Towards the future Internet: a step forward the successfully evolution of WSNs into the IoT

Stefano Bocchino
ReTiS Lab.
Scuola Superiore Sant’Anna

A thesis submitted for the degree of
Doctor of Philosophy

Tutor: Prof. Giuseppe Lipari
Supervisor: Dr. Paolo Pagano

October the 16th, 2014
Abstract

Wireless Sensor Networks (WSNs) have experienced a rapid progress in the last decade, and traditional WSNs have gone from being simple isolated monitoring systems to powerful and inter-operable systems connected to the Internet world. However, to reach a wide adoption of the WSNs in our daily life still several limitations of current systems persist. To this end, this thesis discusses the main challenges and the advances of WSNs at the beginning of the IoT era in order to exploit WSN technology in our daily life.

One of the challenges treated in this thesis is the integration of the WSN technology in the Internet world by following the IoT vision. In order to reach a seamless integration of tiny mote devices in Internet, the protocols developed for traditional WSNs must be modified or adapted by taking into account the constraints and benefits of the new IoT protocols. In this regard, two original contributions are presented: SPEED-6LoWPAN and SPEED-3D. SPEED adaptations support soft real-time, load balancing and flow shaping mechanisms making themselves effective solutions in supporting packet routing in 6LoWPAN networks.

Recent progresses in hardware solutions have permitted to integrate on tiny motes powerful sensors, e.g., camera, Radio-Frequency IDentification (RFID), thus leading the research community to develop advanced WSN-based applications in new scenarios. In this regard, the thesis describes two application scenarios where this new sensors can be used: (i) video-streaming, by proposing a low-complexity algorithm based on background subtraction and error resilience techniques, and (ii) limited access zones, by proposing an integration of WSN and RFID technologies in the IoT scenario.

Although WSN is a key technology for IoT, their pace of prevalence is slower than anticipated by the research community, and one of the reasons is the required embedded systems expertise for developing and deploying WSNs. To this end, this thesis also discusses the use of abstraction for simplifying the development of WSN applications by presenting PyFUNS, a Python-based Framework for Ubiquitous Networked Sensors. PyFUNS handles low level and networking functionality and leaves to the user only the task of application development in the form of Python scripts. This approach reduces required expertise in embedded systems to develop WSN based applications. PyFUNS also uses 6LoWPAN and CoAP standard protocols to enable interoperability and ease of integration with other systems, pursuing the Internet of Things vision.

In order to complete this work of thesis, the thesis presents the advances discussed applied to one of the most promising use cases of the smart cities scenario: Intelligent Transport System (ITS).
To my family.

To my wife, Loredana.
## Contents

Abstract iii  
List of Figures xi  
List of Tables xv  
Acronyms xvii  

Chapter 1. Introduction 1  
1.1. Wireless Sensor Networks at the beginning of the IoT era 2  
1.2. Research challenges in next generation WSN as motivation of this thesis 3  
1.3. Main contributions 5  
1.4. Outline of this thesis 6  

Chapter 2. Internet of Things Protocols Adaptation 7  
2.1. Background 8  
2.2. Related works 8  
2.3. SPEED protocol overview 9  
2.4. SPEED-6LoWPAN 11  
2.4.1. Routing protocol adaptation 11  
2.4.2. Implementation in ERIKA OS 15  
2.4.3. Performance evaluation in a real scenario 17  
2.5. SPEED-3D 23  
2.5.1. Routing protocol adaptation 23  
2.5.2. Implementation in ERIKA OS 24  
2.5.3. Performance evaluation in a real scenario 25  
2.6. Final remarks 32  
2.7. Conclusion 33  

Chapter 3. Combine IoT protocols with new technologies 35  
3.1. Internet of Things architecture in Wireless Sensor Networks 36  
3.2. Video streaming applications in IoT scenario 37  
3.2.1. Background 39  
3.2.2. Related works 40  
3.2.3. Wireless Camera Networks video compression and transmission 41  
3.2.4. Performance evaluation 51  

vii
3.3. WSN and RFID integration in the IoT scenario
   3.3.1. Background
   3.3.2. Related works
   3.3.3. WSN and RFID integration through the REST paradigm
   3.3.4. System design
   3.3.5. Performance evaluation
3.4. Conclusion

Chapter 4. Abstraction in 6LoWPAN
   4.1. Motivations
   4.2. Related work
   4.2.1. Techniques for remote reprogramming of WSNs
   4.2.2. Frameworks enabling easier programming of WSN
   4.3. PyFUNS design
   4.3.1. Dynamic services over WSN
   4.3.2. Application components
   4.3.3. Application life-cycle
   4.3.4. Application flow
   4.3.5. Application RESTful interface
   4.3.6. PyFUNS implementation
   4.3.7. Example of usage
4.4. Performance evaluation
   4.4.1. Flash and RAM requirements
   4.4.2. Native code versus Python script
   4.4.3. Real case evaluation
4.5. PyFUNS extension
4.6. Conclusions

Chapter 5. Application scenario
   5.1. Intelligent Transport System and Internet of Things
   5.1.1. Wireless Camera Network Applications in the ITS scenario
   5.1.2. REST paradigm in IoT-based WSNs
   5.2. Proposed Middleware Architecture for IoT-based WCNs
   5.2.1. RESTful Web Service
   5.2.2. Configuration Manager
   5.2.3. Resource Processing Engine
   5.3. Middleware Instantiation for the Parking Lot Monitoring Use Case
   5.3.1. Use case scenario, exposed resources and their interaction
   5.3.2. Middleware implementation
5.4. Conclusions

Chapter 6. Conclusions
List of Figures

2.1 The speed-based forwarding mechanism in SPEED. 10
2.2 IPv6 vs. 6LoWPAN encapsulation in IEEE 802.15.4 data frame. 12
2.3 SPEED-6LoWPAN messages encapsulation. 14
2.4 SPEED-6LoWPAN modules. 16
2.5 Network stack architecture in ERIKA. 17
2.6 Network configuration. 18
2.7 The FLEX development board. 18
2.8 Ad-Hoc on Demand Distance Vector (AODV) transmission path. 19
2.9 AODV route formation time. 20
2.10 SPEED-6LoWPAN transmission path. 21
2.11 AODV vs. SPEED-6LoWPAN route formation time. 21
2.12 Transmission paths for flow shaping property evaluation. 22
2.13 SPEED-6LoWPAN round trip time for high and low priorities. 22
2.14 SPEED-3D messages encapsulation. 24
2.15 6LoWPAN task activation. 25
2.16 Network configuration. 26
2.17 Round-Trip Time (RTT) Distribution for six, eight and ten hops. 28
2.18 RTT and Packet Lost Rate (PLR) versus number of hops. 28
2.19 RTT and PLR as a function of AST. 29
2.20 RTT and PLR varying the traffic load. 31
3.1 6LoWPAN architecture. 37
3.2 Wireless camera network architecture based on 6LoWPAN communications. 38
3.3 From top left to bottom right: WiCa, Mesheye, CITRIC, Vision Mesh, Seed-Eye and FPGA-based CN. 40
3.4 Video-surveillance functional block diagram. 41
3.5 The video-encoder. 43
3.6 Encoder. 43
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.7</td>
<td>The video-decoder.</td>
<td>44</td>
</tr>
<tr>
<td>3.8</td>
<td>Decoder.</td>
<td>44</td>
</tr>
<tr>
<td>3.9</td>
<td>Proposed codec: $\rho$ and PSNR as a function of the BRP.</td>
<td>46</td>
</tr>
<tr>
<td>3.10</td>
<td>The ghost phenomenon.</td>
<td>47</td>
</tr>
<tr>
<td>3.11</td>
<td>JPEG: $\rho$ and PSNR as a function of the Quality.</td>
<td>48</td>
</tr>
<tr>
<td>3.12</td>
<td>Self-reconstructing packet header fields description.</td>
<td>49</td>
</tr>
<tr>
<td>3.13</td>
<td>Standard and proposed MAC data messages.</td>
<td>50</td>
</tr>
<tr>
<td>3.14</td>
<td>Concealment techniques.</td>
<td>51</td>
</tr>
<tr>
<td>3.15</td>
<td>Performance comparison for the different improvements of the proposed codec.</td>
<td>53</td>
</tr>
<tr>
<td>3.16</td>
<td>Comparison of Peak Signal-to-Noise Ratio (PSNR), data rate and PLR between</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>the proposed codec and JPEG.</td>
<td></td>
</tr>
<tr>
<td>3.17</td>
<td>IoT based WSN and RFID integrated network.</td>
<td>59</td>
</tr>
<tr>
<td>3.18</td>
<td>System components and exposed resources.</td>
<td>60</td>
</tr>
<tr>
<td>3.19</td>
<td>Messages exchanged among network devices.</td>
<td>61</td>
</tr>
<tr>
<td>3.20</td>
<td>Seed-Eye board.</td>
<td>62</td>
</tr>
<tr>
<td>3.21</td>
<td>H node with actuation resource.</td>
<td>62</td>
</tr>
<tr>
<td>3.22</td>
<td>HT node embedding a semi-passive tag.</td>
<td>63</td>
</tr>
<tr>
<td>3.23</td>
<td>HR node embedding an Radio-Frequency IDentification Reader (RFID-Reader).</td>
<td>63</td>
</tr>
<tr>
<td>3.24</td>
<td>Response time pdfs for the cases of 2 and 5 hops.</td>
<td>66</td>
</tr>
<tr>
<td>3.25</td>
<td>Response time pdf for logging location data.</td>
<td>68</td>
</tr>
<tr>
<td>4.1</td>
<td>Scripts flow chart.</td>
<td>76</td>
</tr>
<tr>
<td>4.2</td>
<td>Application finite state machine.</td>
<td>77</td>
</tr>
<tr>
<td>4.3</td>
<td>Script flow chart.</td>
<td>78</td>
</tr>
<tr>
<td>4.4</td>
<td>The structure of an application resource.</td>
<td>79</td>
</tr>
<tr>
<td>4.5</td>
<td>The PreScript of Security application.</td>
<td>82</td>
</tr>
<tr>
<td>4.6</td>
<td>The MainScript of Security application.</td>
<td>82</td>
</tr>
<tr>
<td>4.7</td>
<td>The PostScript of Security application.</td>
<td>82</td>
</tr>
<tr>
<td>4.8</td>
<td>The PreScript of Motion Detection application.</td>
<td>83</td>
</tr>
<tr>
<td>4.9</td>
<td>The MainScript of Motion Detection application.</td>
<td>83</td>
</tr>
<tr>
<td>4.10</td>
<td>The PostScript of Motion Detection application.</td>
<td>83</td>
</tr>
<tr>
<td>4.11</td>
<td>The PreScript of Lamp Control application.</td>
<td>84</td>
</tr>
<tr>
<td>4.12</td>
<td>The MainScript of Lamp Control application.</td>
<td>84</td>
</tr>
<tr>
<td>4.13</td>
<td>The CC2538dk board integrated with PIR sensors and alarm.</td>
<td>85</td>
</tr>
<tr>
<td>4.14</td>
<td>Network topologies: star (a-b-c), mesh (d) and tree (e-f-g-h-i).</td>
<td>88</td>
</tr>
<tr>
<td>Figure</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.15</td>
<td>Security - Energy consumption in star (a-b-c) and mesh (d) topologies.</td>
<td>90</td>
</tr>
<tr>
<td>4.16</td>
<td>Security - Energy consumption in tree (e-f-g-h-i) topologies.</td>
<td>90</td>
</tr>
<tr>
<td>4.17</td>
<td>Security - Network traffic in star (a-b-c) and mesh (d) topologies.</td>
<td>90</td>
</tr>
<tr>
<td>4.18</td>
<td>Security - Network traffic in tree (e-f-g-h-i) topologies.</td>
<td>91</td>
</tr>
<tr>
<td>4.19</td>
<td>Security - Delay time in star (a-b-c) and mesh (d) topologies.</td>
<td>91</td>
</tr>
<tr>
<td>4.20</td>
<td>Security - Delay time in tree (e-f-g-h-i) topologies.</td>
<td>92</td>
</tr>
<tr>
<td>4.21</td>
<td>Light Control - Energy consumption in star (a-b-c) topologies.</td>
<td>93</td>
</tr>
<tr>
<td>4.22</td>
<td>Light Control - Energy consumption in tree (e-f-g) topologies.</td>
<td>94</td>
</tr>
<tr>
<td>4.23</td>
<td>Light Control - Energy consumption in tree (h) topology.</td>
<td>94</td>
</tr>
<tr>
<td>4.24</td>
<td>Light Control - Energy consumption in mesh (d) and tree (i) topologies.</td>
<td>94</td>
</tr>
<tr>
<td>5.1</td>
<td>Wireless Camera Network exposing parking space occupancy levels as resources.</td>
<td>98</td>
</tr>
<tr>
<td>5.2</td>
<td>IoT protocol stack versus Internet protocol suite.</td>
<td>101</td>
</tr>
<tr>
<td>5.3</td>
<td>Example of an IoT-based WSN supporting ITS services.</td>
<td>101</td>
</tr>
<tr>
<td>5.4</td>
<td>Middleware components.</td>
<td>103</td>
</tr>
<tr>
<td>5.5</td>
<td>Exposed resources and interaction model.</td>
<td>106</td>
</tr>
<tr>
<td>5.6</td>
<td>Parking slot application: PreScript and MainScript.</td>
<td>107</td>
</tr>
</tbody>
</table>
**List of Tables**

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Speed-6LoWPAN.</td>
<td>27</td>
</tr>
<tr>
<td>2.2</td>
<td>RTT and PLR performance: numerical results.</td>
<td>30</td>
</tr>
<tr>
<td>2.3</td>
<td>RTT and PLR performance: numerical results.</td>
<td>30</td>
</tr>
<tr>
<td>2.4</td>
<td>Analytic model validation.</td>
<td>32</td>
</tr>
<tr>
<td>3.1</td>
<td>Selected metrics.</td>
<td>45</td>
</tr>
<tr>
<td>3.2</td>
<td>Performance results of the proposed codec.</td>
<td>46</td>
</tr>
<tr>
<td>3.3</td>
<td>Performance results of the proposed codec versus JPEG.</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>Network metrics.</td>
<td>52</td>
</tr>
<tr>
<td>3.5</td>
<td>Performance results comparison between the different composition of the error resilience improvements.</td>
<td>52</td>
</tr>
<tr>
<td>3.6</td>
<td>Performance results comparison between the proposed codec and JPEG.</td>
<td>54</td>
</tr>
<tr>
<td>3.7</td>
<td>Response time as a function of the number of hops.</td>
<td>65</td>
</tr>
<tr>
<td>3.8</td>
<td>False rejection rate vs. number of reading attempts.</td>
<td>67</td>
</tr>
<tr>
<td>4.1</td>
<td>Application state transition.</td>
<td>78</td>
</tr>
<tr>
<td>4.2</td>
<td>PYFUNS APIs.</td>
<td>81</td>
</tr>
<tr>
<td>4.3</td>
<td>Code size and RAM requirements for a WiSMote and CC2538dk devices.</td>
<td>85</td>
</tr>
<tr>
<td>4.4</td>
<td>Performance benchmarks in Cooja.</td>
<td>86</td>
</tr>
<tr>
<td>4.5</td>
<td>Performance benchmarks on WiSMote.</td>
<td>87</td>
</tr>
<tr>
<td>4.6</td>
<td>Security Control service deployment configurations.</td>
<td>89</td>
</tr>
<tr>
<td>4.7</td>
<td>Light Control service deployment configurations.</td>
<td>92</td>
</tr>
<tr>
<td>5.1</td>
<td>Code size and RAM requirement.</td>
<td>108</td>
</tr>
</tbody>
</table>
Acronyms

