful for different applications and/or entities. In this section, we consider a generic scenario where a producer has to send information to multiple consumers.

IoT components include a lot of low power devices like sensors or actuators. Evidently, this raises scalability and low power efficiency challenges similarly to WSNs. Components may be also very dynamic by moving, disappearing or appearing which thus needs some fault tolerance mechanisms as well. For efficient dissemination information in such an environment, routing protocols coming from the WSN community have been proposed to address these challenges [11]. In particular, two major categories can be identified: data-centric and hierarchical protocols. Hierarchical approaches, as for example [12], promote to group together nodes sharing some properties and to elect a leader to serve as a kind of proxy with other leaders. Data-centric network like [13] advocates for a name based routing where data follows the reverse path of the corresponding query by a hop by hop mechanism. Thus, this alleviates the need of a specific network layer and dedicated protocols. In addition, service discovery protocols to retrieve the address associated to a content are obsolete with such an approach. Hence, data-centric approaches should help in saving resources. However, one major drawback identified in [11] is the potential side-effect of large flat naming scheme in large scale networks.

The data-centric also known as data-oriented networking has continued to be researched mainly under the term Information Centric Networks. Regarding the latter, several architectures have been proposed of which CCN is one representative. Here, we promote the usage of CCN for IoT for the following reasons. First, CCN implementations are already available on both standard computers or on Android phones, CCNx[14], as well as on Contiki[15], one major operating system for low power devices. Therefore, it is a good candidate in the context of IoT as it can be deployed on heterogeneous types of devices. Secondly, CCN combines the advantage of the data-oriented approach and hierarchical routing by leveraging a hierarchical naming scheme and therefore aggregation on routing information. It then integrates security mechanisms by design which is nowadays essential.

Finally, CCN has been proven compatible with current IP infrastructures so that an incremental deployment is possible. This means that CCN data can be carried over a regular IP network when needed, but also that IP(v6) packets can be carried over CCN. So, there is no a priori incompatibility with IoT architectures like the one envisioned by IoT6. Part of the communications (most probably within the Sensornet and/or the Localnet) could use CCN up to a dual stack gateway, while the rest of the architecture would remain unchanged.

3.3.2 CCN basics

CCN is designed for distributed environments where each node plays a role in the routing decisions. The basic routing scheme can be summarized as follows: a node interested in a given content (consumer) sends an interest which is forwarded thanks to the routing table until another can satisfy such request (producer) by sending a data message. This message is then forwarded using the reverse path. To achieve that in an efficient and scalable way, each CCN node maintains vital information in three tables highlighted in Figure 11:

- the Pending Interest Table (PIT) tracks interest that the node has forwarded and are still not satisfied yet, waiting for the proper content in response;
- the Forward Information Base (FIB) maintains the next hop information used to forward interests, based on content’s announcements;
- the Content Store (CS) represents content which is actually stored or cached on the local machine and which can be directly provided.
CCN also introduces the concept of faces. Faces can be anything capable of serving as a medium for transmitting and receiving the two types of messages illustrated in Figure 12. So, when a content $c$ is requested by an interest $i$ on a current node $n$, received from another node source on the face $f$, the steps are as follows.

1. the requested content $c$ is looked up in the CS. If it exists, the content is stored on the local node and the latter can thus satisfy the interest $i$ by sending a data message to using the face $f$ where the interest was received. CS serves also as a cache and allows a fast retrieval of currently popular demands using a Least Recently Used caching policy.

2. if no match is possible with the CS, the node executes a lookup in the PIT. If there is a corresponding entry, this means that the node has already forwarded a similar interest $i$ and is waiting for the response, so $i$ is not forwarded anymore. In addition, the PIT records the faces where the interests were received. If the new interest $i$ is coming on a new face $f$, the latter is added to the list of faces associated to $c$ in the PIT in order to reach all the consumers when the data will be available.

3. if no match exists in the PIT, the FIB is similar to a routing table in IP by returning all potential faces where the current interest $i$ should be forwarded regarding the content $c$ which is searched. If no face is returned, there is no possible route and so the searched interest cannot be satisfied or a default routing policy can be tried. Otherwise the interest is forwarded to the proper face and an additional entry is created in the PIT for the content $c$.

The data message sent by the content provider is forwarded back using the PIT table as it contains the faces where the corresponding content was requested. While all this process is applied when an interest arrives, the same applies when the source node is searching for the content. So, the tables will be queried and updated in the same order. For instance, it is possible that a node looking for a content it has already in its cache, i.e. in CS, due to a previous data forwarding.

---

**D3.2 Smart Routing Mechanisms Design**

![Figure 11 CCN Forwarding Engine](image)

CCN Forwarding Engine

<table>
<thead>
<tr>
<th>Content Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>/uni.lu/videos/intro.avi/v3/s0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pending Interest Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
</tr>
<tr>
<td>/uni.lu/videos/intro.avi/v3/s1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Forward Interest Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix</td>
</tr>
<tr>
<td>/uni.lu</td>
</tr>
</tbody>
</table>

Face 0 → Wireless device
Face 1 → Ethernet device
Face 2 → Application

Figure 11 CCN Forwarding Engine
D3.2 Smart Routing Mechanisms Design

More precisely, data can only "satisfy" a specific interest if the content name of the interest message is a prefix of the Data message. CCN names are defined in [16] as "opaque, binary objects composed of an (explicitly specified) number of components". This structure allows a fast and efficient prefix-based lookup similar to the IP lookup currently used. This new routing paradigm is based on hierarchical naming as shown in Figure 13 instead of regular host IP addresses, making names directly intuitive, and not needing an indirection mechanism between names and contents like DNS. An example of this hierarchical naming structure used in FIB is presented in Figure 13 for a content item named ccnx:/uni.lu/videos/intro.avi. To fill out the FIBs, nodes have to announce their content which is propagated to all other nodes with a limited overhead using aggregation. A typical example is the router of an organization does not announce individual content but only a prefix like ccnx:/uni.lu.

In terms of security, data is always authenticated by its provider [16] which is necessary as there is no end-to-end connections, the security consists in encrypting every content rather than a specific path or connection.

3.3.3 Problem definition

A common IoT deployment is illustrated in Figure 14 where there are things interconnected within a dedicated network as for example using IEEE 802.15.4/Zigbee [17]. Thanks to gateways, such a network is connected to Internet building so the Internet of things. From the Internet side, the standard devices are present like user computers, servers or databases but things tend also to be more and more directly connected, especially using cellular networks,
D3.2 Smart Routing Mechanisms Design

like smartphones or connected cars.

As envisioned in the previous section, we promote the use of CCN for low power devices. As CCN tends to make every node as a router, it is also possible to scale down the number of intermediate equipment (routers, relays) by enhancing neighbor to neighbor communications as shown in Figure 15. In addition, CCN can also be used partially in the Internet side, in particular on local networks, which are also in the scope of IoT6.

However, CCN is envisioned mainly for low power devices as shown on the left hand in Figure 15. As highlighted before, CCN is well designed for distributed environment. Data-centric approaches are helpful for saving resources while the issue of a flat naming scheme is resolved with CCN. In addition, the mechanisms of CCN allow the routing of interests on multiple paths which thus helps to improve the quality-of-service. Caching mechanisms helps in improving the availability of popular content and, in the meantime, limit data retransmission. That is why the use of CCN is advocated in this section.

These low power networks are usually composed of many information producers (like sensors) which are requested by other low power devices (like actuators) and devices in Internet (servers, smartphones, etc.). Especially, data is usually valuable for different applications and users. Therefore, a single information producer may have several information consumers. In such a case, the information consumers can desire different granularity. As an example, a temperature sensor can send information to a fire detection system at a high frequency, every 10 seconds, but such information provided only every 5 minutes is enough for automatically adjusting the heating system.

Therefore, the information should not be multi-cast to every consumer at the same regularity to limit the communication overhead in low power networks. Sending information temperature every 10 seconds during five minutes represents an overhead of 29 useless messages which can be forwarded through multiple paths and so consume resources.

The general scenario is depicted in Figure 16 where there is an intermediate CCN node Ri which has to forward the information from M producers transmitted by a previous router like Ri-1 towards K destinations which are accessible through other downstream nodes like Ri+1. While this depicts the general scenario, upstream and downstream routers can be also producers or consumers as highlighted before. To optimize the overhead of messages forwarded by Ri, we consider a sub problem that we qualify as local regarding one router and
D3.2 Smart Routing Mechanisms Design

one producer. Resolving the more general problem with multiple producers consists in
resolving multiple times the previous problem.

Formally the problem is defined as follows. Assuming (1) an intermediate node, a router \(R_i\),
(2) one producer \(P\) able to send an information \(inf\) every \(x\) seconds, (3) \(N\) consumers
\(C = \{c_1, \ldots, c_N\}\) and (4) that each consumer \(c_i\) is interested to have \(inf\) every \(x \times i\) seconds
where \(i_i \in \mathbb{N}\) (sampling period), the goal is to minimize the number of messages sent by \(R_i\).
Since each node has a limited capacity, this optimization can only use a limited number of
additional resources \(C_{cap}i\). This assumes that a proportion of resources is already reserved for
the common CCN operations (with its dedicated optimization like the prefix aggregation),
which is equivalent in the different communication strategies envisioned in next section.

3.3.4 High-level Smart Routing

3.3.4.1 Pull-based communications

By definition, CCN nodes are stateful and only forward data on the back path if an interest
was emitted beforehand. Therefore, a basic usage of CCN requires that information
consumers directly request the producers as illustrated in Figure 17 where \(C_1\) requests a
content of \(P\) through a multi-hop path composed of several nodes/routers.

This strategy is called pull and allows CCN to send data only when needed (on demand
delivery) but the initial interest packet represents an overhead. However, an interest is not
forwarded if a previous similar one has been already sent, and a given data message is only
sent once per face which is then multi-casted by design.

The producer also needs to initialize the FIB of the CCN nodes by announcing the hosted
content. Such a step is not considered in the overhead evaluation as this initialization is
required only once.

Assuming the high level smart routing definition, the name based routing makes the problem
easier as the information sending (\(P\) representing the sensor) does not need to know the
receiver itself. It is the responsibility of the users/devices interested in a content to request it.
To illustrate this we assume the generic scenario described in section 2.4. In a CCN word, the
information producer should announce a content like ccnx:/roomA/alarm. Then each,
interested consumer should request such a content to retrieve the current temperature in room
A.
D3.2 Smart Routing Mechanisms Design

To adapt it to the smart routing use case, aggregation is used so that the producer announces only `ccnx:/roomA/alarm` but can provide two contents:

- `ccnx:/roomA/alarm/high`
- `ccnx:/roomA/alarm/low`

Then the content itself can for example precise information about the alarm (not only high or low).

However, due to cache mechanisms, the producer which requests for example `ccnx:/roomA/alarm/high` might be able to retrieve such a content even if the alarm is now low. To circumvent this problem, two solutions are practicable:

- define the caching policy such that information sent by the producer P is never cached in the CCN nodes
- use versioning in content names as highlighted in Figure 13. Thus, a consumer who previously received a version v3 will then request `ccnx:/roomA/alarm/high/_v4`. If the alarm is no longer low, such a content cannot be retrieved and so avoids receiving obsolete information

### 3.3.4.2 Push-based communications

In the following discussions, the targeted router for sampling optimization purposes is \( R_i \) and the considered producer is \( P \). For sake of clarity, only one content is supposed for this producer, named \( P_{\text{name}} \), but the approach can be easily applied in parallel to different contents.

Pushing data over CCN was envisioned in [18] which describes potential solutions and finally ends up designing a publish-subscribe mechanism where a node interesting in a certain content can subscribe for it, but in this case the diffusion is still based on IP. In IoT6, we consider a strategy, push, where data is directly transmitted using the FIBs. In fact, the subscription is done in the same way that the registration of contents populating the FIBs and the sensors' data dissemination is similar to the propagation of interests. In particular, sensors' data are not cached and corresponds to a one-way message. To do so, a simple option is to disseminate such very small and ephemeral data directly inside interests like for example `/roomA/temperature/ts=10/value=20` assuming that the temperature sensor has registered the content `/roomA/temperature/`. Such an interest is forwarded using the FIB but without creating any entry in the PIT, since no data will be sent back. This can be easily done using a flag in the content name or in the interest header. However, a main advantage of creating an entry in the PIT, in the common usage of CCN, is to avoid routing loops. Unlike [18] that use standard IP unicast for avoiding routing loop, we propose to use a timestamp, \( ts \), in the content name in conjunction with a new field in the FIB called `last_seen_version` in order to route only the latest content, which is the most valuable. Every new transmission for a given content will be checked against the `last_seen_version` value. More recent data will be forwarded while older obsolete ones will be discarded, thus avoiding loops and saving bandwidth. Evidently, there is a risk to drop packets arriving in a disorder but, regarding the context envisioned by using CCN, forwarding an old information which has been already set obsolete by another already transmitted and received (`last_seen_version`) is useless. Another option would have been to keep track of more than one latest timestamps which thus allows some disordering.

Hence, we assume our aforementioned mechanism or the one described in [18]. This corresponds to `data` messages forwarded in Figure 18. However, since the information is transmitted regularly independently of the demands, each value out of the sampling period \( (i_t) \) of a consumer \( C_i \) is dropped, which represents an overhead of useless messages. Both of these
strategies use common CCN mechanisms and the pushing mode only requires one additional flag and associated test. Therefore, the additional resources, $C_{api}$, can be neglected.

![Figure 18 Subscription based mechanism (push)](image)

### 3.3.5 Low-level smart routing

#### 3.3.5.1 Optimal Forwarding Strategy

To reduce the message overhead in an optimal way, the goal is to combine the advantages of both pulling and pushing modes. Hence, the router should only forward data to $C_i$ in pushing mode emitted at the period $t$ only if the consumer $C_i$ has expressed its interest regarding this period: $t$ being a multiple of the sampling period $i$. This implies that when the consumers register themselves, they have to specify their sampling period such that $R_i$ can keep track of that by having a dedicated counter. This is shown in Figure 19 where $C_1$ is registered to $P_{name}$ with $i_1 = 2$. As highlighted, the counter is directly inserted in the FIB and updated when each data message arrives.

![Figure 19 Optimal forwarding strategy](image)

However, this optimal strategy assumes to have one counter per registration which can be implemented as individual counters (with testing if equal zero) or a global counter (with multiple modulo tests). The additional resources $C_i$ are considered in terms of number of counters, independently of the implementation. To limit the number of messages, identical messages are combined into one. Assuming $i_1 = 2$ and $i_2 = 3$, the sixth message produced will not be forwarded twice as $D_1$ and $D_2$ but only once as $D_{1+2}$. The different notation of the same message just helps in identifying the purpose of the message. In fact, all these messages are exactly the same and will be forwarded to both consumers thanks to the propagation mechanism of CCN, which makes feasible the use of a single message $D_{1+2}$. This simple optimization is applied to the ones described in the next section as well.

The optimal strategy cannot be applied with a fixed $C_i$ value because the number of registrations can be very large. While not being practical, this strategy is considered as the baseline for assessing the other ones.
3.3.5.2 Smart forwarding strategy

Assumes a fixed maximal number of counters $C_i$, the goal of a smart forwarding strategy is to refine the pushing mode while limiting the number of forwarded messages.

The extreme case considers a single counter which serves all subscribed periods. For example, with multiples like $i_1 = 2$ and $i_2 = 4$, a unique counter looping between 1 and 2 is enough and does not entail additional messages. This can be extended using the Greatest Common Divisor (GCD) (strategy $gcd$). With $i_1 = 4$ and $i_2 = 6$ as in Figure 20, the unique counter loops over $gcd(4, 6) = 2$ which thus avoids to forward message at every period. However, there is an overhead of useless messages like the first forwarded message $D_{1+2}$ which is then transferred by other routers as $D_1$ or $D_2$ before being dropped by the consumers. Moreover, there is a high risk that with numerous subscriptions, the global GCD reaches 1 which does not represent any improvement compared to the basic pushing forwarding scheme.

With more counters, i.e. $Cap_i > 1$, this can be extended by selecting $Cap_i$ values, $G = \{g_1, ..., g_{Cap_i}\}$ such that:

$$\forall C_i \in G, \exists g_k \in G, \alpha \in \mathbb{N}, g_k \times \alpha = i \_i$$

Figure 21 considers two counters and so introduces a specific structure, the Information Sampling Table (IST), which keeps track of them. With three consumers and $i_1 = 9$, $i_2 = 12$ and $i_3 = 5$, $R_i$ will forward every 3 and 5 data messages. Again, this will entail some message overhead but it is better than having considered the unique GCD which is one.
After having filtered identical intervals and those being multiple of others, the smart forwarding strategy affects one counter per unique remaining sampling interval $G' = \{g_1, ..., g_M\}$. If the number of needed counters is acceptable, i.e. $|G'| \leq Cap_i$, the algorithm stops and $G = G'$. Otherwise it iterates until this condition is reached by merging at each step two counters, $g_{merge} = \gcd(g_i, g_j), (g_i, g_j) \in G' \times G', g_i \neq G'$. These counters are selected to maximize the GCD in order to limit the message overhead:

$$\forall (g_k, g_l) \in G' \times G', (g_k, g_l) \neq (g_i, g_j), \gcd(g_i, g_j) \geq \gcd(g_k, g_l)$$

### 3.3.6 Evaluation

The evaluation assesses the number $msg_s$ of messages forwarded by the router $R_i$ regarding the chosen strategy among pull, push, optimal, gcd or smart and a configuration defined by:

- $MAX_T$: the maximal sampling interval that a consumer can register,
- $N = |C|$: the number of consumers as defined before but which register a unique sampling interval. The latter is chosen randomly between 1 and $MAX_T$,
- #counters: the number of available counters (only impacts for the smart forwarding strategy),
- #steps: the number of steps assuming that the initial sampling period of the producer is one step.

Based on preliminary experiments, #steps was fixed to $5 \times MAX_T$, which thus guarantee that every consumer has received data at least five times during the discrete events simulation. In fact, the optimal strategy is used as the baseline to compute an overhead ratio defined as:

$$f_s = \frac{msg_s}{msg_{optimal}}$$

Eq.1

The number of consumers is an important parameter which is evaluated in Figure 22 by plotting the average of $f$ over 10 experiments for each configuration. In this figure the number of consumers is expressed as a proportion $p$ of $MAX_T$ which has been fixed to 3000.
D3.2 Smart Routing Mechanisms Design

# counters was set to 100. The usual CCN mechanism (pull) highlights a ratio of 2 which is coherent since every data is sent after an initial interest.

When $p$ increases, all other curves tend to one because the optimal strategy also leads to forward more messages due to more consumers. In particular, when $p$ increases, the chance to have a consumer $C_i$ registering an interval $i_i = 1$ is higher which thus leads to forward a message at every step. Due to the same mechanism, the benefit of smart forwarding compared to the push mode is reduced. Therefore, using optimization has logically a higher impact when the number of registered intervals is low. However, the worst case for smart routing is for $p = 0.08$ representing 240 distinct intervals which are handled by the 100 counters. As highlighted, the gcd strategy leads to the same result that the simple push strategy which forwards every data message because the overall GCD computation reaches always 1 even with $p = 0.05$ due to the probability to have randomly selected a prime numbers (430 between 1 and 3000).

The smart forwarding strategy proposed in IoT6 always exhibits a ratio lower or equal to others. It varies between 1 and 1.67. When $p$ is very small, there is enough counter to be optimal and starting at $p = 0.04$, the optimization starts by having shared counter for multiple intervals which entails a message overhead. Around 0.1, the optimal strategy has to forward many messages which thus impacts highly on the denominator in equation Eq.1.

![Figure 22 Impact of the number of consumers](image)

In Figure 23, the number of counters for the smart forwarding strategy varies on x-axis and different maximal sampling intervals have been tested. The number of uniquely registered interval is set to $0.05 \times MAX_T$ and so varies between 100 and 1500 depending on the configuration. When the number of usable counter increases, the overhead is logically reduced. It is even faster if there are less registered intervals (lower $MAX_T$). Such an experiment helps to choose the right trade-off between message overhead and hardware/software additional resource (number of counters). Then depending on the physical communication layer and the hardware, it could be possible to select the most efficient number of counters in terms of energy consumption. For instance, assuming that the maximal acceptable overhead is expressed as $f = 2$, the proper number of counters can be chosen regarding the number of uniquely registered intervals.
3.3.7 Conclusion

As previously mentioned, an important missing feature to use CCN for IoT is a push-based mechanism for communications. Instead of probing a sensor regularly, which consumes energy, a better way would be to let the sensor transmit its data when needed. Doing this in CCN is not natural but yet feasible. Van Jabobson et al. solved a similar issue to enable CCN-based conversational communications like voice over IP [19]. They propose a solution based on two properties of CCN: on-demand publishing to request content that has not been published yet, which simulates a rendez-vous point, and a deterministic algorithm followed by the two parties to find this service name. The approach in [18] handles small ephemeral messages, more precisely events communication typically produced by IoT. This allows CCN to support publish/subscribe communications by adding a new unicast forwarding table and a new set of messages. In particular, one way messages for event notifications can follow the path of subscriptions up to the subscribers. In [15], a full CCN layer is provided for Contiki which enables CCN for low power devices and IoT.