**6LBR:** 6LoWPAN Border Router

**6LoWPAN:** IPv6 protocol over Low power Wireless Area Network

**6LR:** 6LoWPAN Router

**AODV:** Ad-Hoc on Demand Distance Vector

**API:** Application Programming Interface

**AUC:** Area Under Control

**BRP:** Background Refresh Period

**CN:** Camera Node

**CoAP:** Constrained Application Protocol

**CSMA-CA:** Carrier Sense Multiple Access with Collision Avoidance

**FCS:** Frame Check Sequence

**GTS:** Guaranteed Time Slot

**H:** Host

**HR:** Host Reader

**HT:** Host Tag

**ICT:** Information and Communication Technology

**IETF:** Internet Engineering Task Force

**IoT:** Internet of Things

**IPS:** Internet Protocol Suite

**ITS:** Intelligent Transport System

**MAC:** Medium Access Control

**MPH:** Multimedia Processing Hub

**NA:** Neighbor Advertisement

**ND:** Neighbor Discovery

**NS:** Neighbor Solicitation

**PAN:** Personal Area Network
**Acronyms**

**PHY**: Physical layer  
**PLR**: Packet Lost Rate  
**PSNR**: Peak Signal-to-Noise Ratio

**QoS**: Quality of Service  
**RA**: Router Advertisement  
**RFID**: Radio-Frequency IDentification  
**RFID-Reader**: Radio-Frequency IDentification Reader  
**RFID-TAG**: Radio-Frequency IDentification TAG  
**RFT**: Route Formation Time  
**ROI**: Regions Of Interest  
**RPL**: IPv6 Routing Protocol for Low power and Lossy Networks  
**RS**: Router Solicitation  
**RTT**: Round-Trip Time

**WCN**: Wireless Camera Network  
**WMSN**: Wireless Multimedia Sensor Network  
**WPAN**: Wireless Personal Area Network  
**WSN**: Wireless Sensor Network
CHAPTER 1

Introduction

SINCE its creation, the Internet has evolved over time by integrating new technologies developed to meet the changing needs of industry and society. This flexibility must be considered a main factor of its growth, and nowadays the Internet embraces the world bringing data and information to billions of people. The convergence of fixed and wireless technologies have helped to make the Internet an unique infrastructure, always available and accessible, to support a wide range of applications. The ubiquitous computing [Wei99] is the next step for the Internet growth and will be obtained by interconnecting the information network and the real world of concrete objects and places, the so called Internet of Things (IoT). The IoT concept is attributed to Auto-ID Center[1] founded in 1999.

The phrase “Internet of Things” covers the entire structure (hardware, software and services) that extends the Internet to physical objects. This new scenario announces a technological revolution which represents the future of computing and communications [Uni05], where the connection of the objects through the network will allow them to take an active role on the Internet, exchanging information on their identity, their physical properties and the measurements retrieved from the environment in which they are deployed. The development of this technology strictly depends on innovations in key areas such that of Wireless Sensor Network (WSN) [ASSC02,PK00], and on the information services related to the identity, status and position of the objects. In order to connect all objects and devices to the network of networks (Internet), a simple, unobtrusive and cost-effective system of node identification is crucial. After the identification, each node can transmit the data to aggregation points where they can be collected and processed. The exponential growth of Internet-connected devices has led to an increased request of IP addresses to uniquely address each device. Such a demanding request has brought the scientific community to start the standardization process of services and interfaces in order to deal with the long-anticipated problem of IPv4 address exhaustion. The outcome of this effort/process is the definition of IPv6 and of its related RFCs by the Internet Engineering Task Force (IETF) [2] organization. These new standards, compared to the previous version (IPv4), is characterized by an increased address space, while providing a simplified header format, improved support for options and stateless address auto-configuration.

Thanks to new standards and technologies, new types of devices, such as embedded sensors and actuators, can be used in the global network, making the network itself a powerful tool able to support new interesting applications with high added value. The combination of global

---


addressing (IPv6) and advanced data collection capabilities offered by embedded systems will create an Internet of Things able to connect the world objects in a innovative and intelligent manner. Moreover, recent progress in hardware technologies of embedded devices has enabled the possibility of increasing the power of WSNs by using embedded intelligence in the “things themselves”, and by distributing information processing to nodes. Industrial products and our daily life will benefit of this new concept of Internet by exploiting the smart characteristics and capabilities of the objects.

Although the integration of objects in the Internet world has begun with the adaptation of solutions coming from the WSN research area, nowadays research efforts are also oriented at extending embedded systems capabilities by integrating new powerful sensors (e.g., camera, microphone). Indeed, one of the recent advances in technology have permitted to extend the original idea of WSNs, based on simple scalar sensors (e.g., temperature, light), to what are nowadays called Wireless Multimedia Sensor Networks (WMSNs), a.k.a Smart Camera Networks (SCNs) or Wireless Camera Networks (WCNs), where more complex vectorial sensors are used (e.g., audio, image). Extension of traditional WSNs has been reached also by integrating the Radio-Frequency IDentification (RFID) technology by abstracting Readers and TAGs as WSN sensors. Such advancements open new interesting scenarios, where new technologies can be integrated in the IoT global communication network enabling a plethora of new pervasive and low-cost applications (e.g., surveillance, traffic monitoring, limited access zone).

This chapter first introduces WSNs by detailing the main research areas since their born. Then it underlines the motivation of the thesis, which can be considered a step forward the successfully evolution of WSNs into the IoT world. In the last part of the chapter main contribution and the outline of the thesis are presented.

1.1. Wireless Sensor Networks at the beginning of the IoT era

Research in Wireless Sensor Network has started over a decade ago with great enthusiasm and community expectations to revolutionize our daily life. It will not be an exaggeration to consider WSNs as one of the most researched areas in the last decade. With several applications and business opportunities arising every day, the WSN market is forecast to rise from $0.45 billion in 2012 to $2 billion in 2022 [Har12]. WSNs have the potential to drastically change a wide array of application areas by providing an unprecedented density and fidelity of instrumentation.

Wireless Sensor Networks are a steadily growing technology, that can be seen as the cornerstone of the Internet of the Things paradigm. Designed as networks composed of few units, WSNs have increased ceaselessly in complexity and size. Network structure and single node capabilities have evolved, making this technology appealing for a large number of domains. The core units of a WSN are referred as “Smart sensor node” (from now on simply referred as “nodes”), electronic devices that can extract precious information from the surrounding environment relying on different types of sensors and that can share the acquired information with the other nodes.

Decades of researches on WSNs have produced in literature several works on different domains, such as routing techniques [AKK04 EH12 AY05], MAC protocols [NS04 HXS13], data collection [DFDA11 WL11], energy conservation [ACDFP09 YJWZ11], localization
Keys aspects to be considered for the growth of WSNs and their success in our daily life are raising the level of their pervasiveness, their autonomy and abstraction for programmers. The former can be seen as a measure of the network gathering capabilities. To a great extent, an increase in the number of nodes or an enhancement in the sensing capabilities of the single node augments the pervasiveness capabilities of a WSN. Pervasiveness is generally maximized by keeping the cost of the node limited, by adopting wireless links and by exploiting small batteries, making possible the deployment of the nodes even into unfriendly and harsh environments (e.g., no connection to the power grid, difficult access to the node). Autonomy, instead, represents the longest amount of the time the node can operate without human intervention. In other words, the lifespan of a node is determined by the power consumption of the node as well as by the nature and capacity of the power supplying and power harvesting units (e.g., batteries, solar panels). In order to maximize autonomy, a typical WSN node is designed favoring low-power microcontroller (generally with a reduced features sets and slow operating speed) and low-energy wireless transmitters/receivers. Programming abstractions represents the simplicity of changing the services provided by a certain node during the WSN lifetime. The task of logic reprogramming requires expertise in various technical domains, indeed, to change the logic in a certain node requires to deal with many low levels details regarding sensing and node to node communication. In order to open WSN technology to a wider audience, the programming task must be simplified as much as possible. The level of programming abstractions is increased by using different technologies: middleware APIs, database centric, event based, virtual machines and scripts.

At the present day, a growing number of applications relies on WSNs for implementing advanced functionality, such as climate analysis [MOH04], monitoring of building structural integrity [LCL+07, THGT07], health-care systems [LNHL09] and supervision of industrial plants.

1.2. Research challenges in next generation WSN as motivation of this thesis

After numerous WSNs deployments in academic research projects, wireless sensor networks are nowadays reaching the industrial and consumer markets for large-scale deployments. As matter of example it is possible to cite the GINSENG [OBB+13] and SmartSantander [SMG+14] projects where the potential of WSNs has been proved through real large-scale deployments. Distributed smart sensors able to interact with the physical world exchanging data through wireless communications are nowadays considered the key components in the Smart City scenario. However, to reach a wide adoption of the WSNs in several domains still several limitations of current systems persist.

Following the IoT vision, WSNs have experienced a lot of progress thanks to the work done by the research community and standard organizations in promoting the adoption of standard protocols. Traditional WSNs are moving from being simple isolated monitoring systems to powerful and inter-operable systems connected to the Internet world. Nonetheless, in order to
reach a wide adoption of the WSNs in our daily life still several advances of current systems must be taken into account:

- **Protocol adaptation:** research effort at integrating WSN technology in the Internet world following the IoT vision has produced new standard protocols (e.g., 6LoWPAN, CoAP) to be integrated in WSN. The adoption of standard protocols has improved the WSN powerful in terms of interoperability, pervasiveness, scalability and abstraction. However, the use of standard protocols has led to the inoperability of the protocols designed for traditional WSNs. Hence, work on the adaptation of traditional WSN protocols must be accomplished taking into account the new benefits and the new constrains introduced by IoT protocols.

- **Combine IoT protocols with new technologies:** recent progresses in WSN have permitted to extend the original idea of WSN based on simple scalar sensors with more complex or vectorial sensors (e.g., camera, microphone, Radio-Frequency IDentification Reader (RFID-Reader), Radio-Frequency IDentification TAG (RFID-TAG)). Such advance has increased the interoperability and the power of the WSNs. Next natural step is to integrate these technologies in 6LoWPAN networks, following the IoT vision. If on the one hand the integration of different technologies in the IoT context opens WSN world to a new interesting application scenarios that can change our daily life, on the other hand raises new challenges which have to be tackled into account in order to successfully benefit of such integration. For example, video streaming in 6LoWPAN networks has to consider the limited bandwidth available Examples of application scenarios can be video surveillance or objects tracking that can be low-cost and deployed in a pervasive manner.

- **In-network processing:** advancements in hardware technologies has led to have available on the market WSN devices equipped with more powerful resources and computational capability (e.g., shifting from 8-bit to 32-bit microcontroller, from 8MHz to 80 MHz, from 10 Kbyte to 128 Kbyte of RAM, from 48 Kbyte to 512 Kbyte of Flash memory). Such advancement in hardware has enabled the possibility of increasing the power and the flexibility of WSNs by using embedded intelligence in the “things themselves”, and by distributing information processing to nodes. This distributed approach can be place side by side with the centralized approach, where all the information is transmitted to a sink node which is in charge of the data processing. The use of in-network processing allows to reduce network traffic and energy consumption, moreover it improves the reaction time and the scalability of the systems.

- **(Ease of) Reprogramming:** another requirement to be taken into account is the possibility of change remotely the services provided by a node during the WSN life cycle. Moreover, in order to open WSN technology to a wider audience, the reprogramming task must be simplified as much as possible. Indeed, reprogramming WSNs applications is challenging: developers have to be experts in various technical domains, from low-level programming to low-power networking over embedded operating systems. This
makes installing new functionality on WSNs tedious and difficult. Defining and developing new tools to facilitate WSN reprogramming is the next step to open the WSN technology to a wide number of people.

The aforementioned challenges are at the basis of this work of thesis in which all listed topics are addressed. In the next section more details on the main contributions of this thesis are provided.

1.3. Main contributions

The main contributions of this thesis are summarized as follows in respect of the main areas described in the previous Section.

- **Protocol adaptation:**
  - adaptation of the Speed geographic routing algorithm in a 6LoWPAN scenario is presented and its experimental validation in a real test bed is shown by presenting a comparison with the AODV protocol. Avoiding route creation time, and showing reduced memory occupation, load balancing and flow shaping properties Speed-6LoWPAN is an effective solution to support packet routing in 6LoWPAN networks.
  - Speed-3D, an extension of the Speed-6LoWPAN geographic routing algorithm, adapted to 6LoWPAN networks that caters for 3D routing is presented. In addition to Speed-6LoWPAN features, Speed-3D is able to work seamlessly in a 3D setting (i.e., a tall building).

- **Combine IoT protocols with new technologies:**
  - a low-complexity algorithm based on background subtraction and error resilience techniques is described. Aimed at reducing the transmission bandwidth of a video stream of uncompressed images, the proposed algorithm can reach a higher frame rate in 6LoWPAN networks.
  - the integration of WSN and RFID technologies in the IoT scenario is presented. The proposed approach is based on the REST paradigm, thanks to which the two technologies can be seamlessly integrated by representing sensors, actuators and RFID related data as network resources globally addressable through state-of-the-art IoT protocols.

- **In-network processing and (Ease of) Reprogramming:**
  - to enable application logic reprogramming on WSN node a Python-based Framework for Ubiquitous Networked Sensors, PyFUNS, is designed and implemented. PyFUNS handles low level and networking functionality, using the services provided by the Operating System, and leaves to the user only the task of application development in the form of Python scripts. This approach reduces required expertise in embedded systems to develop WSN based applications. PyFUNS also uses 6LoWPAN and CoAP standard protocols to enable interoperability and ease of integration with other systems, pursuing the Internet of Things vision.
In order to complete this work of thesis, the proposed contributions are applied to one of the most promising use cases of the smart cities scenario: Intelligent Transport System (ITS). Indeed, in the last several years a large number of projects have been realized with the aim of creating effective ITSs in which useful and cost-effective services can be provided to the final users. In this direction the European Union (EU) has progressively increased its contribution in funding research activities aiming at: (i) making optimal use of road, traffic and travel data; (ii) guaranteeing continuity of traffic and freight management in ITS services, and (iii) creating ITS road safety and security applications. Such requirements have been specified by the EU in the Directive 2010/40/EU [EUd10].

1.4. Outline of this thesis

The rest of this thesis is organized as follows:

Chapter 2 describes the adaptation of geographic routing protocol for 6LoWPAN networks. The chapter presents also the performance evaluations in a real scenario of the proposed routing protocol in order to prove its effectiveness to support packet routing in 6LoWPAN networks.