Our approach also deals with the problem of efficient dissemination of event notifications in a sensor network. Some works are close to this topic even if they do not optimize the diffusion of information according to fixed sampling rates. For example, the authors in [20] used non-linear optimization theory to optimize the rate at which sensors can be probed by a control system under network limitations. Another way to save network resources in WSN is to rely on in-network aggregation techniques which have been largely investigated through different strategies [21], even secured ones [22]. This solution is particularly efficient when data is gathered by a few sinks while we envision in our case a more neighbor-to-neighbor communication pattern between sensors.

In this deliverable, CCN is leveraged for distributed IoT communications, especially in the context of producers regularly updating the information sent to multiple consumers. A new push-based mechanism is adapted and optimized. Even under strict resources constraints, the nodes optimize message forwarding based on the update frequencies required by the consumers. Compared to the optimal but unrealistic approach, the results show a limited
D3.2 Smart Routing Mechanisms Design

overhead, always under the native CCN pull approach.
3.4 OpenFlow Based Smart Routing

Flow routing is another interesting approach to investigate in order to achieve smart routing. Flow routing handles flows (data streams) as the single meaningful end-to-end activities over the network instead of packets as in traditional networking. For this reason, it is a good candidate for achieving high-level smart routing in the context of IoT6 project. In general, the special handling of data streams in today’s networks yields increased efficiency avoiding excessive latency and jitter for streaming data, such as VoIP.

IPv6 networks support direct flow handling, by the Flow Label in the protocol header, as described in section 3.1. The combination of IPv6 flow handling with the OpenFlow-based forwarding control can be an efficient solution to manage smart routing in future Localnets.

3.4.1 Introducing Software-Defined Networking (SDN)

Internet traffic has been increasing explosively due to growth in the number of Internet users and diversification of applications such as voice communication and video delivery. To efficiently handle such traffic, novel routing technologies are required. In this context, Software-Defined Networking (SDN), promoted by the Open Networking Foundation (ONF) [25], has emerged. In SDN, network control is decoupled from forwarding and is directly programmable. In other words, traffic control mechanisms (e.g., routing, bandwidth allocation, etc.) are implemented on software, and these features enable flexible and efficient network control.

In fact, the migration of control, formerly tightly bound in individual network devices, into accessible computing devices enables the underlying infrastructure to be abstracted for applications and network services, which can treat the network as a logical or virtual entity.

![Software-Defined Networking Architecture](image)

**Figure 24 Software-Defined Networking Architecture**

A logical view of the SDN architecture is shown in Figure 24. Network intelligence is (logically) centralized in software-based SDN controllers, which maintain a global view of the network. As a result, the network appears to the applications and policy engines as a single, logical switch. With SDN, enterprises and carriers gain vendor-independent control over the entire network from a single logical point, which greatly simplifies the network design and operation. SDN also greatly simplifies the network devices themselves, since they no longer need to understand and process thousands of protocol standards but merely accept instructions
D3.2 Smart Routing Mechanisms Design
from the SDN controllers.

3.4.2 OpenFlow: an overview

OpenFlow is the first standard communications interface defined between the control and forwarding layers of a SDN architecture.

Currently, in Ethernet architectures these two planes are located in the same device – the control plane maintaining network information and routing tables used to maintain connectivity and the data plane providing the interface for incoming and outgoing packets. This requires all networking devices to have access to the same tables and related information, which is usually updated manually as changes to the network are implemented.

As shown in Figure 25, under OpenFlow, the data plane remains in the switch, but the control plane is placed on a separate controller, with OpenFlow enabling communication between the two. By separating control and data forwarding, network configuration settings and updates can take place in software, which opens up the possibility of embedding network information and requirements on the application level. This allows networks to be configured and reconfigured on the fly, with little or no direct involvement on the hardware level, essentially providing virtual, abstract networking environment.

3.4.2.1 Routing in OpenFlow

OpenFlow is currently seen as one of the promising approaches that may pave the way towards the Future Internet. OpenFlow was first proposed in [26] as a way to enable researchers to conduct experiments in production networks. However, its advantages may lead to its use beyond research.

At its core, OpenFlow offers a higher flexibility in the routing of network flows and the freedom to change the behavior of a part of the network without influencing other traffic.

OpenFlow speeds up the forwarding and routing processes, making the control function independent of the hardware it controls.

OpenFlow is based on a packet switch, with an internal flow table and a standardized interface to add/edit/remove flow table entries. It exploits the fact that most modern switching equipment contains tables that run at line-rate and map incoming traffic to outgoing ports. While each vendor’s equipment is different, OpenFlow exploits a common set of functions
D3.2 Smart Routing Mechanisms Design

implemented in all switches, common to all distinct proprietary and open-source solutions. Moreover, the open standardized interface, i.e., the OpenFlow protocol, takes the control of the switches out-of-the-box and places it in the controller.

In fact, being based on the SDN architecture, an OpenFlow network includes a forwarding plane and a control plane. The forwarding plane consists of OpenFlow switches that are responsible for directly forwarding packets in the network on per-flow basis. The control plane is a remote controller, typically a standard server, which is responsible for managing OpenFlow switches in the network, and handling high-level routing decision.

As shown in Figure 26, each OpenFlow switch contains (i) one or more FlowTables, which perform packet look-ups and forwarding, and (ii) a Secure Channel that connects the switch to the controller. This secure channel is the communication channel which the controller uses to operate all switches in the network via the OpenFlow Switching Protocol. Each table contains Flow Entries which are used as rules to process matching packets. The controller can add, update or delete flow entries in the flow table and this is the way it controls flow traffic in the network. Each flow entry mainly consists of three components. First, a set of 12 fields with information found in the packet header is used to match incoming packets. Second, a list of actions dictates how to handle matched packets. Third, a collection of statistics for the particular flow, like number of bytes, number of packets, and the time passed since the last match.

When the OpenFlow switch receives a packet, it extracts the packet header information, and then it looks-up its flow tables to find matching flow entry. If matching entry is found, the switch applies instructions defined in the entry to process the packet and update statistics for the flow table entry.

If there is no matching entry (i.e., the incoming packet is not registered in the switch flow table), the switch sends a query to the controller. The controller may add a new flow entry into the switch’s flow table with instructions about how to process the packet (e.g., sending data by routes that are the faster or have the fewer number of hops, etc.) or simply drop the packet. By doing so, the controller determines the action of the switch, while imposing policies on the network traffic flows.

The entire process is illustrated in Figure 27.
It has to be noticed that while each packet is switched individually, all packets in a flow are switched in the same way, making the flow the fundamental unit of manipulation within the switch. Moreover, OpenFlow allows switching at different network layers to be combined. In fact, a flow can be defined as a combination of any of all the fields that make up a header. For instance, a flow can be defined as a combination of Ethernet and IP src/dst addresses, or a combination of an incoming port and the TCP/UDP src/dst ports. By doing so, the traffic for a given application can be routed differently from the rest of the traffic in the network. Thus, OpenFlow allows granularity and layering to be suited to the user/application’s needs and requirements.

3.4.2.2 IPv6 support in OpenFlow Switching Protocol

In December 2009, the Open Networking Foundation released the first version of the OpenFlow Switch Specification that describes the formats and protocols by which an OpenFlow Switch receives, reacts to, and responds to messages from an OpenFlow Controller. The objective of this dialogue is to instruct the forwarding plane of the OpenFlow Switch to treat incoming packets in a particular way.

Other versions have followed to this first one, all built upon previous releases, but including at the same time significant improvements. In particular, for the first time, support for IPv6 was included in the OpenFlow 1.2 Switch Specification [36]. In addition to the previous support for IPv4, MPLS, and L2 headers, OpenFlow 1.2 supports matching on IPv6 source address, destination address, protocol number, traffic class, ICMPv6 type, ICMPv6 code, IPv6 neighbor discovery header fields, and IPv6 flow labels.

OpenFlow 1.3 [37] is the latest update to the OpenFlow protocol. It describes the various port configurations, channel types and flow tables, as well as the relationships between these elements, to be used in OpenFlow-compatible switches.

Every OpenFlow switch contains a number of flow and group tables used in the packet forwarding process, as well a communications channel to an external controller. In this way, the controller is able to supply the switch with updates to flow tables and other pertinent information needed to maintain network pathways. OpenFlow tables support two kinds of pipeline processing between tables: OpenFlow-only and OpenFlow-hybrid, used for single-protocol and mixed environments respectively.

OpenFlow 1.3.1 also provides information on the port configurations needed to connect switches to each other. The protocol supports three types of ports: physical, logical and reserved. Physical ports correspond directly to a hardware interface, while logical ports exist on the higher, abstracted plane. Reserved ports handle generic forwarding instructions to non-
D3.2 Smart Routing Mechanisms Design

OpenFlow systems.

OpenFlow 1.3.1 includes a number of additions to the previous 1.2 spec. These include:

- Support for IPv6, that allows controllers to implement new routing deployment configurations and Request for Comment (RFC) specifications
- Tunneling and logical port abstractions that can be used in datacenter, Virtual Private Network (VPN) and other deployments
- Provider Backbone Bridging (PBB), which provides a lightweight tunneling method for datacenter-to-datacenter connectivity
- Enhanced per-flow metering and per-connection filtering techniques designed to improve data flow, bandwidth management and QoS

Obviously to provide the minimal IPv6 functionality (similar to the IPv4 one), the switch must be able to match on various IPv6 header fields, and execute actions on those fields.

OpenFlow 1.1 uses rigid match structures; and only allows the definition of new structures, but no extension of existing standard match structure. So for IPv6 a new match structure has been defined. This is pretty similar to the standard match structure, only the IPv4 fields have been replaced with IPv6 ones, including masks and wildcards.

Some of the standard actions have been reused for IPv6 processing -- such as TTL decrease --, but for some fields new actions have been defined; either because of the increased field size (IPv4 addresses vs IPv6 addresses), or because those did not exist in IPv4 (e.g. flow label).

### 3.4.3 Feasibility study for using IPv6 Flow label with Openflow

Figure 28 shows an example of OpenFlow-based IPv6 network where the source sensor nodes (i.e., information producers) communicates with the destination nodes (i.e., monitor system, alarm system, etc.) via OpenFlow switches. As explained before, OpenFlow switches can take their own action on the basis of header information of the incoming packet (according to the rules set by the controller and registered into the switches’ flow table). This allows controlling traffic flows on the basis of various routing granularities. In order to achieve high-level smart routing, we herein propose a scheme that uses IPv6 flows, identified by different IPv6 Flow label, as routing granularity.
In detail, the OpenFlow switches will not forward the incoming packets on the basis of the destination IP address, or IP subnet (classical approach used in networks using general routing), but they will forward the packets based on the value of the flow label field in the packet header. The suggested scheme assumes that each sensor node will have to set the flow label, based on the collected information, i.e., the value of the monitored parameter (e.g., temperature). Therefore, some thresholds will be fixed by the network manager in order to associate a given value of temperature to a given flow label. In the example in Figure 28, two different flow labels have been defined: FLA (Flow Label Alarm) and FLN (Flow Label Normal), assigned respectively when the monitored value of temperature is above or below a fixed alarm threshold. Of course, by defining more thresholds, and more flow labels, it is possible to achieve a finer routing granularity. The packets marked with FLA will trigger a call to the Alarm System, in order to take prompt actions in case of fire, or other alarming conditions. Instead, a packet marked with FLN, will be simply forwarded to the monitoring system, along a multi-hop path, according to the routing policies defined by the OpenFlow controller.

The suggested scheme implies the need of some intelligence into the information collectors, in order to translate the value of the monitored parameter into a flow label. This can be seen as a drawback. Actually it does not impact on the energy consumption of the sensor node, given that it is a simple check (comparison among two values, the monitored one, and the reference one, i.e., the threshold) and it does not requested heavy computation.

As described in Sec. 3.3.2.2 OpenFlow 1.3.1 includes support for IPv6, and in detail for IPv6 Flow Label. Several projects are working for implementing the new protocol specification. However, at the time OpenFlow was started to be investigated for IoT6 smart routing purposes, no open source implementation was available. Therefore, OpenFlow-based smart routing described in this section is proposed based on the features described in the specification but no real implementation was possible. It is important to note that most of commercial products do not support this new specification as well. Once, this gap will be filled, in other words, when software and hardware supporting the IPv6 Flow Label will be available, it will be possible to test the performance achievable with the suggested high-level
D3.2 Smart Routing Mechanisms Design

smart routing solution.

3.5 RPL Based Smart Routing

As shown in Figure 2, the IoT6 architecture includes IPv6 sensor networks, and in detail 6LoWPAN networks. Routing issues are very challenging for 6LoWPAN, given the low-power and lossy radio-links, the battery supplied nodes, the multi-hop mesh topologies, and the frequent topology changes due to mobility. Successful solutions should take into account the specific application requirements, along with IPv6 behavior and 6LoWPAN mechanisms [40]. An effective solution is being developed by the IETF "Routing Over Low power and Lossy (ROLL) networks" working group. Recently, it has proposed the leading IPv6 Routing Protocol for Low-power and Lossy Networks (LLNs), RPL, based on a gradient-based approach [41, 42, 43, 44].

RPL can support a wide variety of different link layers, including ones that are constrained, potentially lossy, or typically utilized in conjunction with host or router devices with very limited resources, as in building/home automation, industrial environments, and urban applications [45, 46, 47, 48]. It is able to quickly build up network routes, to distribute routing knowledge among nodes, and to adapt the topology in a very efficient way.

In the most typical setting entailed by RPL, the nodes of the network are connected through multi-hop paths to a small set of root devices, which are usually responsible for data collection and coordination duties. For each of them, a Destination Oriented Directed Acyclic Graph (DODAG) is created by accounting for link costs, node attributes/status information, and an Objective Function, which maps the optimization requirements of the target scenario. It is identified with a DODAGID. The topology is set-up based on a Rank metric, which encodes the distance of each node with respect to its reference root, as specified by the Objective Function. Regardless the way it is computed (see Section 3.5.2 for more details), the Rank should monotonically decrease along the DODAG and towards the destination, in accordance to the gradient-based approach.

RPL can encompass different kinds of traffic and signaling information exchanged among nodes (as well as ancillary data structures) depending on the requirements of the considered data flows.

The Multipoint-to-Point (MP2P) is the dominant traffic in many LLN applications. It is usually routed towards nodes with some application relevance, such as the LLN gateway to the larger Internet or to the core of private IP networks. In general, these destinations are the DODAG roots and they act mainly as data collection points for distributed monitoring applications. Contrariwise, Point-to-Multipoint (P2MP) data streams can be used for actuation purposes, by means of messages sent from DODAG roots to destination nodes. Finally, Point-to-Point (P2P) traffic is necessary to allow communications between two devices belonging to the same LLN, e.g., a sensor and an actuator. In this case, a packet will flow from the source towards the common ancestor of those two communicating devices; then, downward towards the destination.

As an obvious consequence, RPL has to discover both upward routes (i.e., from nodes to DODAG roots) in order to enable MP2P and P2P flows, and downward routes (i.e., from DODAG roots to nodes) to support P2MP and P2P traffic.

3.5.1 RPL Topology Formation

The simplest RPL topology is made by a single DODAG with just one root, e.g., a WSN monitoring a small size area.

A more complex scenario encompassed by RPL is composed of multiple uncoordinated DODAGs with independent roots, that is, the LLN is split in several partitions depending on
D3.2 Smart Routing Mechanisms Design

the needs of the application context.

A more sophisticated and flexible configuration could contain a single DODAG with a virtual root that coordinates several LLN root nodes. The main advantage in this case, with respect to the previous one, is the absence of limitations on the parent set selection, given that all nodes belong to the same virtual DODAG, although a stronger coordination is needed among the root nodes.

Depending on the application requirements, it is also possible to combine the three examples presented so far in more complex topologies.

Moreover, multiple instances of RPL may run concurrently on the network devices and each instance has specific routing optimization objectives, such as the minimization of delay and energy consumption. To this aim, a $RPL_{InstanceID}$ is employed to identify one of the possible RPL instances running on the same network.

The formation of all these possible kinds of topologies relies on the RPL information dissemination mechanism, which enables a minimal configuration in the nodes and allows them to operate mostly autonomously. In this sense, a key role is played by DODAG Information Option (DIO) messages, containing information about the Rank, the Objective Function, the IDs, and so on. They are multi-casted (periodically and link-locally) by each node to create the DODAG, thus establishing paths towards the roots.

In detail, according to RPL specifications, in order to implement network formation and management operations, all nodes execute several operations: they send and receive DIOs; they compute their own Rank, based on the information included in the received DIOs; they join a DODAG and select a set of possible parents in that DODAG among all nodes in the neighborhood; they select the preferred parent among the possible ones.

A node receiving a DIO message uses its information to join a new DODAG, or to maintain an existing one, according to the Objective Functions and the Ranks of their neighbors. It can also detect possible routing loops. To reach these goals, the following function is used:

$$DAG_{Rank}(N) = \text{floor} \{ \frac{\text{Rank}(N)}{\text{MinHopRankIncrease}} \}$$

Where $N$ is the node identifier, $\text{Rank}(N)$ is the Rank of node $N$, floor{x} is the greatest integer less than or equal to $x$; and MinHopRankIncrease is the implementation-dependent minimum hop rank increase value, representing the minimum difference between the Rank of a node and the Ranks of its possible parents.

Upon a DIO message is received from a neighbor, a node defines its own Rank to a value that is a function of both the neighbor Rank and the cost to reach the DODAG root through it. The considered node lets its set of possible parents contain only that neighbor, if one of the following conditions is true: (i) the node Rank was not already setup; (ii) the old value, $A$, of the node Rank and the computed one, $B$, verify the relation $DAG_{\text{Rank}(A)}>DAG_{\text{Rank}(B)}$. Instead, if $DAG_{\text{Rank}(A)}=DAG_{\text{Rank}(B)}$, the neighbor is added to the set of possible parents. In other cases, the DIO is not further considered. Finally, each node can select its preferred parent within its set of candidate parents based on several possible rules, such as Objective Function, path cost, Rank, etc.

On the other hand, a node advertises its presence, the affiliation with a DODAG, the routing cost, and the related metrics by sending DIO messages to nodes in its neighborhood, only if it has already computed its own Rank. An exception is allowed to the DODAG root, which is configured to get its own Rank equal to the value MinHopRankIncrease, and to send it with the DODAGID, the routing cost, and the related metrics into DIO messages. In this way, a DODAG is constructed in a widening-wave fashion, starting from the DODAG root.

It is worth to remark that these procedures are useful to establish upward routes only. Therefore, in presence of P2MP and P2P traffics, an additional mechanism is required to
create downward paths. To this end, RPL uses Destination Advertisement Object (DAO) messages to back-propagate routing information from leaf nodes to the roots. They are triggered by the reception of a DIO message, or in global and local repair operations. After receiving a DAO message, each node forwards it to its parent at the expiration of a timer, which is implementation-dependent [41].

To avoid redundancies and to control the signaling overhead, the *trickle* algorithm [49] triggers, for each node, a new DIO message only when the overall amount of control packets already sent in the neighborhood of that node is small enough.

### 3.5.2 RPL Control Messages and Metrics

RPL control messages are encapsulated into ICMPv6 packets, according to [50]. The structure of a message is reported in Figure 29.

![Figure 29 RPL encapsulation](image)

The Code field indicates which kind of control message is present after the Checksum. The Base field is the RPL message header and it contains only the basic information related to the functions of the carried object. Instead, the Options field is the body of such messages and, depending of the needs, it may be composed of any combination of optional functions (padding, metric containers, route information, DODAG configuration, RPL target, and so on).

Each RPL message has a secure variant providing integrity and protection as well as optional confidentiality and delay features.

Regarding the possible metrics (which can be fruitfully exploited for timely adapting the topology to changing network conditions) RPL can use [51]: node energy, hop count, link throughput, latency, link reliability, and link color. In particular, with the term “colors” RPL refers to specific properties of links, so that the link color is used to include or exclude such links from the paths.

This richness of information, from one side, makes RPL highly adaptable to different operating conditions. On the other hand, it is necessary to keep under control the adaptation rate of routing metrics in order to avoid path instabilities, which would severely impair LLN performance and scalability.

All the available metrics can be advertised in control messages.