Chapter 3 focuses on enabling new key technologies in 6LoWPAN networks. Two main technologies have been considered: image sensing in WSN and Radio-Frequency IDentification. Regarding the image sensing in WSN, a low-complexity algorithm for enabling video streaming of uncompressed images is presented. Concerning RFID, the integration of WSN and RFID technologies in the IoT scenario is proposed, and a real use case is described to underline the power of such integration.

Chapter 4 presents a proposed high-level abstractions for IoT-based WSNs. The proposed approach reduces required expertise in embedded systems to develop WSN based applications and it uses 6LoWPAN and CoAP standard protocols to enable interoperability and ease of integration with other systems, pursuing the Internet of Things vision.

Chapter 5 proposes the use of IoT in an Intelligent Transport System scenario. The proposed use case shows the integration between IoT based networks and Wireless Camera Networks.

Chapter 6 concludes this thesis and provides concluding remarks.
CHAPTER 2

Internet of Things Protocols Adaptation

RESEARCH efforts in the last several years have been oriented at integrating the Wireless Sensor Network technology in the Internet world following the Internet of Things vision.

The IoT is the envisioned extension of the Internet to physical objects: every “things” will be connected to the Internet through wired or wireless links with a global address to create a single world-wide network of electronic devices. In such a scenario, the integration of the WSN technology in the Internet world is a necessary step towards the full accomplishment of the IoT vision. An example of application where WSNs fully accomplish the IoT vision is in smart buildings, where sensor devices and smart objects communicate and cooperate with each other to achieve a set of global goals, like energy saving, temperature control and air conditioning, access control to specific areas, etc. Enhancement can be achieved also in industrial automation that may rely on sensor devices to coordinate and monitor production processes and to perform access control, enforcing as well, safety regulations. Whereas such applications can be realized using dedicated local networks, connections to Internet open new perspectives in terms of re-usability, ease of deployment and standardization.

If on the one hand the development of Internet addressable WSNs opens interesting application scenarios, on the other hand it raises new challenges which have to be tackled in order to successfully benefit of such integration. In the IoT vision, each device must be uniquely addressable within the network and the communication must be based on standard protocols. These two main requirements are fulfilled in high-end devices by using IPv6 addressing and the Internet Protocol Suite (IPS), while considering tiny mote devices the IPv6 addressing method and all the other protocols must be modified or adapted to match the WSN requirements. In WSNs, in fact, the use of IPv6 based addressing and protocols should consider the bandwidth limitations imposed by Medium Access Control (MAC) mechanisms adopted in such networks. In other words, the use of the IPv6 based communication mechanisms in WSNs requires an adaptation of the original standard developed for high-end devices. For WSNs compliant with the IEEE 802.15.4 [Soc11] standard, the IPv6 suite has been standardized by the IETF with the name of 6LoWPAN [KMS07].

Contributions The rest of the chapter discusses the author’s contributions to the topic protocol adaptation in the direction of enabling effective IoT systems. After an overview of the original work targeted to WSNs, Section 2.3, a first adaptation of the SPEED geographic routing algorithm in the 6LoWPAN scenario is presented [BPP+11] in Section 2.4 Section 2.5

presents Speed-3D, an extended work of Speed-6LoWPAN that caters for 3D routing \[^{[BPP*]}\]. Avoiding route creation time and showing reduced memory occupation, load balancing and flow shaping properties Speed-3D is an effective solution to support packet routing in 6LoWPAN networks.

2.1. Background

IoT-based communication rely on standard protocol solutions covering all the layers of the well-known Internet Protocol suite. The standard protocol, specifying both the Physical (PHY) and Medium Access Control (MAC) sub-layers of the ISO/OSI communication model, is the IEEE 802.15.4 \[^{[Soc11]}\]. The standard has been released in its first version in the 2003 with the aim of enabling energy-efficiency communications in Low-Rate Wireless Personal Area Networks (LR-WPANs). In the 2007 Kushalnagar et al. \[^{[KKGB12]}\] proposed the adaptation of the IPv6 protocol over Low power Wireless Area Network (6LoWPAN), thus specifying a Network (NET) layer for enabling Internet like communication in IEEE 802.15.4-based networks. The 6LoWPAN concept comes from the idea that "the Internet Protocol could and should be applied even to the smallest devices" \[^{[Mul07]}\], and that low-power devices with limited processing capabilities should be able to participate in the envisioned IoT. 6LoWPAN defines the frame format for the transmission of IPv6 packets, as well as mechanisms for header compression, and formalizes how to create IPv6 global addresses on top of IEEE 802.15.4 networks. Along with the definition of IPv6-based communications in standard wireless sensor networks, another major point to consider is related to routing protocols. Routing issues are very challenging for 6LoWPAN due to the low-power nature of such networks, multi-hop mesh topologies to be managed, and topology changes due to node mobility. Successful solutions should take into account the specific application requirements, along with IPv6 behavior and 6LoWPAN mechanisms. The IPv6 Routing Protocol for Low power and Lossy Networks (RPL) \[^{[WTB+12]}\] can be considered to be the state-of-the-art routing algorithm developed by the networking community. RPL has been proposed by the IETF Routing over Low-power and Lossy networks Working Group (ROLL) as a standard routing protocol for 6LoWPAN, since existing routing protocols do not satisfy all the requirements for Low power and Lossy Networks (LLNs).

2.2. Related works

Since the first 6LoWPAN document release, many IPv6-based protocols have been adapted for the IoT vision. The most significant solutions for each routing protocol category are briefly introduced in the following.

Routing protocols can be categorized into flooding-based, hierarchical-based and geographic-based protocols. Flooding-based routing protocols are built on the idea of flooding the network with control packets, to create an effective sender-receiver transmission route in case of expressed request (reactive protocols) or in a proactive way (proactive protocols). The most popular protocol in this category is Ad-Hoc on Demand Distance Vector (AODV) \[^{[PBRD03]}\], a reactive protocol. Several 6LoWPAN adaptations have been proposed as enhancements of AODV, such
as 6LoWPAN Ad-hoc On-Demand Distance Vector (LOAD) \cite{KDPM07}, Multipath-based 6LoWPAN Ad-hoc On-Demand Distance Vector (MLOAD) \cite{CCYC10}, Dynamic MANET On-demand for 6LoWPAN (DYMOLow) \cite{KMP07}, and Sink Routing Table over AODV (S-AODV) \cite{CL10}. Concerning proactive protocols, a notable proposal is Optimized Link State Routing (iOLSR) \cite{LFPL11}. In general, flooding protocols need to broadcast a large amount of packets wasting significant amounts of energy, therefore they are not well suited to WSNs. In particular, proactive protocols create all routes during the network initial setup, and then need to periodically update the network routing tables during operation, even for unused routes. The update mechanism leads to a waste of energy and communication bandwidth that is unacceptable in WSNs.

Hierarchical-based protocols are mainly adopted in cluster-based networks in which data transmission goes from cluster members to cluster heads. 6LoWPAN routing protocols belonging to this class are: Hierarchical Routing over 6LoWPAN (HiLow) \cite{KYP07}, Collection Tree Protocol (CTP) \cite{GFJ09}, RPL \cite{WTB12}. An enhanced version of above cited protocols is Improved HiLow (I-HiLow) \cite{YH11}. Cluster-based networks present little adaptability to dynamic changes in the network, and need a proper set-up of the network. For example, when two neighbor nodes belong to different network sub-tree, the communication path needs to pass through a common ancestor in the tree, thus increasing the communication delay. This drawback is addressed by P2PRPL \cite{GBBM13,BPG11} which tries to create direct links between neighbor nodes. Also, cluster-based protocols do not scale well with the size of the network and the covered area.

### 2.3. Speed protocol overview

**Speed** is a routing protocol that supports soft real-time communications in large-scale sensor networks. The end-to-end soft real-time is achieved by maintaining a desired delivery speed across the network by means of feedback control and non-deterministic geographic forwarding. The use of geographic location for packet forwarding strictly requires that each node in the network is Geo-referenced. The protocol provides three types of soft real-time communication services:

- **real-time unicast**: a packet is sent to a specific node within the network, which is identified by its geographic position and global network address;
- **real-time area-multicast**: this service allows to send a data packet to all the nodes inside a destination area identified by its center (expressed in geographic coordinates) and radius;
- **real-time area-anycast**: a packet is sent to at least one node inside an area specified by its center (expressed in geographic coordinates) and radius.

According to its main working principle, **Speed** aims at delivering a data packet to the destination node with a transmission speed faster than a certain desired value, the *speed parameter*, \( v \). The delivery speed is guaranteed hop by hop by covering the next-hop distance, \( Ad \), in a bounded time equal to \( Ad/v \). As a consequence of such a delivery strategy, the shortest path between sender and receiver is selected for all the sent packets, moreover no transmissions in
the opposite direction are allowed because of negative speed values. The speed-based forwarding mechanism provided by the protocol is depicted in Figure 2.1 in which a network of five nodes is represented: the sender (S), the destination (D) and three other nodes (Hi, Hj, and Hk).

Figure 2.1. The speed-based forwarding mechanism in Speed.

When sending a packet from node S to node D it is possible to identify two possible space regions originated by S: (i) the Sender Radio Range (SRR), which is centered in S and contains all the nodes directly reachable from S, and (ii) the Sender Feasible Candidate (SFC), which has the S in its edge and contains all the possible next-hop candidates. Recalling the example in Figure 2.1 nodes Hi, Hj, and Hk are in the SRR region because directly reachable from S, while only nodes Hi and Hj (having positive speed) are in the SFC region, whereas node Hk is not a feasible solution (having negative speed). Hence, the next-hop will be selected between Hi and Hj. In case both nodes present the same transmission delay, node Hj will be selected because it is closer to destination, thus maximizing the delivery speed. The key features of the protocol can be summarized as follows:

1. Stateless architecture. In forwarding packets towards the destination the protocol only maintains immediate neighbor information, while avoiding to store a full routing table. Such a feature guarantees to reduce the memory required by the protocol, thus really becoming a suitable solution for WSNs in which nodes are usually resource constrained (both in computational capabilities and memory availability);

2. Soft real-time. The possibility of associating a speed threshold at each packet transmission permits of having a per packet based upper bound of the maximum transmission delay, thus supporting soft real-time applications in a WSN scenario;

3. Minimum MAC layer support. In order to provide soft real-time mechanisms, Speed does not require real-time or QoS-aware MAC layers, but its working principle is compatible with all the MAC layers providing a best effort service;

4. QoS routing and congestion management. In reactive routing protocols, when a route gets congested it is necessary to find a new path restarting the route discovery procedure. In Speed, congestion is managed with feedback control and Backpressure
rerouting schemes. When a node $N$ realizes through feedback information coming from neighbors that the elected next-hop $M$ is congested, it starts a Backpressure procedure to select a new feasible next-hop for future transmissions. More in detail, the node $N$ sends to its neighbors a Backpressure message signaling the possibility of reaching the node $M$ in the congestion area with a delay which is the average of the whole area. Bigger delays result in slower velocities, thus favoring the selection of an alternative route outside from the congested area, and consequently allowing to reduce the congestion in the area;

(5) **Traffic load balancing.** One of the most challenging design constraint in WSNs is energy consumption. The sensor devices deployed in a certain environment must survive as long as possible with a small battery. Because of this, it is worth to use several possible paths to send packets from sender to destination. The absence of deterministic routing table in Speed from one hand requires that at each hop the next-hop has to be evaluated, while on the other hand it permits to balance each flow among multiple concurrent routes;

(6) **Localized behavior.** In a network composed by several devices, an algorithm is said to be localized if any action invoked by a node does not affect the operation of the whole system. This is not the case of reactive routing protocols, in which the route formation procedure requires the use of the flooding mechanism to discover new paths. Packet flooding may result in network congestion and consequently system failure. In Speed all distributed operations are localized thus achieving high system scalability;

(7) **Void avoidance.** Speed is not a pure greedy geographic algorithm in finding a forwarding route. If a path exists, even if it is not a greedy one, the Speed algorithm is able to discover it, thus guaranteeing a void avoidance property. This feature is reached by the protocol by using Backpressure messages which signal the possibility of reaching a certain destination area with an infinite delay, thus forcing forwarding nodes to discover an effective route after several attempts.

### 2.4. Speed-6LoWPAN

This section presents the extension of the Speed geographic routing algorithm target to 6LoWPAN networks. To reach a seamless integration of Speed in 6LoWPAN networks the following steps have been performed: (i) protocol messages adaptation using ICMPv6 messages; (ii) flow shaping capability added to the protocol by using functionality provided by the IPv6 standard; (iii) code source implementation in ERIKA OS; (iv) experimental validation in a real test bed and comparison with the AODV protocol.

#### 2.4.1. Routing protocol adaptation

As already mentioned, 6LoWPAN refers to the adaptation of IPv6 addressing and protocols in IEEE 802.15.4 networks, better known as Wireless Personal Area Network (WPAN). According to the ISO/OSI model, the IEEE 802.15.4 standard only specifies the physical and the medium access control layers, whereas 6LoWPAN defines the network level.
The first version of the IEEE 802.15.4 standard has been published in 2003 with the goal of enabling low-speed ubiquitous communication among devices with reduced computational capabilities and strong power saving requirements. To achieve such objectives the standard guarantees a maximum transmission rate of 250 kbps with a maximum packet size of 127 bytes.

The first problem in adapting the IPv6 standards for IEEE 802.15.4 refers to the IPv6 address size. Indeed, according to the IETF specifications the IPv6 address is 128 bit long, and its encapsulation in IEEE 802.15.4 MAC data packets requires header compression functionality to save useful bytes for application purposes. In Figure 2.2 a comparison of IPv6 and 6LoWPAN header occupancy is reported. The use of header compression guarantees an overhead reduction equal to 40% (13 bytes for the 6LoWPAN against 64 bytes for the IPv6).

![IPv6 vs. 6LoWPAN encapsulation in IEEE 802.15.4 data frame.](image)

In 6LoWPAN networks the adopted routing protocols require to support both mesh and star topologies, as reported in [KKGB12]. To perform an effective communication among devices, 6LoWPAN nodes require the implementation of Neighbor Discovery (ND) services [NNSS07, SCNB12] and network management functionality, provided by the Internet Control Message Protocol version 6 (ICMPv6). In particular, the use of the Router Advertisement (RA), Router Solicitation (RS), Neighbor Advertisement (NA) and Neighbor Solicitation (NS) messages permits to update periodically the status of the network, thus reacting to network changes.

The adaptation of the Speed protocol in a 6LoWPAN context requires to define encapsulation strategies for the protocol messages, as well as optimization based on the services provided by the ICMPv6 protocol. Speed-6LoWPAN operates in multi-hop networks by means of three messages:

1. **Beacon message.** This message is periodically broadcast by each node to announce its presence to neighbors. The main information carried out by the message is a `GlobalNodeID`, its geographical position, and a delay estimation in possible packet forwarding. All these data are kept by neighbors and stored in a table which has four entries: `NeighborID`, `Position`, `SendToDelay`, `ExpireTime`. The latter is used to validate the entries in the neighbor table. If a node does not hear from a neighbor for a time longer than `ExpireTime`, the appropriate entry into the table is deleted.
(2) **Backpressure beacon message.** This kind of message is used both to reduce network congestion and to solve the void avoidance problem. When a node receives a packet to be forwarded towards the destination while it is experiencing congestion problems, it broadcasts to its neighbors a Backpressure message. In such a message the node announces its GlobalNodeID together with the commitment to reach a certain Destination with a new SendToDelay value (congestion management). If the node can not reach the destination due to unfeasible paths, SendToDelay is set equal to infinity (void avoidance).

(3) **Data packet message.** Speed-6LoWPAN is a pure localized protocol. The routing operations are not distributed, but localized in each node. This means that the protocol does not need to send packets to build the route, but the next-hop in the path is locally selected. This behavior of the protocol on the one hand guarantees high scalability to the system, on the other hand requires that each data packet must include some additional information such as PacketType (unicast, area-multicast or area-anycast), GlobalNodeID (used for unicast transmission), DestinationArea (center and radius) and TTL (packet time to live in the destination area for broadcasting purposes).

Each protocol message requires to identify the geographical position of the node by means of global coordinates. To this end, Speed-6LoWPAN adopts the coordinates assigned by the Global Positioning System (GPS), consisting of latitude, longitude. Regarding the GlobalNodeID, this is the IPv6 address associated to the device. As previously introduced, to implement Speed-6LoWPAN in an effective and efficient way all the possibilities provided by the ICMPv6 protocol have been investigated. In order to reduce the protocol overhead, as number of transmitted packets, the periodic Beacon message is sent through the RA message provided by the ICMPv6, and more specifically using the Source/Target Link-Layer Address Option (SLLAO) \[NNSS07\].

The RA message is sent periodically, as required by the beacon message, from each node to advertise their presence in the network. The packet encapsulation is depicted in Figure 2.3. Starting from left to right there are the MAC, 6LoWPAN and ICMPv6 headers respectively. After the SLLAO field follow all the Speed-6LoWPAN parameters:

- **Type:** identifies the type of the message;
- **Flow label:** this field is an addendum of the Speed-6LoWPAN beacon message in order to support flow labeling based on the functionality provided by the IPv6 standard \[ACJR11\]. The parameter indicates to which flow the node belongs;
- **Res:** reserved bits for future expansions;
- **Position parameters:** latitude and longitude of the node;
- **SendToDelay:** delay estimation for forwarding packets towards next-hops.

A three bytes padding field has been added after the SendToDelay parameter in order to follow the requirements from the standard (SLLAO plus additional fields must be multiples of 8 bytes).

The Backpressure beacon message is not periodic, and it is sent in regular 6LoWPAN messages. The Speed-6LoWPAN fields added to the packet are:

- **Type:** identifies the type of the message;
2. INTERNET OF THINGS PROTOCOLS ADAPTATION

Beacon message

<table>
<thead>
<tr>
<th>9 B</th>
<th>4 B</th>
<th>64 B</th>
<th>10 B</th>
<th>14 B</th>
<th>2 B</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>Header</td>
<td>IPv6</td>
<td>ICMPv6 Fields</td>
<td>SLLAO</td>
<td>SPEED Fields</td>
</tr>
</tbody>
</table>

3 b 3 b 1 b 1 b 4 B 4 B 2 B 3 B

Type Flow Label NS Ind EW Ind Latitude Longitude Send to Delay Padded with 0

Data packet message

<table>
<thead>
<tr>
<th>9 B</th>
<th>11 B to 20 B</th>
<th>4 B</th>
<th>12 B</th>
<th>66 B to 75 B</th>
<th>2 B</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>Header</td>
<td>IPv6</td>
<td>EHF</td>
<td>SPEED Fields</td>
<td>Payload</td>
</tr>
</tbody>
</table>


Type Res SD LaD SD LoD NH LaD NH LoD Radius Padded with 0

Backpressure beacon message

<table>
<thead>
<tr>
<th>9 B</th>
<th>3 B</th>
<th>3 b</th>
<th>3 b</th>
<th>1 b</th>
<th>1 b</th>
<th>4 B</th>
<th>4 B</th>
<th>2 B</th>
<th>2 B</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC</td>
<td>Header</td>
<td>IPv6</td>
<td>Type</td>
<td>Res</td>
<td>NS Ind</td>
<td>EW Ind</td>
<td>Latitude</td>
<td>Longitude</td>
<td>Send to Delay</td>
</tr>
</tbody>
</table>

Res: reserved bits for future expansions;
Position parameters: latitude and longitude of the node;
SendToDelay: delay estimation for congestion control and void avoidance purposes.

Regarding the Data packet message, must be again outlined that Speed-6LoWPAN requires to include additional information in each data packet in order to locally select the next-hop. Such additional parameters must be encapsulated in 6LoWPAN packets adding an additional Extension Header Fields (EHF), as described in [CJ13]. The routing Speed-6LoWPAN parameters added to each data packet are depicted in Figure 2.3 and listed in detail below:

- Type: identifies the type of the message;
- Res: reserved bits for future expansions;
- SD_xxD: latitude (SD_LaD) and longitude (SD_LoD) differences between sender and destination;
- NH_xxD: latitude (NH_LaD) and longitude (NH_LoD) differences between next-hop and destination;
- Radius: radius parameter to be used in case of area-multicast and area-anycast messages.
2.4. Speed-6LoWPAN

The geographic destination of the packet is due to NH_LaD and NH_LoD parameters. The difference between the next-hop position and the destination is used for header compression purposes. The SD_LaD and SD_LoD parameters are used for end-to-end communications between source and destination, thus avoiding the use of an additional table for mapping the GlobalNodeID with its geographical position. The TTL (packet time to live in the destination area) has not been inserted as explicit Speed-6LoWPAN field because it can be extracted from the 6LoWPAN header. By recalling the contributions of this work, it must be stressed that the flow-shaping capability has been added by introducing a specific field in the Beacon message.

2.4.2. Implementation in ERIKA OS

Speed-6LoWPAN has been implemented in the ERIKA Enterprise Real-Time Operating System (RTOS) network stack. ERIKA is a multi-processor real-time OS which architecture is composed by three layers: hardware abstraction, kernel and application. The kernel layer implements the tasks management and tasks scheduling policies. Tasks in ERIKA have a run-to-completion semantic, and are scheduled according to fixed and dynamic priorities with the possibility to share resources among them. Interrupts preempt the running task to execute urgent operations required by devices.

Speed-6LoWPAN has been implemented in the ERIKA network stack on top of µWireless, a custom implementation of the IEEE 802.15.4 MAC layer. µWireless supports packet transmission in both peer-to-peer and star topologies. Moreover, it is possible to create both beacon-enabled Personal Area Networks (PANs) and non beacon-enabled PANs. The former enables the possibility to perform slotted Carrier Sense Multiple Access with Collision Avoidance (CSMA-CA) mechanism while achieving synchronization among nodes. In this case the device communication can be based on Guaranteed Time Slots (GTSs). In the non beacon-enabled PANs the devices can transmit their data using unslotted CSMA-CA. In this case, nodes are not synchronized (synchronization can be achieved by application-level protocols). However, larger networks topologies are possible by forming a mesh of multiple neighborhood clusters. In order to respond to the requirements coming from 6LoWPAN, Speed-6LoWPAN has been implemented using the unslotted mechanism to create a complex network over a multi-hop topology.

According to the original Speed algorithm, Speed-6LoWPAN implementation provides four possible Application Programming Interfaces (APIs):

- **UnicastSend**: sends a packet to a node identified by its IPv6 address;
- **AreaMulticastSend**: sends a packet to nodes in an area identified by its center and radius;
- **AreaAnyCastSend**: sends a packet to at least one node in an area identified by its center and radius;
- **SpeedReceive**: accepts packets delivered to the node.

To support the four APIs the protocol software architecture has been divided, as depicted in Figure 2.4, into six interacting modules:
- **Stateless non-deterministic geographic forwarding.** This module is the core of Speed-6LoWPAN and it is in charge of the next-hop selection. In selecting the next-hop, the module requires information and functionality from all the other modules;

- **Beacon exchange.** The main objective of this module is to periodically send the beacon message to neighbors for filling the neighbor tables;

- **Delay estimation.** This module measures the delay for a one-hop transmission. The delay is measured at the sender node when it receives acknowledgment for the data. The measured delay is used to evaluate the network congestion;

- **Neighborhood feedback loop.** The main goal of this module is to maintain the single hop relay speed. Its main characteristics are discussed in [SHA+01]. Basically the module provides a control of the forwarding speed for evaluating misses in reaching the target value. This information is forwarded to the stateless non-deterministic geographic forwarding module which can decide for faster routes;

- **Backpressure rerouting.** This module works with the stateless non-deterministic geographic forwarding and the neighborhood feedback loop modules. Once network congestion is detected the module starts a Backpressure rerouting procedure sending a Backpressure beacon message to neighbors.

- **Last mile process.** This module is activated in the delivering area: it manages the packet delivery to the final destination (unicast). Whenever the packet is sent via multicast and the node is one of the destinations the packet is both pushed to the transport layer and resent to other destinations.

The Speed-6LoWPAN routing protocol has been inserted into the ERIKA 6LoWPAN implementation according to the interaction diagram represented in Figure 2.5 as proposed in [TDJ+07].

In the 6LoWPAN implementation design the input and output queues manage the incoming and outgoing messages, respectively. The forwarding engine analyses each incoming packet in order to evaluate whether the packet has to be forwarded to other devices, in which case it is enqueued into the output queue, and the routing protocol services are invoked.
2.4.3. Performance evaluation in a real scenario

The performance of the Speed-6LoWPAN protocol has been evaluated in a real indoor scenario with a performance comparison with respect to the 6LoWPAN AODV. In the following of this Section the test bed scenario and network topology is first discussed, then the adopted hardware is briefly described before the results about performance comparison. The key metrics adopted in comparing the two protocols are: Route Formation Time (RFT), routing table memory occupancy, Round-Trip Time (RTT) and path management properties.

Test bed scenario and network topology

The network test bed has been deployed in an indoor scenario consisting of corridors and rooms of the “Institute of Communication, Information and Perception Technologies” (TeCIP), Pisa, Italy. The selected environment guarantees a physical connectivity among devices due to the short transmission distances involved. Moreover, no obstacles have been placed along each communication path (line-of-sight transmission), thus avoiding packet losses and additional delays in route creation. In Figure 2.6 the network configuration adopted for the performance evaluation experiments is presented. The network is composed by nine nodes, among them the 0x21 has been selected as source node while the 0x28 is the destination node for each performed analysis. Neighbor nodes have been depicted by means of line connections on which the distance between the nodes is reported. The dimension of each data packet has been set equal to 20 bytes, while each wireless communication has been performed at 2.4 GHz with a transmission power of 1 mW.

Flex development board

The adopted hardware is the Flex Board (Base Board plus Demo Board) equipped with a CC2420 transceiver, Figure 2.7. The Flex Base Board is equipped with a Microchip dsPIC-33F 16bit-microcontroller which operates at 40 MIPS and has 30 KB of internal RAM memory and 256 KB of flash memory. The board exports all the pins of the microcontroller in order to connect additional boards with different sensor devices, such as the Flex Demo Board. The Flex Demo
Board is a daughter board suited for prototyping and laboratory experiments. The features hosted on the FLEX Demo Board are 2 DAC outputs, a 3-axis accelerometer, push buttons, LEDs, an LCD, a buzzer, a potentiometer, a thermal sensor, a light sensor, an InfraRed device, and a CC2420 connector. The CC2420 is a single chip IEEE 802.15.4 compliant R/F transceiver operating in the 2.4 GHz band.

![FLEX development board](image)

**Figure 2.7.** FLEX development board used in performance evaluation. The board is equipped with three different sensors: a 3-axis accelerometer, a temperature sensor, and a light sensor.

**Performance results**

SPEED-6LoWPAN and AODV performance in 6LoWPAN networks have been compared according to the metrics introduced in the beginning of this Section. Before presenting and discussing all the results, a briefly introduction of AODV is presented, namely explaining how it creates a route between two nodes and stressing on its main property.

AODV is a reactive routing protocol; this means that a node starts a route discovery throughout the network, only when it wants to send packets to a certain destination. The sender node starts the route discovery with a Route Request (RREQ) packet which is sent to one-hop distance nodes (TTL of the packet equal to one). If the destination is not reached then the RREQ
packet is sent again embedding an increased value for TTL (measured in number of hops). Once the destination is reached, a Route Reply (RREP) packet is sent back from the destination to the source. The route creation mechanism is targeted to find the shortest transmission path between sender and destination based on the minimum number of hops. This working paradigm is completely different from the one of SPEED in which the shortest transmission path is evaluated with the goal to cover the minimum geographical distance.

In the following relevant comparison results are presented and discussed. It is important to underline at this point that all the experienced time delays include the network stack delay, thus explaining differences with respect to other works, such as GSAP06 in which other performance evaluation for AODV protocol has been performed.

Route formation time. As mentioned above AODV creates a route between sender and destination on-demand only. The full route discovery process requires time and a large number of packets to be sent within the network, having as final result a path with the minimum number of hops. In the considered network scenario the AODV path from node 0x21 to node 0x28 is depicted in Figure 2.8. It counts four hops only. Figure 2.9 shows the route formation time distribution of above one thousand entries, the average time is equal to 192.3 ms.

Concerning the SPEED-6LoWPAN protocol the route formation time is equal to zero. This is because SPEED-6LoWPAN does not need to create a route before sending a packet from source to destination. The path is created hop by hop sending the packet to the next-hop to reach the final destination covering the minimum geographical distance. Moreover, the presented adaptation of the SPEED-6LoWPAN protocol does not require additional messages for announcing a node to its neighbors. This is because of the encapsulation of the beacon messages in the Router Advertisement messages provided by the ICMPv6, periodically sent by the nodes.

Routing table memory occupancy. In discovering the transmission path the two protocols need to store some information in the device memory. The memory occupation in each node must be as small as possible because of the reduced memory capabilities of the devices. For each routing table entry AODV needs to store the following parameters: Destination IP (8 bytes), Destination Sequence Number (4 bytes), Valid Sequence Number (1 bit), State (1 byte), Hop Count (1 byte), Next-hop (8 bytes) and Precursor (8 bytes), for a total amount of 30 bytes. In the presented SPEED-6LoWPAN adaptation the only additional parameters to be stored are:
Latitude (4 bytes), Longitude (4 bytes), North\South indicator (1 bit), East\West indicator (1 bit), SendToDelay (2 bytes) and ExpireTime (2 bytes), for a total amount of 12 bytes. The reduction in memory occupancy for each entry is equal to 60%. Even if Speed-6LoWPAN reduces the amount of memory for each routing table entry it requires to store data in all nodes having neighbor. This is not necessary in AODV where the entry is created on-demand. In large sparse networks Speed-6LoWPAN requires less memory with respect to AODV because of the reduced number of entries and their smaller size.