### 3.5.3 The Objective Function

In RPL, the Objective Function translates key metrics and constraints into a *Rank*, which models the node distance from a DODAG root, in order to optimize the network topology in a very flexible way. Furthermore, the Objective Function allows the selection of a DODAG to join and the identification of a number of peers in that DODAG as parents. Generally speaking, the parent selection at a node could be triggered in response to several events, such
D3.2 Smart Routing Mechanisms Design

as the reception of a DIO message, a timer elapse, all DODAG parents become unavailable, or a trigger indicating that the state of a candidate neighbor has changed. After the Objective Function has scanned all the interfaces at a node to check whether they can be eligible for establishing a link in the topology, all candidate neighbors are examined to evaluate if they can act as RPL router. These preliminary operations are useful to exclude all those links and candidate nodes that do not match basic Objective Function compatibility rules, e.g., related to security issues, performance, etc. Then, the node scans the list of the candidate parents that passed the preliminary tests. The Rank that would result from having each of them as parent is evaluated. The preferred parent is elected as the one that can grant the smallest Rank, provided that this Rank is smaller than the one currently held by the node itself. Obviously, these operations can be iterated when more than one parent has to be selected.

The Objective Functions proposed by IETF are described below.

1) Objective Function 0

It requires only the information in the RPL DIO header. A node Rank is obtained by adding a normalized scalar, RankIncrease, to the Rank of a selected preferred parent. The RankIncrease value is a multiple of 0x100, so that Rank values can be stored in one octet. Given that in the RPL main specification [41] there is neither default Objective Function, nor default metric container, it might happen that two implementations, following different guidelines for a specific problem or environment, will not support a common Objective Function which they could interoperate with. Therefore, Objective Function 0 is designed as a common denominator among all the generic implementations. It ignores metric containers and it leaves to implementation the responsibility to compute how link properties are transformed into a RankIncrease.

2) Minimum Rank Objective Function with Hysteresis

It is designed to find the paths with the smallest path cost while preventing excessive churn in the network [52]. A node switches to the minimum cost path, NewPathCost, only if the following inequality is verified:

NewPathCost < CurrentPathCost - gamma

Where CurrentPathCost is the path cost of the current path, and gamma is the PARENT_SWITCH_THRESHOLD, implementing hysteresis.

This Objective Function may be used with any additive metric listed in [51] as long the routing objective is to minimize the given routing metric. Besides, it employs a DODAG parent set with only one node. This node is automatically chosen as the preferred parent. As a consequence, any candidate neighbor may become the preferred parent.

3.5.4 Conclusion

From the overview provided in previous sections, we can conclude that RPL is a good candidate as routing protocol able to provide low-level smart routing in IPv6 scenarios. In fact, first of all, it has been designed for 6LoWPAN networks. Thus, by design, it fulfills the needs of such low-power IPv6 networks.

Moreover, by properly defining metrics and object functions, it is possible to implement best effort and QoS mechanisms, which are able to fulfill the requirements of the specific applications, while building paths and routing graphs.
3.6 **ARCs Based Smart Routing**

Available Routing Construct (ARC) is a new routing algorithm designed for wireless sensor intra-network communications. This new paradigm was designed by P. Thubert and P. Bellagamba, from Cisco systems in October 2012 [30]. ARC is a two-edged routing construct. Its main advantage relies on fast recovery in case of network failures. Alternative routes can be retrieved in case of local traffic congestion (multipath routing and load balancing). Traditional routing is based on a forwarding scheme where a path, composed by an ordered list of network nodes is performed for each communication between a source and a destination. Thus if a simple failure appears on the path, the route becomes useless. In fact the path needs to be recomputed. This process adds delays in order to locally update the network routing tables impacted by the node or link breakage. Unfortunately a single failure on any new path needs also to trigger a re-routing calculation process.

ARC is based on the innovative concept of routing construct made of a sequence of nodes and links with 2 outgoing edges. As a consequence, each node can still reach one of the outgoing edges upon a single breakage. An ARC topology is resilient to one breakage per ARC. Thus each ARC provides its own independent domain of fault isolation and recovery. ARC can simplify and significantly improve the network utilization in wireless sensor networks (fast re-routing, load balancing) in respect to its fault-tolerance intrinsic properties.

### 3.6.1 Definitions

Before providing an in depth description of the ARC algorithm, in this section we introduce some definitions that are useful for the description of the algorithm itself.

Let Omega be the abstract root of an ARC Set.

An ARC is a loop-free ordered set of nodes and links whereby traffic may enter via any node in the ARC but may only leave the ARC through either one of the ARC edges.

A Comb is a generalization of an ARC. It is an n-edged loop-free set of nodes and links with \( n \geq 2 \). Traffic may enter via any node in the Comb but may only exit the Comb through one of its \( n \) edges.

A Cursor is a virtual point along an ARC. It can be located on a node or on a link between 2 nodes. In normal operations, the traffic along the ARC flows away from its Cursor towards one of its edge. The traffic can be distributed on both directions. Changing the Cursor location has a direct impact on the traffic load-balancing along an ARC. Upon a failure, data packets may bounce on the breakage points and flow the other way along the ARC to take the other exit. Moreover, if the Cursor is correctly placed at the location of a failure, then the traffic can be routed away from that point.

An Intermediate ARC Node is different from the extremity nodes of an ARC. It can receive traffic and forward traffic between its adjacent ARC nodes. By definition, an Intermediate Link between two Intermediate ARC Nodes is reversible.

A collapsed ARC is composed by a single Intermediate ARC Node, being at the same time the cursor and Edge nodes, and two exit nodes.

An ARC set is a Directed Acyclic Graph (DAG) with ARCs as vertices. In the DAG, an edge between the ARC A and the ARC B corresponds to a link from an edge ARC Node in ARC A and an arbitrary ARC Node in ARC B.

The ARC Height is an arbitrary distance from Omega on an ARC topology. The height of an ARC must be superior to the height of any of the ARCs where it terminates into. The order of the ARCs formation by a given algorithm can be used as the main parameter of each ARC Height. By definition, the ARC Height of adjacent ARCs must be different.
D3.2 Smart Routing Mechanisms Design

A Buttressing ARC is a split ARC that is merged into another ARC at one of its edge. An ARC and one or more Buttressing ARCs form a Comb construct that is resilient to additional breakages.

A node is Safe if there is no single point of failure, apart from the node itself, on its way to Omega. A node is Safe if it has at least two non-congruent paths to two different other Safe Nodes. As a consequence, a Safe node has at least two completely disjoint paths to Omega. When an ARC has been successfully constructed, its nodes become safe according to the ARC root Omega.

A node N is said dependent on the node S or S-dependent (noted ?-S) if S is the last single point of failure along the shortest path between S and Omega.

Figure 30 shows the Graph View representation that is similar to the classical routing graph. A simple arrow corresponds to an oriented link (e.g., the link from the node J towards the node K1). A dual arrow represents a reversible links where communications can be done in the two directions (e.g., the link between the nodes H and H1).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{routing_graph.png}
\caption{Routing graph view}
\end{figure}

In the graph in Figure 31 we can highlight 11 distinct ARCs, listed in Table 1.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{ARC} & \textbf{Information} \\
\hline
A & \{Omega\} \{A A1\} \{Omega\} \\
B & \{Omega\} \{B1 B2 B3 B\} \{A\} \\
C & \{A1\} \{C2 C C4\} \{Omega\} \\
D & \{B1\} \{D1 D D3\} \{B3\} \\
E & \{B\} \{E1 E\} \{A\} \\
F & \{A1\} \{F1 F F2\} \{C2\} \\
G & \{C\} \{G\} \{C4\} \\
H & \{D\} \{H2 H H1\} \{E1\} \\
I & \{H1\} \{I\} \{J1\} \\
J & \{F1\} \{J1 J\} \{K1\} \\
\hline
\end{tabular}
\caption{ARCs’ information}
\end{table}
By definition, each ARC is composed by three subsets of nodes \{Exit1\}{Intermediate Nodes}{Exit2}. The data traffic must leave the ARC from at least one of its node from \{Exit1\} and/or \{Exit2\}. Inside the intermediate subset, the traffic can be forwarded towards each direction. In fact, the corresponding links are reversible.

In Figure 31, we show the ARC View representation, where an ARC is represented as a sequence of reversible links, except the edge links which are directed.

In normal conditions, only the cursor may distribute its traffic between the Edges nodes. For instance, all the traffic of a cursor can be forwarded towards an Edge Node. If congestion appears, the cursor can forward new data traffic towards another edge node from the opposite ARC side. If congestion is still present, then the cursor can be moved towards the congested Node or Edge in order to redirect more traffic towards the other opposite direction.

3.6.2 ARC Algorithm

We aim to build ARCs on a physical topology connecting smart objects together. When a path is requested by a source node towards a destination node, conventional routing algorithms update the routing table stored in each network node with the best neighbour candidate. Thus data packets can be forwarded in order to reach the destination in an optimal way.

The ARC routing provides an innovative approach as two candidates are directly assigned to the path performance between two nodes. The main advantage relies on the fast-rerouting feature that enables to directly re-route data packets in case of a network topology breakage.

The open Lowest ARC First (oLAF) is a Shortest Path First (SPF) variation that creates ARCs by connecting SPF paths. The ARCs include the SPF tree. The algorithm identifies the mono-connected zones and provides redundancy inside such zones. oLAF is based on the Dijkstra’s algorithm for SPF. The reverse SPF tree towards a destination is conserved and preferred for forwarding along the resulting ARC set. Omega is the destination that we want to reach with an ARC set. By definition, both ends of an ARC cannot terminate in the same node. A dependent Set of nodes, each owned by a Safe Node S, DSet(S) is...
D3.2 Smart Routing Mechanisms Design

created. It contains the nodes that are not determined to be Safe at the current stage of the computation, and for which S, the owner Safe Node, is the last single point of failure on the shortest path tree to Omega. We create DSet(S) in which we place S. We select the node in the set of Nodes that is closest to Omega using the cost towards Omega like in traditional reverse SPF tree construction. We place the selected node in the same dependent set as its parent in the reverse SPF tree. Each sub-tree grows separately inside DSet(S) where S is the virtual Safe node (the root of the sub-tree). Once we have placed the selected node in DSet, we consider its neighbours one by one. If at least one of the neighbours is already in a different DSet than this node, we select the neighbour that provides the shortest alternate path to Omega for the selected node. As a consequence two paths are isolated. The two sets are different and have no intersection. Finally the forwarding path between a source and a destination follows a cascade of ARCs. Unicast packet may enter an ARC via any of its nodes, but it may only leave the ARC through one of its two edges.