**Round-Trip Time.** The RTT experienced by the two routing protocols has been measured with the aim of evaluating the overhead introduced by Speed-6LoWPAN in choosing the next-hop at each intermediate node. The transmission path for AODV is depicted in Figure 2.8, it is the same for transmissions from node 0x21 to 0x28 and and vice versa. For a fair comparison with Speed-6LoWPAN the neighbor table for each node has been preset in order to have as neighbors only nodes which are at one-hop distance. For packet transmission from node 0x21 to node 0x28 the path is the same as for AODV, while varies from 0x28 to 0x21, as depicted in Figure 2.10 Speed-6LoWPAN selects the next-hop having as main objective to reduce the distance to the final destination. In case of packets transmission from 0x21 to 0x28 as a packet reaches the node 0x23, the 0x25 is selected as next-hop because it is closer to the final destination. Instead, when a packet is sent back from node 0x28, the furthest reachable node is the 0x26, thus selected as next-hop.

The RTT distributions for both protocols are shown in Figures 2.11a and 2.11b. AODV shows an average Round-Trip Time value equal to 185.9 ms, while the value measured with Speed-6LoWPAN is equal to 190.7 ms. The difference between the two protocols is small (equal to 5.2 ms) and it is related to the computation overhead introduced by Speed-6LoWPAN for the selection of the next-hop.

**Path management: load balancing.** Another result of the previous analysis is the intrinsic load balancing characteristic shown by Speed-6LoWPAN. The AODV path selection
based on minimum number of hops does not guarantee to share the load among nodes. In the selected scenario 100% of the packets are forwarded by the node 0x25 when AODV is used, while its load is halved when using Speed-6LoWPAN. In a WSNs context this is a notable property, a device which sent half number of packets almost doubles its life in the network.

Path management: flow shaping. In the proposed Speed-6LoWPAN adaptation in the beacon message a Flow label field for supporting per flow packet priorities has been added. Each node in the network announce to its neighbors not only its position, but also the priority flow it belongs to. For this experiment the network topology has been modified in order to better show this characteristic, more in particular the link between nodes 0x23 and 0x25 has been deleted. The new network topology is depicted in Figure 2.12. The nodes connected with the dashed line support high transmission priority, while the nodes connected with a continuous line support low transmission priority. In the route from 0x21 to 0x23 there is only a possible transmission path which is used for sending packets belonging to both flows. As the path forks, if a packet belongs to the high priority flow it is forwarded along the short path, thus guaranteeing a higher delivery
speed. The packets selected for a low priority transmission are sent toward the destination across a longer path, thus reducing the delivery speed. In Figure 2.13 is reported the RTT distributions for both priority flows. Packets sent with a high priority show an average RTT equal to 196.2 ms, while the average RTT is equal to 244.5 ms in case of low priority transmission. The flow shaping mechanism is not supported in AODV because of its property of selecting at each time the shortest path in number of hops. Multiple paths are not supported.

![Figure 2.12. Transmission paths for flow shaping property evaluation.](image)

![Figure 2.13. Speed-6LoWPAN round trip time for high and low priorities.](image)

Starting from the presented performance, Speed-6LoWPAN shows a negligible bigger round trip time with respect to AODV while saving time in route creation and routing table memory occupancy. Moreover Speed-6LoWPAN supports soft real-time, load balancing and flow shaping mechanisms making itself an effective solution in supporting packet routing in 6LoWPAN networks.
This section presents SPEED-3D, an extended work of SPEED-6LoWPAN, that caters for 3D routing. In the proposed 6LoWPAN based 3D all the intrinsic characteristics of the original protocol are preserved, while adding several new features. More in detail, in the presented work: (i) the third dimension (altitude) is introduced to the position of nodes in order to easily support routing in tall buildings; (ii) enhancement of the modular task design in ERIKA OS to minimize the delay of message forwarding, and to control the execution of the network stack is discussed; (iii) performance of SPEED-3D in a realistic indoor scenario in terms of RTT and Packet Lost Rate (PLR) is evaluated, with the aim of providing an analytic model to estimate the Round-Trip Time in large-scale 6LoWPAN networks.

Concerning geographic routing protocol, the third dimension is an important feature to be added in large-scale deployment. Indeed, the third dimension improves and refines the next-hop selection, bringing advantages in terms of energy saving (locally and in the whole network), congestion and time to delivery.

### 2.5.1. Routing protocol adaptation

The 3D routing has been enabled by adapting protocol messages, adding the third dimension in each packet, and next-hop selection procedures. In Figure 2.14 the new SPEED-3D packets format is depicted. Starting from *Beacon message*, after the SLLAO field follow all the SPEED-3D parameters:

- **Type**: identifies the type of the message;
- **Flow label**: this field is an addendum of the SPEED-3D beacon message in order to support flow labeling;
- **Res**: reserved bits for future expansions;
- **Position parameters**: latitude, longitude and altitude of the node;
- **SendToDelay**: delay estimation for forwarding packets towards next-hops.

The *Backpressure beacon message* fields are:

- **Type**: identifies the type of the message;
- **Res**: reserved bits for future expansions;
- **Position parameters**: latitude, longitude and altitude of the node;
- **SendToDelay**: delay estimation for congestion control and void avoidance purposes.

The routing SPEED-3D parameters added to each data packet are depicted in Figure 2.14 and listed in detail below:

- **Type**: identifies the type of the message;
- **Res**: reserved bits for future expansions;
- **SD_xxD**: latitude (SD_LaD), longitude (SD_LoD), and altitude (SD_AlD) differences between sender and destination;
- **NH_xxD**: latitude (NH_LaD), longitude (NH_LoD), and altitude (NH_AlD) differences between next-hop and destination;
- **Radius**: radius parameter to be used in case of area-multicast and area-anycast messages.
Beacon message

```
<table>
<thead>
<tr>
<th>MAC</th>
<th>Header IPv6</th>
<th>ICMPv6 Fields</th>
<th>SLLAO</th>
<th>SPEED Fields</th>
<th>FCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 b</td>
<td>3 b</td>
<td>2 b</td>
<td>4 B</td>
<td>2 B</td>
<td>1 B</td>
</tr>
<tr>
<td>Type</td>
<td>Flow Label</td>
<td>Res</td>
<td>Latitude</td>
<td>Longitude</td>
<td>Altitude</td>
</tr>
</tbody>
</table>
```

Data packet message

```
<table>
<thead>
<tr>
<th>MAC</th>
<th>Header IPv6</th>
<th>EHF</th>
<th>SPEED Fields</th>
<th>Payload</th>
<th>FCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 b</td>
<td>5 b</td>
<td>2 B</td>
<td>2 B</td>
<td>2 B</td>
<td>5 B</td>
</tr>
<tr>
<td>Type</td>
<td>Res</td>
<td>SD LaD</td>
<td>SD LoD</td>
<td>SD</td>
<td>NH LaD</td>
</tr>
</tbody>
</table>
```

Backpressure beacon message

```
<table>
<thead>
<tr>
<th>MAC</th>
<th>Header IPv6</th>
<th>Type</th>
<th>Res</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Altitude</th>
<th>Send to Delay</th>
<th>FCS</th>
</tr>
</thead>
</table>
```

Figure 2.14. Speed-3D messages encapsulation.

2.5.2. Implementation in Erika OS

As described in [NPM+09], in Erika the MAC functionality is mapped onto real-time tasks scheduled following a priority-based mechanism. The same paradigm has been adopted to implement the network layer services, while carefully considering the execution pattern and the interference arising from the scheduling of MAC services. According to Figure 2.15, four main tasks manage both incoming and transmitted packets in both MAC and NET layer.

Task **MacRXData** is activated asynchronously, by means of an interrupt, when a new data packet is received at MAC layer. The task is in charge of queuing up the message to the network layer input queue. The **NetProcessRX** task is activated periodically to check the input queue of the network layer. If a message is found in the queue, a specific function is invoked with the following rational: if the packet is not for the node itself, the forwarding engine module is called. This module creates the 6LoWPAN packet, interacts with the routing protocol module, sets some fields in the header, and pushes the message into the network layer output queue. The period of **NetProcessRx** is a compile-time constant that can be tuned to the needs of the application. The output queue is also periodically checked by the **NetSend** task, with the same period as the
Fixed Priority scheduling policy has been adopted in the tasks implementation: four priority levels (high, high/medium, medium/low, low) have been assigned in order to guarantee a fixed activation pattern to tasks devoted to packet forwarding. The tasks activation order is shown in Figure 2.15. Please note that the priorities are enforced in decreasing order from the receiver to the sender. That is aimed to empty the receiving buffers at the MAC layer as soon as it is possible to make space for other packets thus minimizing the probability of losing packets due to buffers overflowing. Therefore, if while the previous packet is being processed a new packet arrives, it is immediately copied in the input queue by the high priority \texttt{MacRXData} task, thus making space for the next packet. At the same time, the control of the number of packets that can be processed in the time unit, therefore, the two network layer tasks are periodic and can only process one packet per period. Moreover \texttt{NetSend} and \texttt{NetProcessRx} frequency is directly associated with forwarding time overhead. By changing their period it is possible to tune this response, thus letting the node be less or more prompt in forwarding services.

2.5.3. Performance evaluation in a real scenario

This section reports the results of experiments conducted to evaluate the performance of \texttt{Speed-3D} in a realistic scenario. After presenting both experimental setup and adopted hardware, the performance of the protocol is presented by dividing them in two main parts: the first demonstrates the \texttt{Speed-3D} extended scope with respect to \texttt{Speed} through test cases; the second shows \texttt{Speed-3D} performance in terms of round-trip time and packet loss rate. Finally, the obtained results analysis is presented in order to infer analytic models that correlates the performance of the protocol to different values of the parameters. Performance results have been studied only through real experiments because of the choice of evaluating \texttt{Speed-3D} functionality in a real 6LoWPAN network with nodes running a real-time kernel.
**Test bed scenario and network topology**

The considered network scenario is composed by nine nodes deployed in an indoor environment over two floors of the “Institute of Communication, Information and Perception Technologies” (TeCIP), Pisa, Italy. The whole topology, used for all tests, is depicted in Figure 2.16.

![Figure 2.16. Network configuration.](image)

All nodes placed in the first floor have the altitude parameter equal to 1 m, whereas all nodes placed in the second floor have an altitude equal to 5 m. As shown in the picture, the nodes on the first floor are connected to those on the second floor with a single link between nodes 0x22 and 0x32, which have the same latitude and longitude. The network topology has been created ad-hoc by setting the nodes transmission power to the lowest level (-25 dBm). All the experiments have been performed out of working hours, in order to minimize interference from human activities. The physical IEEE 802.15.4 channel is set to 2405 MHz (channel 11) aiming at minimize interference with existing IEEE 802.11 networks. In each test the sender node transmits 2000 packets.

In all the experiments the adopted hardware is the Flex Board described in Section 2.4.3.

**SPEED-6LoWPAN versus SPEED-3D**

In order to compare SPEED-6LoWPAN and SPEED-3D, node 0x21 has been selected as source and the node 0x34 as destination. In SPEED the position of all nodes is expressed only with latitude and longitude, without altitude. Using SPEED-6LoWPAN as routing protocol, the messages sent by 0x21 can not reach the destination. When device 0x22 processes an incoming packet, the routing protocol does not select any next-hop. This happens because no neighbor is found in the Sender Feasible Candidate region. In Table 2.1a the 0x34 position is reported, while in Table 2.1d the 0x22 position is reported with its distance from node 0x34. Moreover, Table 2.1e reports the neighbor table for the node 0x22: the Next-Hop Distance (NHD) column is filled with the distance between 0x34 (final destination) and the considered neighbor (the node in the first column). The Distance Difference (DD), instead, is the distance progress in using
Table 2.1. Speed-6LoWPAN.

<table>
<thead>
<tr>
<th>Node</th>
<th>Lat.</th>
<th>Lon.</th>
<th>Alt.</th>
<th>NHD</th>
<th>DD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>43.71832</td>
<td>10.42456</td>
<td>1 m</td>
<td>14.95 m</td>
<td>-5.13 m</td>
</tr>
<tr>
<td>0x21</td>
<td>43.71829</td>
<td>10.42442</td>
<td>1 m</td>
<td>14.50 m</td>
<td>-4.68 m</td>
</tr>
<tr>
<td>0x32</td>
<td>43.71828</td>
<td>10.42450</td>
<td>5 m</td>
<td>8.97 m</td>
<td>0.85 m</td>
</tr>
</tbody>
</table>

RTT and PLR performance

Speed-3D has been rated against two notable metrics: round-trip time and packet loss rate. The first one is defined as the interval from message transmission (time stamped at the source node) and reception of the reply (transmitted by destination node and time stamped at the source as well); the second measurement is the ratio of packets transmitted and not followed by a reply. To evaluate such metrics, in each experiment the source node sends periodically a message, then waits for its acknowledgment from the destination node. Each message sent through the network is identified by a unique sequence number that is used for reply packets.

There are many factors affecting the packet delivery performance in a real-time 6LoWPAN network with multi-hop communication. The selected metrics (i.e., RTT and PLR) have been measured as function of number of hops, frequency of periodic tasks (NetProcessRX, NetSend and Application Send) and the traffic load, as discussed in the following.
Number of Hops As first study RTT behavior as a function of the number of hops is investigated. Experiments have been performed with one single sender and one single receiver. The source node is fixed, whereas the destination varies to increase the number of hops. The period of the network layer tasks has been set equal to 10 ms; the source sends 1 packet every 100 ms. RTT distributions for six, eight and ten hops are shown in Figure 2.17. As it can be expected, the average RTT value increases with the number of hops as a linear function (Figure 2.18a). Regarding the PLR, the experiments results, shown in Figure 2.18b confirm results obtained in [ZG03] in which PLR values less than 10% have been experienced in a similar scenario.

FIGURE 2.17. RTT Distribution for six, eight and ten hops.

FIGURE 2.18. RTT and PLR versus number of hops.

Network and Application Tasks The frequency of the periodic tasks affects the RTT and PLR, as described in Section 2.5.2. The presented implementation includes three periodic
First tests were performed fixing the period of NPT and NST equal to 10 ms and varying the period of AST. RTT and PLR have been evaluated with AST period equal to 30 ms, 40 ms, 50 ms, 100 ms and 200 ms. The communication path is eight hops long. As expected, the average RTT value increases when the AST period decreases, due to the greater number of packets in the transmission path which increases the packets waiting time in the NET queues. It is very interesting to note that in some intervals (for instance 100 to 200 ms and 40 to 50 ms) RTT does not depend on AST very strongly (see Figure 2.19a). That is due to:

- **AST period from 100 to 200 ms.** In this case the average RTT value is about 64 ms. This time is lower than the AST period, therefore a packet sent by source node is covering the full path without interference from other packet transmission;
- **AST period from 40 to 50 ms.** With this parameter setting, the average RTT value is about 69 ms, that is greater than the AST period. In this case a packet sent at time $t_i$ by source node interferes with the $t_{i-1}$ transmission throughout the path. This causes a delay and increases the average RTT;
- **AST period equal to 30 ms.** The average RTT value is 71.32 ms. Packet sent at time $t_i$ may encounter packets sent at time $t_{i-1}$ and $t_{i-2}$ throughout the path. The average RTT value is doubled with respect to measurement taken with AST period equal to 50 or 40 ms.

**Figure 2.19.** RTT and PLR as a function of AST.

RTT and PLR are also affected by the periods of the network layer tasks. The same set of measurement with period equal to 20 ms are performed to check the dependence of RTT and PLR on NPT and NST period. This produces only a global scaling factor in the RTT, as shown in Figure 2.19a. In Figure 2.19b, PLR is plotted against AST period for both 10 ms and 20 ms period of network layer tasks. Generally speaking, apart from channel noise fluctuations, PLR increases as AST period increases. This increment is caused by packet collisions over-air. Table 2.2 reports detailed results for experiments hereby discussed.
Traffic Load As last study, the dependence of SPEED-3D performance from the traffic load is analyzed. More in detail, the evaluation of RTT and PLR is presented in two cases:

- **two flows**: one main flow, from node 0x21 to node 0x34, and one interference flow, from node 0x31 to node 0x34;
- **three flows**: one main flow, from node 0x21 to node 0x34, and two interference flows, the first one from node 0x31 to node 0x34 and the second one from node 0x36 to node 0x34.

The presented RTT and PLR results are related to the main flow. For both cases the period of NPT and NST are fixed and equal to 10 ms, instead the AST period varies: 100 ms, 200 ms and 400 ms. All the flows had the same AST period. The results are summarized in Table 2.2.

<table>
<thead>
<tr>
<th>AST Frequency</th>
<th>RTT</th>
<th>PLR</th>
<th>RTT</th>
<th>PLR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Hz (200 ms)</td>
<td>63.59 ms</td>
<td>6.00%</td>
<td>78.13 ms</td>
<td>6.95%</td>
</tr>
<tr>
<td>10 Hz (100 ms)</td>
<td>64.67 ms</td>
<td>6.15%</td>
<td>79.47 ms</td>
<td>4.95%</td>
</tr>
<tr>
<td>20 Hz (50 ms)</td>
<td>68.91 ms</td>
<td>10.10%</td>
<td>83.61 ms</td>
<td>9.75%</td>
</tr>
<tr>
<td>25 Hz (40 ms)</td>
<td>68.62 ms</td>
<td>9.80%</td>
<td>84.23 ms</td>
<td>10.20%</td>
</tr>
<tr>
<td>33 Hz (30 ms)</td>
<td>71.33 ms</td>
<td>13.85%</td>
<td>89.85 ms</td>
<td>14.05%</td>
</tr>
</tbody>
</table>

Table 2.2. RTT and PLR performance: numerical results.

As expected, the average RTT value increases with the frequency of AST as a linear function (Figure 2.20a). Obviously the increment is greater in case of three flows with respect to two flows: the probability to encounter packets throughout the path is greater for the former case. In Figure 2.20b the PLR value is plotted. For AST period equal to 400 ms or 200 ms the PLR is similar for the cases of two and three flows. With AST period equal to 100 ms the PLR is greater for the case of three sources because of the higher number of collisions over-air.

Analytical study

Starting from the experiments presented above an analytic model for the RTT has been investigated. More in detail, a linear model has been built that is valid under “normal” traffic conditions, then the model has been compared against the results of the experiments.
As starting point RTT can be expressed as the sum of three components: \( RTT_{base} \), \( T_{int} \) and \( T_{flow} \) (Equation 1).

\[
RTT = RTT_{base} + T_{int} + T_{flow}
\]  

\( RTT_{base} \) represents the value of RTT in absence of any interference. A linear dependence between \( RTT_{base}/N_{hop} \) and the period of the network layer tasks (\( P_{net} \)) is assumed:

\[
RTT_{base}/N_{hop} = a + b \cdot P_{net}
\]

Constant \( a \) represents the Stack Execution Time (\( SET \)), i.e., the execution time of the protocol software stack in forwarding data packet. Constant \( b \) models the Queue Execution Time (\( QET \)), i.e., the time that a packet has to wait in the queue before it is processed. By using the Least Square Method on the measures obtained in the experiments, the best values for these constants are show in the following: parameter \( SET \) is equal to 6.3207 ms and parameter \( QET \) is equal to 0.1765 ms. By using the introduced parameters Equation 2 can be rewritten as:

\[
RTT_{base}/N_{hop} = (SET + QET \cdot P_{net}) \cdot N_{hop}
\]

\( T_{int} \) is the overhead time introduced in case of RTT value lower than the AST period, and \( P_{app} \) is the period of AST. A simple lower bound for \( RTT_{base}/P_{app} \) is the number of packets flowing throughout the path, while \( T \) is the additional waiting time experienced by a packet in the queue before it is processed. \( T \) is proportional to \( P_{net} \). This is translated into the following dependence:

\[
T_{int} = \left[ RTT_{base}/P_{app} \right] \cdot T
\]

\( T_{flow} \) is the overhead time introduced in case of multiple sources. \( N_{flow} \) is the number of flows along the communication path. This variable is obtained with an estimation of the number of packets throughout the path multiplied by \( T \). Assuming that \( T_{flow} \) depends on \( RTT_{base} \), \( N_{flow} \), \( P_{app} \) and \( k \) through the following relationship:
\[ T_{\text{flow}} = \left\lfloor k \cdot N_{\text{Flow}} \cdot \left( \frac{\text{RTT}_{\text{base}}}{P_{\text{app}}} \right) \right\rfloor \cdot T \]  

(5)

where \( k \) is a constant that, by using the Least Square Method with the presented experiments, can be set equal to 2.4.

In Table 2.4, the real RTT values measured in each experiment are reported in Column 5. Estimated RTT (Column 6) is the value calculated by Equation 1 and Error (Column 7) is the difference between the measured RTT value and the RTT value calculated by Equation 1. Finally, last column reports the error percentage. As it is possible to see, the maximum error from real world measurements is always lower than 2.23 ms corresponding to a percentage of 3.17%. Therefore, the presented analytic model closely captures the behavior of the system under analysis. The proposed model can be used as a way to estimate RTT in a wide range of not-overloaded scenarios.

<table>
<thead>
<tr>
<th>( P_{\text{net}} ) (ms)</th>
<th>( P_{\text{app}} ) (ms)</th>
<th>( N_{\text{hop}} )</th>
<th>( N_{\text{flow}} )</th>
<th>Real RTT (ms)</th>
<th>Estimated RTT (ms)</th>
<th>Error (ms)</th>
<th>Error / Real RTT(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>100</td>
<td>6</td>
<td>0</td>
<td>48.21</td>
<td>48.51</td>
<td>-0.30</td>
<td>0.62</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>8</td>
<td>0</td>
<td>64.68</td>
<td>64.69</td>
<td>-0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>10</td>
<td>0</td>
<td>81.35</td>
<td>80.86</td>
<td>0.49</td>
<td>0.60</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>8</td>
<td>0</td>
<td>71.33</td>
<td>71.49</td>
<td>-0.16</td>
<td>0.22</td>
</tr>
<tr>
<td>10</td>
<td>40</td>
<td>8</td>
<td>0</td>
<td>68.62</td>
<td>68.09</td>
<td>0.54</td>
<td>0.78</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>8</td>
<td>0</td>
<td>68.91</td>
<td>68.09</td>
<td>0.83</td>
<td>1.20</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>8</td>
<td>0</td>
<td>64.68</td>
<td>64.69</td>
<td>-0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>8</td>
<td>0</td>
<td>63.60</td>
<td>64.69</td>
<td>1.09</td>
<td>1.71</td>
</tr>
<tr>
<td>20</td>
<td>30</td>
<td>8</td>
<td>0</td>
<td>89.85</td>
<td>92.41</td>
<td>-1.91</td>
<td>2.13</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>8</td>
<td>0</td>
<td>84.23</td>
<td>85.61</td>
<td>-1.38</td>
<td>1.64</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>8</td>
<td>0</td>
<td>83.61</td>
<td>85.61</td>
<td>-1.99</td>
<td>2.38</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>8</td>
<td>0</td>
<td>79.48</td>
<td>78.81</td>
<td>0.67</td>
<td>0.85</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>8</td>
<td>0</td>
<td>78.13</td>
<td>78.81</td>
<td>-0.68</td>
<td>0.87</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>8</td>
<td>1</td>
<td>68.23</td>
<td>68.09</td>
<td>0.15</td>
<td>0.21</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>8</td>
<td>1</td>
<td>66.79</td>
<td>64.69</td>
<td>2.11</td>
<td>3.16</td>
</tr>
<tr>
<td>10</td>
<td>400</td>
<td>8</td>
<td>1</td>
<td>64.89</td>
<td>64.69</td>
<td>0.2</td>
<td>0.31</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>8</td>
<td>2</td>
<td>74.09</td>
<td>74.89</td>
<td>-0.8</td>
<td>1.08</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>8</td>
<td>2</td>
<td>70.32</td>
<td>68.09</td>
<td>2.23</td>
<td>3.17</td>
</tr>
<tr>
<td>10</td>
<td>400</td>
<td>8</td>
<td>2</td>
<td>65.52</td>
<td>64.69</td>
<td>0.83</td>
<td>1.27</td>
</tr>
</tbody>
</table>

Table 2.4. Analytic model validation.

### 2.6. Final remarks

Speed adaptation in 6LoWPAN networks has been realized by considering possible optimization provided by the use of IPv6 related protocols. Moreover further analysis must be done
The control overhead required by Speed-3D is minimal, and mainly due to additional bytes to be introduced in messages already defined by 6LoWPAN protocol. Indeed, no additional periodic control packets have been added because of the use of ICMPv6 RA messages to encapsulate the protocol Beacon message. In this respect, the only introduced overhead is due to the additional 14 bytes added as extension of the RA message. The real control overhead introduced by the routing protocol is due to the Backpressure beacon message, which has a global size of 27 bytes, and is sent only in case of congestion or void avoidance events. Such control message is not periodic, and for networks with regular topologies (no strong void avoidance required) and light/medium traffic transmission (no heavy congestion control) its total overhead is negligible respect to 6LoWPAN control messages. Due to the locally-based selection of the next hop adopted by the Speed protocol, part of the control overhead can be attributed to the Data packet message. Such overhead consists of additional 20 bytes to be considered in each 6LoWPAN data message, again a negligible impact in regular WSN applications.

The energy consumed by each routing protocol is strictly related to the introduced overhead. Because geographical routing requires low resource consumption and communication overhead with respect to other routing techniques [SMZ07], geographic routing protocols are efficient in WSNS. Indeed, as discussed in [SMZ07], geographic routing protocols require that only the direct neighbors information are maintained by each node, thus conserving energy and bandwidth since discovery floods and current network state propagation are not required. Although in a WSN scenario geographic routing protocols can be considered highly efficient with respect to other routing techniques, different geographical forwarding strategies have a different impact on energy consumption and delivery rate [SZHK04]. As a consequence, Speed-3D can be considered high-energy efficient, even if possible improvements can be reached by using different forwarding techniques.

The main issue of a routing protocol in large-scale networks is mainly related to scalability. In such a scenario, by considering geographic routing protocols, the reduced control overhead enhances the scalability [SMZ07] under the assumptions reported in [SZHK04] for greedy forwarding mechanisms: (i) sufficient network density, (ii) accurate localization, and (iii) high link reliability. When these assumption hold, Speed-3D routing protocol is expected to provide good scalability properties.

2.7. Conclusion

This chapter discussed the author’s contributions related to the protocol adaptation towards the full accomplishment of the IoT vision: the adaptation of Speed routing protocol target to 6LoWPAN networks.

Starting from the packets format of IEEE 802.15.4 and IoT standards, mechanisms and encapsulation policy to adapt Speed routing protocol in 6LoWPAN networks are proposed. The Speed-6LoWPAN has been implemented in the network stack of the Erika OS and its performance has been compared with respect to the implemented 6LoWPAN version of AODV.
In a 6LoWPAN context Speed shows a negligible bigger round trip time with respect to AODV while saving time in route creation and routing table memory occupancy. Moreover Speed-6LoWPAN supports soft real-time, load balancing and flow shaping mechanisms making itself an effective solution in supporting packet routing in 6LoWPAN networks.

Speed-3D is an extension of Speed-6LoWPAN routing protocol that caters for 3D routing. Starting from Speed-6LoWPAN design the third dimension for locate a node as well as modular task design are introduced. Speed-3D performance evaluation is proposed by performing experiments in a realistic indoor scenario. To this end an implementation of Speed-3D in a real-time OS is detailed by introducing a modular design policy for tasks devoted to forwarding services. From the performed experiments Speed-3D effectively allows to create communication path among all nodes in a 3D network, which is a characteristic not provided by the original Speed-6LoWPAN routing protocol. Moreover, to evaluate in-network Speed-3D performance several experiments have been performed to extract both Round-Trip Time and Packet Lost Rate metrics. Such metrics are presented as a function of: number of hops, tasks frequencies, and traffic load. Starting from obtained results an analytic model to estimate the RTT value is presented and discussed. The proposed model closely captures the behavior of a real system, thus serving as a starting point to estimate RTT in a wide range of not-overloaded geographic 6LoWPAN networks.
CHAPTER 3

Combine IoT protocols with new technologies

The Internet of Things vision has been recently enabled by the worldwide spreading of Internet in combination with the development of new miniaturized and low-cost embedded devices with communication capabilities. The main idea behind the IoT concept is to have worldwide interconnected objects, each one individually discovered and addressed as a resource in the network. IoT devices will be remotely accessible, thus making available an enormous amount of data about the physical world. Aside from network communication point of view, recent progresses in hardware solutions targeted to WSNs have also permitted to integrate on tiny motes powerful sensors. These progresses make available a different varieties of data, such as images or additional information coming from Radio-Frequency IDentification, thus leading the research community to develop new WSN-based applications in new scenarios. Wireless Camera Network is one example of the output of this research effort.

WCNs based on low-end devices have experienced a rapid growth in the last several years due to the hardware miniaturization process fostered by the electronic industry, and the simultaneous advances in computer vision, embedded computing and sensor networks disciplines. Research efforts have been oriented in the last years at replacing high-cost and high-performance camera network devices [VV05] with low-cost, low-complexity embedded devices organized in distributed systems [AMC07]. The possibility of integrating multimedia and computer vision based capabilities on low-end devices permits to create innovative and pervasive WCNs [RW09] able to perform high-valued applications such as distributed visual surveillance [RSY11], object tracking [ALB07] and traffic monitoring [SPGP12a]. The integration between WCN and IoT vision would be the next step in order to increase the WCN power, specially in respect of the pervasiveness. Considering video-streaming services targeted to video surveillance applications, a pervasive network would allow to gather complementary views of a given scene at a reduced installation cost, thus significantly increasing the available information content. However, low-complexity camera network devices are strongly constrained in terms of battery, memory, processing capability, and achievable data rate so that each video-streaming protocol is expected to be fully customized from state-of-the-art solutions. Indeed, because of these severe limitations, the highest frame rate reachable by a smart camera node is the result of a strong trade-off between the frame rate achievable by a given compression algorithm, derived from the adopted hardware architecture, and the available transmission bandwidth.