ARCs are formed as follows:

- A height is attributed to each ARC that must be strictly superior to the one of the ARCs it terminates into, if any.
- The ARC terminates in the two Safe Nodes of two DSets. The normal behaviour is to make an Edge Link to the Safe Node.
- If the Safe Node at one end forms a collapsed ARC, it may be absorbed in the ARC in order to form a multi-edged ARC.
- If one of the two Safe Nodes depends on an ARC or a Comb construct that is higher than the other Safe Node, then this ARC may be merged at the Safe Node with its original ARC, in order to form a Comb construct.
- Each ARC is built by merging the two non-congruent paths, isolated for the selected node.
- The farthest node from Omega is selected as the cursor.
- The ARC can be used in both directions. Thus each node along the ARC has two non-congruent paths to the Safe Nodes where the ARC terminates into. So it is a Safe Node. We create individual DSets for each intermediate and we move it to its own DSet.
- Any Intermediate link is reversible. The normal direction, away from the cursor, preserves the shortest path.

The Forwarding process along an ARC Set is done as follows. Under normal conditions, the traffic flows away from the cursor of the current ARC and cascades into the next ARC. The Height of the current ARC is monotonically decreasing from ARC to ARC until Omega is reached. Upon a breakage inside an ARC, a corrective action takes place. It might be either operated at the control plane or the data plane, if immediate action and near-zero packet loss are required.

- Control plane: Upon a first breakage in an ARC, the Cursor is moved to the breakage point, so that the traffic flows away from the cursor again. Upon a second breakage within the same ARC, a segment of the ARC is isolated. All incoming links in the isolated segment are blocked. The traffic exits at the other end of the incoming ARCs. If a segment of an ARC is getting isolated by a dual failure but that ARC segment has incoming Edges then the ARC can be reversed. This reversal is done by reversing all the incoming edges, which become outgoing edges.
- Data plane: Upon a breakage inside an ARC, it is possible in the data plane to reverse
the direction of a given packet once along the ARC so the packets exit over the other Edge Link (turn). In order to avoid loops, it is undesirable to reverse the direction of a given packet a second time. The domain that is impacted by a turn is limited to the current ARC itself. A local strategy must be put in place inside an ARC to allow a given packet to bounce once upon a breakage, and get dropped upon a second breakage. In the case of IP packet forwarding, a packet can be tagged when it bounces inside an ARC, or when it passes the cursor, for instance by reserving a bit for that purpose. When a packet bounces, the bit is set and when the packet leaves the ARC, the bit is reset and may be used again in the next ARC.

3.6.3 Example: Building ARCs on a Simple Network Topology

In order to allow a clear understanding of the ARC algorithm, in this section we shows the different steps of the routing protocol, applied to the network topology represented in Figure 32 composed by 15 nodes and including 24 links.

The ARC algorithm is started by the node R that represents the root node, Omega. To distinguish the state of each link, we use the following convention. A simple segment corresponds to a non-SPF link to join an ARC. A black arrow corresponds to direct link between two nodes. Thus if there is an arrow from the node B to the node A, then A is the SPF successor of the node B. If the arrow is red, then the node A is the non-shortest path successor of the node B. If a red arrow between the nodes B and A is marked as “?-S”, then the link (BA) is unresolved for the Safe Node S. Instead, if the red arrow is marked as “Rev”, then the node B is standby alternate on the A isolation. The nodes A and B are Heir. In fact, at least they are connected to the root node R (Omega).
D3.2 Smart Routing Mechanisms Design

Let “?-A” be the set dependent on R(A) (step 1). In the following step (step 2), the neighbours C and D are also added into the set “?-A”. Then, also the set “?-B” is created, and the node K is added into it (step 3).

Thereafter, the nodes M and J are added into the dependent set “?-B” (step 4); the nodes L and E, and the node G are added into the dependent set “?-A” in two successive steps (steps 5-6).

Then, the node F is picked inside the dependent set “?-A” (step 7), but it also have a link towards the dependent set “?-B”. Therefore, it has two non-congruent paths to the two virtual Safe Nodes R(A) and R(B). After 8 steps, the first ARC infrastructure has been formed. The node F is a Safe Node. All nodes along the ARC are safe. They are placed alone in their own dependent set. All the other nodes are returned to the original set (step 9).
D3.2 Smart Routing Mechanisms Design

Among them, the node D is selected (step 10). It depends on the node A “?-A”. But there exists a direct link towards the Safe Node C. Thus, the node D forms a collapsed ARC (step11): in fact, its parent A and its preferred neighbour C are both Safe Nodes. Then, the node M is picked (step 12): it has also a direct link to the Safe Node J (step 12).

The node M forms also a collapsed ARC (step 13). Thereafter the node L is selected. It depends on the node D (step 14). Finally, the node E is picked (step 15).

The node E has links also to the nodes C and F that end deeper than the collapsed ARC formed by the node D (step 16). The node E adds a buttressing ARC (step 17). The link between the nodes E to D becomes reversible. The node L returns to the set. The node D becomes also the cursor for the Comb. The node L is picked (step 18) and it forms its own collapsed ARC.
D3.2 Smart Routing Mechanisms Design

Then, the node N, G and H are picked, one after the other (step 19, 20, 21).

The node H adds a buttressing ARC (step 22). Three nodes (N, G and I) are still not included inside the ARC set. First the nodes N and G are chosen (step 23). In detail, the node N is connected through the node L that is a safe node. In fact, the node N has a link towards the dependent set “?-A”. Instead, the node G has a link towards the dependent set “?-E”, and it has also a link to the safe node H. Thus the node G forms its own collapsed ARC (step 24).

Then the node I is selected. There are no more nodes in the set. Finally the node N is still dependent (step 25).
Finally, Figure 33 shows (a) the original graph (i.e., the network topology), and the forwarding topology built by (b) the classical rev-SPF, (c) the SPF-based DAG, and (d) ARC. As we can see in Figure 33(d), the ARC topology provides alternative paths in case of node or link breakages.

### 3.6.4 Simulation Environment

Scilab\(^\text{10}\) is an open source software for numerical computation providing a powerful computing environment for engineering and scientific applications. Scilab is released as open source under the CeCILL license (GPL compatible), and is available for download free of charge. Scilab is available under GNU/Linux, Mac OS X and Windows XP/Vista/7/8. Scilab includes hundreds of mathematical functions. It has a high level programming language allowing access to advanced data structures, 2-D and 3-D graphical functions. A large number of functionalities are included in Scilab:

- **Maths & Simulation:** For usual engineering and science applications including mathematical operations and data analysis.
- **2-D & 3-D Visualization:** Graphics functions to visualize, annotate and export data and many ways to create and customize various types of plots and charts.
- **Optimization:** Algorithms to solve constrained and unconstrained continuous and discrete optimization problems.
- **Statistics:** Tools to perform data analysis and modelling.

---

\(^{10}\) [www.scilab.org](http://www.scilab.org)
D3.2 Smart Routing Mechanisms Design

- Control System Design & Analysis: Standard algorithms and tools for control system study.
- Signal Processing: Visualize, analyse and filter signals in time and frequency domains.
- Application Development: Increase Scilab native functionalities and manage data exchanges with external tools.

Network Analysis and Routing eVALuation 2.0\(^\text{11}\), referenced as NARVAL, has been designed on top of the Scilab environment. It was created at the University of Luxembourg within the Interdisciplinary Centre for Security, Reliability and Trust (SnT). Figure 36 and 37 show, respectively, NARVAL logo, and a screenshot of some NARVAL functionalities, available under Windows.

![NARVAL Logo and Screenshot](http://atoms.scilab.org/toolboxes/NARVAL)

**Figure 34 NARVAL module**

NARVAL module is focused on the analysis of network protocols. The main goal of this toolbox is to provide a complete software environment enabling the understanding of available communication algorithms, but also the design of new schemes. NARVAL allows generating random topologies in order to study the impact of routing algorithms on the effectiveness of transmission protocols used by data communications. The target audience includes academics, students, engineers and scientists. The description of each function has been carefully done in order to facilitate the end users' comprehension. The module is self-sufficient as it does not depend on other internal/external Scilab toolboxes.

![NARVAL Screenshot](http://atoms.scilab.org/toolboxes/NARVAL)

**Figure 35 Screenshots of the NARVAL functionalities on Windows**

This toolbox has been developed on top of Scilab5.3.3. NARVAL has been updated into the Scilab ATOMS module manager. NARVAL has been tested and validated under the conventional programming environment

\(^{11}\) [http://atoms.scilab.org/toolboxes/NARVAL](http://atoms.scilab.org/toolboxes/NARVAL)
D3.2 Smart Routing Mechanisms Design

such as Linux, MAC and Windows.

For testing low-level smart routing based on ARC, we have implemented ARC in the NARVAL module. We provide the pseudo-code in the following paragraphs. We want to build the ARC topology for the selected root Omega inside a defined network topology. We build the first ARC A1, starting from Omega towards its direct neighbours heirs H1 and H2. This ARC consists of the merging of the two paths starting from H1 and H2, at their first intersection. The paths are propagating in a hop-by-hop way. Discovery packets are forwarded and propagated to the direct neighbourhood of the node where they are currently located. Thus the process is stopped when two discovery packets coming from H1 and H2 intersect in a common node or link. When the first ARC A1 is found, we update the ARC set with all nodes belonging to A1. When all nodes of the network topology are not included into the ARC set, we run the three following steps:

1. We select a candidate node in the direct 1-hop neighbourhood of the current ARC set. This node is named sub-heir.
2. We find the corresponding sub-root node already included inside the ARC set that is connected to the previous node named sub-heir.
3. We find the first intersection between the new ARC starting from the sub-root node towards the sub-heir node and the current ARC set. For that, the current node where the propagation started, forwards a discovery packet towards all of its 1-hop neighbours. The propagation process iterates as each discovery packet present in a node is forwarded towards each direct neighbours, and so on, until a discovery packet reaches a node belonging to the current ARC set. A TTL parameter can be used in order to limit the research radius of the propagation process. A loop control mechanism has been also implemented. It is not permitted for a discovery packet to cross again the path already travelled. Thus if there is no intersection after TTL hops, we can stop the propagation process. Several reasons can lead to this, mainly due to the network topology in consideration. If at least one intersection is found, we update the ARC set with the list on the new nodes extracted from the newly discovered ARC. We assume that the ARC construction is done in a central node, for instance the root where the ARC algorithm is started.

We provide a non-exhaustive list of functions that are used to build an ARC set on a network topology:

- **NARVAL_R_ARC** generates the complete ARC topology between a root node and its two heirs.
- **NARVAL_R_ARCFirst** performs the first ARC between the root node and its two heirs.
- **NARVAL_R_ARCSourceInit** initializes the propagation process from the source node.
- **NARVAL_R_ARCSourceIter** performs the propagation of the discovery packets used during the ARC construction.
- **NARVAL_R_ShowARC** highlights the \( i^{\text{th}} \) ARC of the ARC topology (chronological order).
- **NARVAL_R_ShowARCs** displays the complete ARC topology on a graph.
- **NARVAL_F_SubARC** enables to extract the two paths where a node can forward its traffic in order to exit its ARC.
- **NARVAL_R_ShowPathARC** highlights the \( j^{\text{th}} \) paths between a defined node of a network topology and the root node Omega where the ARC topology has been built.
- **NARVAL_R_RoutingARC** updates the routing table in each network node after
D3.2 Smart Routing Mechanisms Design

building the ARC topology.

- **NARVAL_R_SortARC** sorts the ARC matrix according to the weight and hop-length of the paths.

### 3.6.5 ARC Simulation on NARVAL

We present in this section a simulation generated by the NARVAL module.

![Figure 36 Network Topology](image)

We have created the same network topology described in Section 3.5.3, and shown in Figure 36. Omega is the node 1. The two heirs are the nodes 2 and 3. We perform the ARC algorithm, using the following code line:

\[ [M, SR] = \text{NARVAL\_R\_ARC}(g, s, h1, h2) \]

Where \( g \) is the graph representation of the network topology, \( s \) is the root node Omega; \( h1 \) and \( h2 \) are the root’s heirs. SR provides the order of the sub-root nodes’ selection. \( M \) is a matrix where we chronologically store the new discovered ARCs. In our case, we obtain 10 ARCs, listed in Table 2.

<table>
<thead>
<tr>
<th>Exit 1</th>
<th>Intermediate Nodes</th>
<th>Exit 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>{5 10 11 6 3}</td>
<td>{1}</td>
</tr>
<tr>
<td>{11}</td>
<td>{7}</td>
<td>{3}</td>
</tr>
<tr>
<td>{11}</td>
<td>{7}</td>
<td>{6}</td>
</tr>
<tr>
<td>{5}</td>
<td>{9}</td>
<td>{10}</td>
</tr>
<tr>
<td>{2}</td>
<td>{4}</td>
<td>{5}</td>
</tr>
<tr>
<td>{2}</td>
<td>{4}</td>
<td>{9}</td>
</tr>
<tr>
<td>{4}</td>
<td>{8}</td>
<td>{9}</td>
</tr>
<tr>
<td>{9}</td>
<td>{13 14}</td>
<td>{10}</td>
</tr>
</tbody>
</table>

*Table 2 Application of NARVAL\_R\_ARC*
D3.2 Smart Routing Mechanisms Design

Then, we obtain \( SR = \{1, 11, 5, 2, 4, 9, 14, 8\} \).

Thereafter we perform the cursor selection for each ARC:

\[
[MC] = NARVAL_R_CursorARC(M)
\]

The selected cursors are listed in Table 3.

**Table 3 Application of NARVAL_R_CursorARC**

<table>
<thead>
<tr>
<th>ARC</th>
<th>Cursor</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>8</td>
</tr>
</tbody>
</table>

Thereafter we calculate the ARC routing.

\[
[P, CP] = NARVAL_R_RoutingARC(MC, s, d, l1, l2, r1, r2, g)
\]

\[
d = 12
\]

\[
l1 = 11 ; \quad l2 = 12
\]

\[
r1 = 5 ; \quad r2 = 19
\]

P is a matrix. It provides all alternative paths between the root node s and the destination node d. As shown in Table 4, 8 different paths are identified. The parameters \((l1, l2, r1, r2)\) are used during the visualization, for discerning left and right. In fact, inside an ARC, an Intermediate Node can forward a data packet towards two opposite directions (Left and Right). We use 4 parameters in order to be able to distinguish the travel of a single packet along the succession of ARCs. In fact, if, for instance, a packet follows the right direction on an ARC, and then the right direction on the next ARC, we display each segment with different colours. Four parameters are adequate to face all possible situations.
**Table 4 Application of NARVAL_R_RoutingARC for the destination node 12**

<table>
<thead>
<tr>
<th>Paths</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>a 12</td>
<td>5</td>
</tr>
<tr>
<td>b 12</td>
<td>6</td>
</tr>
<tr>
<td>c 12</td>
<td>6</td>
</tr>
<tr>
<td>d 12</td>
<td>7</td>
</tr>
<tr>
<td>g 12</td>
<td>10</td>
</tr>
<tr>
<td>f 12</td>
<td>9</td>
</tr>
<tr>
<td>g 12</td>
<td>9</td>
</tr>
<tr>
<td>e 12</td>
<td>8</td>
</tr>
</tbody>
</table>

We can also sort the set of available paths. The result on P (respectively CP) is SP (respectively SCP).

\[
[SP,SCP] = \text{NARVAL\_R\_SortARC}(P,CP,g)
\]

We can visualize each alternative path i on the graph go.

\[
[\text{go}] = \text{NARVAL\_R\_ShowPathARC}(SCP,SP,i,g)
\]

We present in Figure 37 different paths generated for the connection between the node 12 and the root node 1 (Omega). They are sorted according to the total path weight.
Let us consider the shortest path provided by the ARC algorithm, shown in Figure 37 a). Data traffic should be forwarded along the path \{12 \; 8 \; 4 \; 2 \; 1\}. If a breakage occurs in the node 2, we assume that data packets travelled along the path \{12 \; 8 \; 4\}. As the breakage can be monitored by the local neighbours of a node, decision can be taken in order to re-route the traffic towards the other direction of the current ARC towards the second exit. For instance, traffic can be routed towards the node 5. Therefore, since the node 2 is not available, the cursor of the current ARC will send the traffic towards the path \{10 \; 11 \; 6 \; 3 \; 1\}. Finally, the alternative path is \{12 \; 8 \; 4 \; 5 \; 10 \; 11 \; 6 \; 3 \; 1\}.

We can do the same path calculation for the destination node 14. The obtained paths are provided in Table 5.

**Table 5 Application of NARVAL_R_RoutingARC for the destination node 14**

<table>
<thead>
<tr>
<th>Paths</th>
<th>14</th>
<th>13</th>
<th>9</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
We present also in Figure 38 the different paths generated for the connection between the node 14 and the root node 1 (Omega). They are sorted according to the total path weight.

![Figure 38](image)

*Figure 38 Multiple paths generated by the ARC algorithm for a communication between the node 14 and the root node 1*

### 3.6.6 Comparison between ARC and RPL

In order to understand the benefit of ARC, in this section we compare its performance with the ones achievable with RPL. We consider the same network topology, shown in Figure 39, and we assume node 1 is the DODAG root of the RPL DODAG, illustrated in Figure 40.

We provide a non-exhaustive list of functions that are used to build a RPL DODAG set on a network topology:

- **NARVAL_R_ARC** builds a DODAG RPL tree from a root node.
- **NARVAL_R_RPLAncestor** finds the first ancestor between two network nodes according to
D3.2 Smart Routing Mechanisms Design

the RPL algorithm.

- **NARVAL_R_RPLInit** initializes the RPL algorithm from a root node.
- **NARVAL_R_RPLPath** performs the path between a node and the DODAG root according to the RPL algorithm.
- **NARVAL_R_RPLPaths** performs a set of paths between a node and the DODAG root according to the RPL algorithm.
- **NARVAL_R_RPLViz** highlights a DODAG tree generated by the RPL algorithm.
- **NARVAL_R_RPLDIO** propagates DIO messages from a node.

The sink (root node) first starts to multicast DODAG Information Object (DIO) messages on the network topology. DIO messages carry information that permits to any node receiving it to discover a RPL instance, learn its configuration parameters, select a DODAG parent set and maintain the DODAG. When a node receives a new DIO version, it performs its rank and propagates its own DIO messages. In fact the Rank of a node is a scalar representation of the location of that node within a DODAG Version. It is used to avoid and detect loops. The rank increases with the hop distance towards the sink. Thus a node presenting a smaller rank can be seen as a potential parent inside the DODAG. The optimal routes are calculated according to defined constraints and metrics. In order to refresh and update the DODAG, the sink periodically sends new DIO messages. As a consequence, if a new node joins the network, or gets disconnected from its parent (predecessor), it can wait for the next DIO message. It can also request the sending of DIO message according to a DODAG Information Solicitation (DIS) message.

The predecessor vector provides for each node its parents where packets will be forwarded in order to reach the root node (see Table 6). The objective function optimizes the hop distance between each node and the root.

**Table 6 Predecessor vector (RPL)**

<table>
<thead>
<tr>
<th>Node</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predecessor</td>
<td>ø</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Node</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Predecessor</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
D3.2 Smart Routing Mechanisms Design

Let us consider the path between the node 14 and the root node 1, in RPL topology. As we can see in Figure 40, it is given by \{14 9 4 2 1\}.

In a network running RPL, all data packets generated by a given node, are forwarded along the same path, built according to the routing table updates. This is clearly a drawback of the RPL protocol. In fact, if a breakage occurs, some actions need to be taken in order to repair the path (local or global repair). A complete rebuild of the DODAG tree can also be seen as the last solution if the failure has a large scale impact on the topology. A local repair can consist of rerouting packets from the breakage location. For instance, if the node 9 does not respond anymore after a breakage, there is a possibility for the latest node alive on the path (in this case node 14) to check if its direct neighbours has an entry in their routing table related to the source node 1. For instance, the traffic can be forwarded to the node 10 that has an operational entry to the root node (node 5). This action is useless if the new path is also affected by the first breakage (for instance the node 13 that forwards the traffic towards the node 9). Unfortunately the local and global repairs add new computations and delays in order to solve the breakage. Unlike RPL, ARC provides immediately alternative paths, thus saving energy, and avoiding delays.
3.6.7 Conclusion

As proposed, ARC was selected as one of the enabling technology for low-level smart routing. The goal was to research on a recent technology that can compete with standard best effort routing and QoS routing algorithms. In particular, it allows the WSN to recover rapidly when a link or node fails. In particular, due to the construction of arcs two edged nodes, this allows rerouting traffic without any additional route calculation.