Another example of output in WSN hardware advances is the integration between Radio-Frequency IDentification and WSN technologies. Albeit the IoT vision has been initially inspired by the success of the RFID technology [KKFS10], such a vision can be effectively enabled...
through a seamless integration of both RFID and WSNs solutions in the Internet scenario. RFID is an extremely low-cost and low-power technology mainly characterized by passive devices, i.e., tags, which are able to send information when powered by electromagnetic fields generated by an RFID-Reader. It is a short-range radio technology mainly used for object identification [BT06] and tracking [WPA07]. WSNs are composed by low-power embedded devices characterized by reduced computational capabilities that actively communicate among them to fulfill complex tasks. The transmission range of WSN devices is in the order of a hundred meters, and they are mainly used for real time environmental monitoring [LWS04], tracking [ZLZ+08], and localization purposes [WyZwKX11]. The advantages provided by both technologies promote the design of an integrated solution in which the outstanding pervasiveness of RFIDs and the advanced sensing and communication features of WSNs are merged together to have pervasive and addressable resources in a worldwide network of objects.

Contributions The rest of the chapter discusses the author’s contributions related to combine Internet of Things protocols with new technologies, such as video streaming and RFID. After an introduction of the 6LoWPAN architecture (Section 3.1), in Section 3.2 a low-complexity algorithm aimed at reducing the transmission bandwidth of a video stream of uncompressed images, thus permitting a higher frame rate in 6LoWPAN networks, is presented [SPM+13]. Section 3.3 proposes and discusses the integration of WSN and RFID technologies in the IoT scenario [PBA+13].

3.1. Internet of Things architecture in Wireless Sensor Networks

Over the past few years the research community has focused its activity on designing protocols for the IoT [MPV11]. Indeed, well-known and widely used Internet protocols are often unsuited for IoT devices that are usually constrained in terms of computational power, memory, and transmission bandwidth. The main outcome of this research effort is an adaptation layer for the IPv6 protocol over Low-power Wireless Personal Area Networks (6LoWPAN) compliant with the IEEE 802.15.4 [Soc11] standard. 6LoWPAN [KMS07], has proven to be a valid alternative to traditional proprietary WSN protocols [HC08]. Indeed, 6LoWPAN based networks can compete with traditional WSNs in terms of power consumption and network throughput, while achieving a seamless integration and interoperability with Internet.

The possibility of interconnect every objects in combination with the development of new miniaturized and low-cost embedded devices equipped with a variety of sensors (e.g., camera, microphone, RFID-Reader, RFID-TAG) led to integrate new interesting technologies in the WSN domain. Such a kind of technologies, on the one hand, open new interesting application scenarios, increasing the services already provided by the WSNs, and on the other hand they raise new challenges which have to be tackled in order to successfully benefit of such integration. In addition to the reduced computational capability, bandwidth and memory constrain, the mapping of each new components with the 6LoWPAN architecture must be taken into account. A 6LoWPAN network, whose architecture is defined in [KKGB12] and pictorially sketched in in Figure 3.1 is mainly composed by three type of nodes: the “simple” Host not implementing forwarding and routing services, the 6LoWPAN Router (6LR) having forwarding and routing capabilities, and
the 6LoWPAN Border Router (6LBR) connecting each subnet to the Internet by translating 6LoWPAN into IPv6 packets and vice versa. All the above introduced 6LoWPAN components will be considered in the following by detailing the role of any introduced system component.

### 3.2. Video streaming applications in IoT scenario

Considering WCN devices compliant with the IEEE 802.15.4 standard the amount of available bandwidth at Physical layer (PHY) is equal to 250 Kbit/s with even lower values at the MAC layer (i.e., less than 200 Kbit/s in case of both unslotted based transmissions [LMM+06] and slotted based transmissions [PKC+05]). To face the strong network bandwidth constraints new possible network architecture for WCNs must be defined. A reference architecture for Wireless Camera Network based on low-cost embedded devices has been preliminary investigated in [MA11], where along with multimedia enabled sensor devices, Camera Node (CN), the concept of Multimedia Processing Hub (MPH) is introduced. A MPH is a node of the network with higher computational capabilities with respect to simple multimedia nodes, and able to aggregate video streams. In a hierarchic view of the whole network of CNs sends video data to the MPH that works as a multimedia sink sending aggregated data towards external network gateways.

Albeit the above proposed WCN architecture is a first step towards an effective development of pervasive camera network systems, it does not take into account advantages and drawbacks coming from new protocol solutions aiming at realizing a full interoperability with the Internet world. A possible mapping of WCN composed by CNs in the 6LoWPAN architecture (Figure 3.1) is depicted in Figure 3.2: a CN is a “simple” 6LoWPAN Host that sends images gathered from the surrounding environment to Internet world without any forwarding and routing services. The adoption of 6LoWPAN in a WCN permits the full interoperability with the Internet at the cost

![Figure 3.1. 6LoWPAN architecture.](image-url)
of a further reduction in the available communication bandwidth (i.e., less than 100 Kbit/s in the average), thus requiring high performance data compression scheme with a possible joint use of low-complexity traffic shaping algorithms to overcome data bursts in the network.

![Figure 3.2. Wireless camera network architecture based on 6LoWPAN communications.](image)

This section presents an innovative video compression approach to be used in video surveillance applications and implementing: (i) in-node video compression based on low-complexity computer vision techniques, and (ii) aiming at reducing the transmission bandwidth by removing inter-frame redundancy. The presented solution is an extension of the work presented in [SPP+12]. More in particular, hereby an extension of the “basic” codec with an error resilient streaming technique, applicable to a 6LoWPAN network with the evident benefit of improving Quality of Service (QoS). All performance results have been obtained by means of a simulative approach, and to be as realistic as possible (i) a “C” library has been developed to emulate the compression and decompression algorithms on a full fledged device, while considering microcontroller based constraints (e.g., no dynamic memory, no floating point operations); (ii) an image dataset has been imported from the IPERMOB project [1] and characterized by real images acquired by sensor devices equipped with a low-cost camera [SEE]; (iii) the packet format is compliant with the 6LoWPAN specifications defined in IETF RFC 6282 [DHT11]; (iv) packet corruption is taken into account by using real network loss traces taken from the IPERMOB data acquisition campaigns. The approach presented is a possible candidate for streaming applications in WSNs based on 6LoWPAN in which video streams originated by multimedia sensors can be consumed by high-end devices in the spirit of the Internet of Things.

3.2. VIDEO STREAMING APPLICATIONS IN IOT SCENARIO

3.2.1. Background

The idea behind WCNs (a.k.a. Smart Camera Networks, or SCN) is the creation of a network of nodes able to locally process the visual information with the aim of extracting a set of features to be sent through the network, thus avoiding the full transmission of raw images. Every CN [RW08] combines visual sensing, image processing, and network communication in a single embedded platform, permitting to transform traditional cameras into smart sensors. In simple terms, the main components of a WCNs are the embedded CNs and the computer vision algorithms performing the feature extraction.

In recent years several research initiatives have produced real prototypes of camera node sensors able to perform image processing as well as to communicate wireless through the IEEE 802.15.4 [Soc11] standard, the first main requirement to enable IoT-based communications. All the solutions presented in the following embed IEEE 802.15.4 compliant transceivers, thus they are presented by detailing only vision capabilities and main application scenarios.

Among the presented devices the first to be cited is the WiCa [AK02] platform developed by NXP Semiconductors Research. Such a device is equipped with a NXP Xetal IC3D processor based on an SIMD architecture with 320 processing elements and can host up to two CMOS cameras at VGA resolution (640 x 480). It has been mainly used for image-based local processing and collaborative reasoning applications. The Mesheye [HPFA07] project is another research initiative aimed at developing an energy-efficient visual sensor based on an ARM7 processor, and able to run intelligent surveillance applications. Regarding the vision capabilities, the MeshEye mote has an interesting special vision system based on a stereo configuration of two low-resolution, low-power cameras, coupled with a high-resolution VGA color camera. In particular, the stereo vision system continuously determines position, range, and size of moving objects entering its fields of view. This information triggers the color camera to acquire the high-resolution image sub-window containing the object of interest, which can then be efficiently processed. Another interesting example of low-cost embedded vision system is represented by the CITRIC [CAB+08] platform. Such a device integrates a 1.3-megapixel camera sensor, a XScale PXA270 CPU (with frequency scalable up to 624 MHz), a 16 MB FLASH memory, and a 64 MB RAM. CITRIC capabilities have been illustrated by three sample applications: image compression, object tracking by means of background subtraction, and self-localization of the camera nodes in the network. More recent device projects are the Vision Mesh [ZC10] and the Seed-Eye [SEE] boards. The Vision Mesh integrates an Atmel 9261 ARM9 CPU, 128 MB NandFlash, 64MB SDRAM, and a CMOS camera at VGA resolution. The high computational capabilities of the mounted CPU permit to perform advanced computer vision technique targeted to water conservancy engineering applications. The Seed-Eye board has been specifically designed for image-based applications in sensor networks. Indeed, its design is very similar to tiny sensor network devices while adding a powerful microcontroller, the PIC32MX795F512L manufactured by Microchip, and a low-cost CMOS camera able to reach up to 1.3-megapixel of resolution. The board has been used for ITS applications by performing an on-board image processing aiming at counting the number of passing vehicles and monitoring the occupancy level of parking spaces. In all the aforementioned camera node devices the image processing tasks
are performed through software applications. A new possible hardware-based solution aiming at speed up processing capabilities in WCN has been recently proposed in [MSP13]. In such a prototype solution an FPGA-based CN is designed with a reconfigurable architecture able to connect optimized hardware processing blocks with the final goal of performing advanced vision tasks. Then, a SoftCore is in charge of the system management by controlling the internal data flow configurations and processing block parameters. Communication capabilities can be demanded to an external microcontroller or to another SoftCore embedded in the FPGA. All the presented visual sensors are depicted in Figure 3.3.

![Visual Sensors](image)

**Figure 3.3.** From top left to bottom right: WiCa, Mesheye, CITRIC, Vision Mesh, Seed-Eye and FPGA-based CN.

### 3.2.2. Related works

In case of video streaming applications over 6LoWPAN based WCNs, the related constraint imposes source coding techniques with a strong trade-off between compression efficiency and complexity. Traditional video compression techniques rely on both intra-frame compression techniques to remove redundancy in standalone frames, and inter-frame compression in which temporal redundancy is further removed. Such approaches are widely used by state-of-the-art video encoders, such as MPEG, H.263, and H.264, that require powerful processing algorithms and high levels of energy consumption, thus making their use completely unfeasible in low-cost wireless camera devices. State-of-the-art video compression techniques in WSNs based on low-cost embedded devices are mainly focused on in-node intra-frame algorithms, such as standard JPEG or derived optimized versions. In [MRX08] a survey of JPEG-based video compression techniques targeted to low-end devices is presented. Along with the benefits of possible JPEG
fixed point implementations \textsuperscript{CM02}, and modified versions with change detection approaches specifically developed for JPEG \textsuperscript{MMM03}, in the survey the main disadvantages in using this standard are pointed out. The big issue in using JPEG in low-end WCNs is the lack of error resilience properties \textsuperscript{ZEP+04} that can be overcome by Forward Error Correction \textsuperscript{LL81} techniques, Erasure Correction codes \textsuperscript{MS04}, Interleaving scheme \textsuperscript{DFL08} and Variable-Length Coding (VLC) \textsuperscript{DFLL11}, thus increasing the required transmission bandwidth \textsuperscript{PGS+11} and lowering the highest reachable frame rate. The same issues can be experienced in other JPEG-based compression approaches \textsuperscript{HBY08}, as well as solutions in which a coding scheme is performed by using frequency transformation functions, such as in \textsuperscript{PBR+11} where a new hybrid DPCM/DCT coding scheme is proposed to achieve an acceptable compression gain with a low-computational complexity, and \textsuperscript{WPW+08} where a wavelet based coding technique is proposed and its performance jointly evaluated with Unequal Error Protection (UEP) \textsuperscript{MHPS13} techniques. Although all the above mentioned techniques focus on in-node low-complexity algorithms, only few works have proposed at design stage the use of distributed techniques \textsuperscript{MA11}, while specific compression scheme targeted to 6LoWPAN WCNs have not been deployed yet.

### 3.2.3. Wireless Camera Networks video compression and transmission

State-of-the-art video-surveillance applications are based on a streaming service between a static camera and a remote control point; the service is required to fulfill the temporal constraints usually defined in terms of frame rate. The minimum frame rate required by such applications is about 1 fps. The diagram shown in Figure 3.4 describes the main functional blocks, and their relations, in a standard video-surveillance application based on a streaming service. In this situation, on the transmitter side, the images acquired by a camera network devices are compressed using a \textit{video-encoder} and fragmented in packets to be transmitted through the wireless network. At the receiver side, the received packets are aggregated in order to decompress the images (using a \textit{video-decoder}) to be shown on the user screen.

![Figure 3.4. Video-surveillance functional block diagram.](image)

As it is discussed in \textsuperscript{PGS+11}, a single stream, at a frame rate of 1 fps and QQ-VGA resolution (i.e., 160×120 pixels), fully saturates the available bandwidth in IEEE 802.15.4 networks. In the same conditions, considering the 6LoWPAN overhead, a single stream would not fit the temporal constraints as imposed by the minimum required frame rate.
As previously introduced, JPEG and Lossless JPEG are state-of-the-art solutions for compressing streams in WCNs. Although the above mentioned compressors offer optimal performance in terms of compression ratio, they are characterized by high computational complexity ($O(N \log N)$) and lack of error resilience [MRX08]. The latter descends from the strong correlation that the compressed blocks have with each other. Consequently the scientific objective addressed deals with the realization of (i) a low-complexity video codec and (ii) a data protection schema showing a stronger error resilience than state-of-the-art solutions. As a further improvement, three types of concealment are applied and compared against an improved version of JPEG.

### The video codec

The considered compression algorithm is based on the classification of each pixel of an image in terms of its information content. More specifically it is possible to label as **background** the static component of a monitored scene, and as **foreground** its dynamic component. Thus, a compression algorithm based on background subtraction is eligible for low-rate networks, as WSNs, since it allows to transmit only the relevant parts of the considered raw image, also called Regions Of Interest (ROI). When a background subtraction approach is adopted, a background updating rule is needed to make robust the compression algorithm to luminosity variations and once-off changes. Deterministically the following logic has been applied: at a certain rate a raw frame is labeled as local background image and is entirely transmitted to the control point to keep it synchronized and updated. The rate is called Background Refresh Period (BRP) and is measured in number of frames.