This is the main benefit of ARC which can be evaluated with the implemented simulation library. Thanks to this contribution and already available routing and QoS algorithms in NARVAL, it is possible to compare the different approaches. As the first open source ARC simulator, this contribution is highly important for IoT6 and for the academic and industrial community. However, the lack of real implementation for this technology made us choose a better supported solution for integration within the IoT6 architecture.
4 Illustrative use cases

4.1 Building Maintenance Process

This scenario presents a situation within a building automation system, where spurious behavior of an automation component is observed. The IoT functionality is then used to investigate the malfunction, order a replacement part, guide the replacement process, and finally confirm the replacement and resume normal operation. During this process, a mobile phone, a maintenance web service (SaaS or UDG with web interface), a Discovery server as a resource discovery tool, a STIS as an information lookup service, an IPv6 – legacy protocols proxy, and a KNX temperature sensor as a legacy non-IPv6-enabled component are utilized.

The following actors/components are part of this scenario:

1) A **Worker** (local personnel) has an individual ID associated with certain access privileges, for instance the ability to initiate a maintenance process within the building automation.

2) A **Mobile Phone** is used as an HMI to interface with the building automation system. In an actual system, a nearby local terminal could also be used.

3) A **Maintenance App** is running on the mobile phone. It handles all communications with the maintenance environment. It is expected to be configured with the ID of the worker, in order to allow proper authorization towards the maintenance environment.

4) A **Maintenance Tool** (a web service, SaaS, or probably the Universal Device Gateway (UDG) with a web interface) accepts the maintenance request and utilizes the other components to carry out the maintenance process.

5) A **Digrectory** (resource directory) is used to locate services accessible within the local IoT. It supports a service description together with a set of parameters, and it responds with the IPv6-address of a node that has registered itself to provide this service.

6) A **UDG** is responsible for controlling and monitoring the automation system. It contains a usually large set of rules describing each control loop in the system. The maintenance is carried out by the maintenance tool, which subsequently notifies the UDG of the various state changes for the component in question.

7) An **Inventory Management** (web service or SaaS) provides an interface to the infrastructure management; in particular, it handles the order process for the replacement part.

8) A **CoAP/oBIX Gateway** is required to interface the IoT6-enabled part of the automation system with non-IPv6-enabled components. It provides access to the BACnet and KNX subsystems.

9) A **Temperature Sensor** is used. As an example of a non-IPv6-enabled automation device, a KNX-based Temperature Sensor can be used.

In this scenario, different servers can be used. For instance, during the Data gathering for the maintenance process, the user needs to log into the maintenance tool. The Discovery service can be contacted in order to get the IPv6 address handling maintenance cases. Maintenance operations can be split among different servers.

There is a possibility that within one building automation system, different parts of the automation are assigned to different maintenance authorities. Providing the RFID tag of a component would allow the Digrectory to direct the request to the appropriate server. The user ID would allow the Digrectory to be set up so that users with different access rights could be directed to different servers. It could be necessary also to provide the RFID tag so
D3.2 Smart Routing Mechanisms Design

that the Discovery service can find the appropriate STIS server. Moreover different parts of the automation can be assigned to different maintenance authorities. Thus requests can be forwarded to the appropriate server. This could possibly serve as a showcase of how the IoT functionality is inherently aware of the properties of the ‘thing’ involved, as opposed to merely routing an opaque data unit to a pre-determined application. The STIS service might also be distributed over several servers with different device subsets assigned to them, meaning that it could be necessary to provide the RFID tag so the Discovery service can find the appropriate STIS server. Smart routing will contribute to the implementation of this use case thanks to the implementation of Content Packet filtering techniques in the smart gateway and of the support of flow label tagging by sensors.

4.2 Safety Alert with QoS

This scenario demonstrates how an alarm handling is managed in the IoT6 architecture. The IoT6 features such as “smart routing”, priority routing and QoS are demonstrated in order to ensure fast and reliable data exchanges between different components and applications. The scope of this use case is the initiation of an alarm, its propagation to different alarm systems.

The following actors/components are part of this scenario:

1) A CoAP/oBIX Gateway is used interface the IoT6 architecture with non-IPv6-enabled components. It provides access to BACnet and KNX subsystems.

2) A Discovery (resource directory) is used to locate services accessible within the local IoT. It is also used to geo-localize mobile devices.

3) A UDG is responsible for controlling and monitoring the automation system. It contains a usually large set of rules describing each control loop in the system. The UDG is responsible for the detection of abnormal conditions through implemented rules (smart routing).

4) A Safety Server 1/Safety Server 2 component is used to relay messages to a number of components and devices in the IoT6 architecture (IoT proxy). It distributes messages according to pre-defined rules implemented by the UDG.

5) A Temperature Sensor is used. As an example of a non-IPv6-enabled automation device, a KNX-based Temperature Sensor is used. During the demonstration, the sensor will be cooled/warmed to a certain temperature to trigger the alarm.

6) Alarm1/Alarm2 components describe IPv6-enabled alarm devices that maybe connected via 6LoWPAN.

7) An Alarm Light is used. As an example of a non-IPv6-enabled alarm device, a BACnet-based Alarm Light is used.

8) A Safety Management Tool (SaaS) accepts incoming alarm messages and relays them to mobile devices according to their vicinity to the alarm location. It keeps track of the generated alarm via a ticketing mechanism and also informs the mobile clients as soon as the alarm has ceased.

9) Safety Management App1/Safety Management App2 components are mobile phone applications running on a mobile client. They are used as a channel to physical persons in order to transfer alarm messages. They handle the communication with the safety management tool.

10) A System Engineer is responsible for handling a specific alarm. He is registered via an IPv6-enabled mobile phone and a Safety Management Application at the Safety Management Tool component.

The Safety Server 1/Safety Server 2 components are configured by a system engineer. The registration is done to the Discovery server. It acknowledges the registration of
D3.2 Smart Routing Mechanisms Design

Alarm 1/Alarm 2. During the device discovery, the Discovery server retrieves the IPv6 addresses of needed services. A rule for handling an abnormal temperature is created (smart routing). The system engineer should define how alarms need to be tagged, and also a rule for processing energy-critical information. He should also install relaying and distribution rules on the safety servers. The UDG is the component that knows which alarm devices should be triggered for a certain alarm event. The distribution route is stored at the safety servers such that these components have a fixed route for each alarm event. The UDG recognizes the abnormal temperature in the received packet and tags the data as alert data. Then the alarm is sent to a group of safety servers (anycast). The IPv6 alarm devices can be any device connected to the IoT6 architecture with IPv6-enabled protocols such as 6LoWPAN. The installation of forwarding rules on the Safety Servers has to be facilitated by the UDG. The safety server component is a simple distribution hub for alarm messages. In practice this server could be integrated into the UDG component. In this scenario, smart routing consists of a specific routing based on content. For example, sensors can mark packets containing safety related information by using either the flow label field or the traffic class field of 6LoWPAN/IPv6 packets. According to that field, they will be processed differently by the smart gateway which will use packet filtering techniques. Finally, packets are sent to the most appropriate destination according to the transported data and in the most efficient way.

4.3 The Smart Office

This scenario is directly inspired by the IIAB proposals. It intends to demonstrate the ability of the IoT6 architecture to interact with heterogeneous devices, including non-IP based protocols, with a focus on energy efficiency and user comfort.

The following actors/components are part of this scenario:

1) A **System Engineer** sets up the automation technology. This person is an expert in the required system technologies and also an expert for configuring the involved IoT6 components (the CoAP/oBIX gateway and the UDG control & monitoring system).
2) An **Employee** is equipped with an RFID/NFC enabled identity card or smartphone and personal work space preferences.
3) A **Visitor** has no permanent relationship to the office and no identification mechanism.
4) A **Smartphone** is used by the employee to adjust his personal workspace settings (light, temperature).
5) A **Smart Office Application** is running on the mobile phone of the employee.
6) A **Directory (local resource directory)** is used as a local DNS based resource directory providing an IPv6 address and servicing description for devices. These records are uploaded to a global Discovery system.
7) A **Discovery** is a Global IoT6 discovery system that can be used to search and discover devices based on certain criteria on resource type and interface.
8) A **UDG** is responsible for controlling and monitoring the automation system. It contains a usually large set of rules describing each control loop in the system. The maintenance is carried out by the Maintenance Tool, which subsequently notifies the UDG of the various state changes for the component in question.
9) A **CoAP/oBIX Gateway** is used to interface the IoT6-enabled part of the automation system. A proxy is required with non-IPv6-enabled components. The CoAP/oBIX Gateway provides access to BACnet and KNX subsystems.
10) **KNX devices** are used (light switch actuator, temperature sensor, presence detector).
D3.2 Smart Routing Mechanisms Design

*BACnet* devices: are used as controller for the interaction with the HVAC system, lighting and temperature sensors.

Based on the configured devices in the control and monitoring system, the system engineer can specify rules and logic for the smart office scenario (smart routing).

5 Conclusion

This deliverable precisely defines what smart routing is. In particular, two levels of smart routing have been indentified. The low level smart routing is more related to the standard QoS. Unlike the low level smart routing that does not affect the final destination of a message, the high level smart routing may alter the destination of a message to forward it to the most suitable recipient. Obviously, it is recommended to use both of them in conjunction to achieve the best performances. Four distinct technologies have been selected. Iptables has been considered since, as a widely supported open-source solution, it allows an easy integration of smart routing in WP5, in particular on the Smart Board with an optimized Linux system. Therefore, precise guidelines, on how to install and configure it properly according to the smart routing definition we propose, are given. Some test scripts have been also implemented to verify the compatibility with the smart board. Moreover, we have identified some limitations (*e.g.* usability of the flow label in Internet) and propose solutions to circumvent them.

While this solution clearly fits the primary objective of T3.2 as outlined in the DoW, as well as to the IoT6 use cases, we have also decided to investigate and propose new innovative propositions by exploring recent paradigms and approaches. This will contribute to new advances in the scientific community and give a higher visibility to the European research and especially IoT6. More precisely, OpenFlow gains a great deal of importance both in the academic and industrial community. Thanks to this deliverable, we highlighted how such a technology can be used for performing smart routing. ARC is also a promising approach which is promoted by Cisco for QoS purposes and in the context of IoT6, the first open source simulator was built to assess its performances for different WSN configurations. Finally, CCN promoted by PARC has been also considered. To the best of our knowledge, we are the first to propose and evaluate potential optimisations when using CCN to carry IoT traffic.
D3.2 Smart Routing Mechanisms Design

References

D3.2 Smart Routing Mechanisms Design


D3.2 Smart Routing Mechanisms Design

specifications/openflow/openflow-spec-v1.3.1.pdf, 2012


D3.2 Smart Routing Mechanisms Design