From a high level point of view, the **video-encoder** is required to detect, for each frame, the occurred changes with respect to a background image, discarding the parts of the frame without information content (inter-frame compression). The **video-decoder**, instead, is in charge of reconstructing the frames starting from the received parts of the image (ROIs) and the last received background image. Along with the raw pixels content, a header descriptor is associated to each ROI introducing a certain overhead in the proposed approach. For instance, in order to properly position a rectangular ROI, the descriptor should contain the image identifier (i.e., 1 byte), x and y coordinates of the left corner (i.e., 2 bytes each), width and height dimensions (e.g., 2 bytes each). In this example the overhead amounts to 9 bytes per ROI.

The diagram in Figure 3.5 shows the main functional blocks of the video-encoder. Accordingly, in the **blob segmentation phase**, an operation of pixel-wise subtraction of the current frame with respect to the background one is performed: the output of this operation is called **difference image**. The background component inside the difference image are characterized by pixel values close to zero (darkest pixels), while the foreground shows bigger pixel values (clearest pixels). To distinguish background regions from foreground, the difference image is thresholded and denoised using morphological operators (i.e., opening) in order to generate a **binarized image**. In this latter image the background component is represented using black pixels, and the foreground with white. Then, in the **blob extraction phase**, the foreground pixels of the **binarized image** are grouped in order to understand the presence of relevant objects (in terms of occupied area) and the obtained ROIs are represented using rectangular bounding-boxes. Finally, the ROIs are
extracted from the difference image and sent through the network. Background image update follows the periodicity discussed above, in order to decorrelate the protocol to the light variation and once-off changes. The whole encoder algorithm is reported as pseudo-code in Figure 3.6.

```plaintext
Frame_id = 0;

while(1) {
    get_image(image);

    if((Frame_id % BRP) == 0) {
        send_image(image);
        bgnd = set_background(image);
    } else {
        diff = blob_segmentation(image, bgnd);
        roi = roi_extraction(diff);
        send_roi(roi, header);
    }

    Frame_id++;
}
```

Figure 3.6. Encoder.

On the other side, in the decoding phase, the image reconstruction process is performed by inserting the received ROIs of the i-th image frame on top of the last received background (by performing a pixel-wise sum operation), as it is shown in Figure 3.7. The decoder algorithm is reported as pseudo-code in Figure 3.8. The decoding phase has been kept as low-complexity as possible in order to permits to reconstruct more video flows in the remote control point.
Figure 3.7. The video-decoder.

```c
Frame_id = 0;
while(1) {
    receive_data(buffer);
    if((Frame_id % BRP) == 0) {
        bgnd = set_background(buffer);
    } else {
        image = reconstruct_image(bgnd, buff);
    }
    Frame_id ++;
}
```

Figure 3.8. Decoder.

**Dataset**

The performance of the proposed coding technique has been assessed by means of a simulation study aiming at evaluating the data rate achievable after the encoding process, as well as the quality of the video stream after a coding and decoding procedure. In all the performed simulations the IPERMOB Data Set (IPERDS) created within the IPERMOB project has been used. IPERDS is basically a collection of images related to traffic and parking lots conditions. The dataset is accessible to the general public via web with no privacy concerns since the low-resolution nature of the images cannot permit to recognize the vehicle plates.

Video traces last for more than 5 minutes at 1 fps frame rate. The dataset is composed by gray scale images having a QQ-VGA and Q-VGA resolution. All the images in the dataset have been collected by using real wireless sensor network devices equipped with a low-cost camera [SEE].

3.2. VIDEO STREAMING APPLICATIONS IN IOT SCENARIO

hence they have all the necessary characteristics to prototype video streaming and computer vision algorithms targeted to low-end camera network devices. The traces are cataloged in two different classes: low and high motion. More in detail, the video traces characterized by a low motion refer to the monitoring of several parking lots in which car parking events are recorded, while the high motion video traces contain the view of both city road and pedestrian walk. To deeply analyze the performance of the proposed coding technique both low and high motion Iperds traces have been considered. In all the performed simulations only QQ-VGA images have been used.

Codec performance assessment and comparison against JPEG

As described in the previous paragraph, the codec performance has been evaluated against the metrics reported in Table 3.1 by considering four IPERDS traces at a QQ-VGA resolution and characterized by low and high motion. Table 3.2 shows the overall performance in terms of data rate, compression ratio and PSNR, as a function of the background refresh period expressed in number of frames, a graphical representation is reported in Figure 3.9. The first row of the table reports the results when all raw images are sent through the network (no compression applied), thus showing the maximum video data rate and quality ($PSNR = \infty$).

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Squared Error (MSE)</td>
<td>$MSE = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (A(i,j) - B(i,j))^2$</td>
</tr>
<tr>
<td>Peak Signal-to-Noise Ratio (PSNR)</td>
<td>$PSNR = 10 \cdot \log_{10} \left( \frac{2^8 - 1}{MSE} \right)$</td>
</tr>
<tr>
<td>Compression ratio ($\rho$)</td>
<td>$\rho = \frac{\text{compressed_size}}{\text{uncompressed_size}}$</td>
</tr>
</tbody>
</table>

Considering the average results between the high and low motion scenario, reported in the right side of Table 3.2 it follows that the best trade-off between perceived video quality ($PSNR = 35.23\, dB$) and compression ratio ($\rho = 20.40\%$) is obtained when $BRP = 10$ frames is set (minimum considered value). In a preliminary analysis it could result intuitive to have an infinity PSNR for each BRP value due to the codec approach in “cutting and pasting” raw ROIs. Albeit this result is completely true in ideal conditions, the external environment, as well as the hardware limits, introduce luminosity variations which results in image differences and PSNR variations. Considering again results in Table 3.2 When $BRP \leq 50$ frames, $\rho$ and PSNR both shows a negative derivative. These trends simply demonstrate that a reduction in the compression ratio is paid at the cost of a lower quality in video reconstruction: indeed the temporal light variations (due to weather changes or internal camera reconfiguration) as for what
Table 3.2. Performance results of the proposed codec.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>153.60</td>
<td>100.00</td>
<td>∞</td>
<td>153.60</td>
<td>100.00</td>
<td>∞</td>
<td>153.60</td>
<td>100.00</td>
<td>∞</td>
</tr>
<tr>
<td>10</td>
<td>20.06</td>
<td>13.06</td>
<td>37.34</td>
<td>42.62</td>
<td>27.75</td>
<td>33.11</td>
<td>31.34</td>
<td>20.40</td>
<td>35.23</td>
</tr>
<tr>
<td>20</td>
<td>12.97</td>
<td>8.45</td>
<td>36.90</td>
<td>40.89</td>
<td>26.62</td>
<td>32.08</td>
<td>26.93</td>
<td>17.53</td>
<td>34.49</td>
</tr>
<tr>
<td>30</td>
<td>10.67</td>
<td>6.95</td>
<td>36.74</td>
<td>42.07</td>
<td>27.39</td>
<td>31.75</td>
<td>26.37</td>
<td>17.17</td>
<td>34.25</td>
</tr>
<tr>
<td>40</td>
<td>10.05</td>
<td>6.54</td>
<td>36.20</td>
<td>40.54</td>
<td>26.39</td>
<td>31.63</td>
<td>25.30</td>
<td>16.47</td>
<td>33.92</td>
</tr>
<tr>
<td>50</td>
<td>10.02</td>
<td>6.52</td>
<td>36.05</td>
<td>40.34</td>
<td>26.25</td>
<td>31.48</td>
<td>25.18</td>
<td>16.39</td>
<td>33.77</td>
</tr>
<tr>
<td>60</td>
<td>9.04</td>
<td>5.89</td>
<td>35.77</td>
<td>42.37</td>
<td>27.58</td>
<td>31.20</td>
<td>25.70</td>
<td>16.73</td>
<td>33.49</td>
</tr>
<tr>
<td>70</td>
<td>8.66</td>
<td>5.64</td>
<td>35.28</td>
<td>43.44</td>
<td>28.27</td>
<td>31.13</td>
<td>26.05</td>
<td>16.96</td>
<td>33.21</td>
</tr>
<tr>
<td>80</td>
<td>9.49</td>
<td>6.18</td>
<td>35.26</td>
<td>43.96</td>
<td>26.66</td>
<td>31.60</td>
<td>26.73</td>
<td>17.40</td>
<td>33.43</td>
</tr>
<tr>
<td>90</td>
<td>10.23</td>
<td>6.66</td>
<td>35.15</td>
<td>44.79</td>
<td>29.16</td>
<td>31.55</td>
<td>27.51</td>
<td>17.91</td>
<td>33.35</td>
</tr>
<tr>
<td>100</td>
<td>11.28</td>
<td>7.34</td>
<td>34.80</td>
<td>44.62</td>
<td>29.05</td>
<td>30.38</td>
<td>27.95</td>
<td>18.20</td>
<td>33.09</td>
</tr>
</tbody>
</table>

Figure 3.9. Proposed codec: ρ and PSNR as a function of the BRP.

Concerns autoexposure) degrade the quality of the reconstructed images because they invalidate
the background image. Conversely, when $\text{BRP} > 50$ frames, ρ slightly increases and PSNR
remains stable around 33 dB. These trends descend from the deterministic nature of the back-
ground image update (occurring at every $N : N/\%\text{BRP} = 0$) that generates the so-called ghosts
phenomenon. Indeed, when the deterministically acquired background image contains a moving
object $G_N$ (see Figure 3.10a), this is expected to leave the view after a certain number of frames
(see Figure 3.10c) causing a false convergence (i.e., an artifact) of the blob detection algorithm
into the region where the moving object was wrongly appointed to be part of the background
image (see Figure 3.10c). Ghosts are marginally affecting the considered performance when
3.2. VIDEO STREAMING APPLICATIONS IN IOT SCENARIO

\[ BRP \leq 50 \] frames because of the more frequent background image update, while they largely downgrade the observed metrics when the dynamics of the moving objects start to be compatible with the refreshing period, as in the case of \[ BRP > 50 \] frames.

![Background figure.](image1)

![i-th image.](image2)

![Binarized i-th image.](image3)

**Figure 3.10.** The ghost phenomenon.

The performance of the proposed codec has been compared with those of the standard JPEG. Concerning computation complexity, the developed solution is lighter than JPEG scaling as \( O(N) \) instead of \( O(N \log N) \), thus resulting more suitable for a real implementation in low-end camera network devices. On the other hand other metrics are in favor of JPEG by considering only a video stream coming from a compression and decompression operation. The results for the JPEG, the compression ratio, data rate, and PSNR are reported in Table 3.3 by comparing results achieved by the proposed technique.

**Table 3.3.** Performance results of the proposed codec versus JPEG.

<table>
<thead>
<tr>
<th>Proposed codec</th>
<th>JPEG</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRP</td>
<td>Data rate</td>
</tr>
<tr>
<td>NC</td>
<td>153.60</td>
</tr>
<tr>
<td>10</td>
<td>31.34</td>
</tr>
<tr>
<td>20</td>
<td>26.93</td>
</tr>
<tr>
<td>40</td>
<td>25.30</td>
</tr>
<tr>
<td>50</td>
<td>25.18</td>
</tr>
<tr>
<td>60</td>
<td>25.70</td>
</tr>
<tr>
<td>70</td>
<td>26.05</td>
</tr>
<tr>
<td>80</td>
<td>26.73</td>
</tr>
<tr>
<td>90</td>
<td>27.51</td>
</tr>
<tr>
<td>100</td>
<td>27.95</td>
</tr>
</tbody>
</table>
Moreover, overall JPEG results are reported in a graphical form in Figure 3.11 by showing the $\rho$ and PSNR behaviors as a function of BRP and JPEG quality. The Quality parameter is appointed as counterpart of BRP in JPEG since the latter is undefined. As it can be seen, in the ideal case of noise-free transmission, JPEG outperforms the proposed compressor. In fact fair quality in the perceived stream (e.g., $PSNR \simeq 35$) corresponds to a compression ratio double in JPEG than in our proposal. If we set the compression ratio (i.e., $\rho \simeq 20$), the quality of reconstruction obtained by JPEG compressor is higher than in our case (35.23 dB for our compressor, 39.04 dB for JPEG).

![Graphs showing $\rho$ vs Quality and PSNR vs Quality](image)

**Figure 3.11.** JPEG: $\rho$ and PSNR as a function of the Quality.

**Error resilience and data concealment techniques**

In this section, different techniques aiming at improving the error resilience of the proposed codec over error-prone networks are described. These techniques derive directly from the intrinsic uncorrelation of raw image data (the numerical representation of the luminance of each pixel is independent from the others). In this situation, if the decoder can reconstruct the image (applying error concealment techniques), the sporadic bit flips of certain pixels do not disrupt the general consistency in the reconstructed frame. The residual errors which cannot be removed (or concealed) rely on the metadata (i.e., descriptor fields) inserted as headers at the application and communication layers. In the following is described the proposed resilience schema for the intra-frame codec and an improved version of JPEG aiming at increasing its robustness in wireless communications.

In the proposed codec a ROI must be transmitted through the network while adding additional information (e.g., x and y coordinates, width and height dimensions). Nonetheless because of the tiny size of the payload in 6LoWPAN data packets (equal to 96 bytes), a single ROI (or background image) is fragmented into many 6LoWPAN packets. Following the arguments introduced above, any error occurring in the application header (encoded in the first packet) will critically disrupt the image reconstruction or the background image update at the receiver side. It is possible to better encode the ROI (or background image) at the network level coupling a descriptor with each fragment and enhance its degree of uncorrelation. In other words it is
possible to define a “self-reconstructing” packet where the header contains all the information required for reconstructing the bytes included in the local payload (see Figure 3.12). The following fields are contained in the header format: ID is the image identifier (to keep transmitter and receiver synchronized); $x$ and $y$ are the absolute coordinates of the first pixel in the payload (in the reference frame of the image); $w_{ROI}$ represents the ROI (or background image) width and $w_r$ is the residual width representing the remaining bytes to be written in the $y$ row. This encoding technique introduces a certain transmission overhead thus worsening the compression ratio and, consequently, the bandwidth used to send an image. On the other hand, it permits a stronger error-resilience because of the higher degree of uncorrelation and the granularity (self consistency) introduced at the fragment level. In order to avoid mixture of ROIs in fragments, in the simulations the final fragment referred to a ROI is padded with zeros.

![Figure 3.12. Self-reconstructing packet header fields description.](image)

To further improve on error resilience a change was made to the basic functionality of the IEEE 802.15.4 standard, which 6LoWPAN relies on, to avoid to completely discard packets with single bit errors. Indeed, according to the IEEE 802.15.4 standard only sent and received packets showing the same Frame Check Sequence (FCS) can be accepted. The FCS is evaluated on the whole data packet. The FCS has been applied on the application and 6LoWPAN headers only, in order to consider valid all those packets having sporadic bit flips in the payload. In Figure 3.13b.
our proposal is sketched in comparison with the native IEEE 802.15.4 format (Figure 3.13a). In the standard the FCS field is positioned at the end of the packet in order to check the consistency of all the bytes in it; the proposal is that of swapping it with the payload to check upon header integrity only. Indeed the proposed solution has the advantage of reducing the number of discarded packets (those having corrupted headers only) while keeping unchanged the complexity of FCS calculation (marginally reducing the actual overhead because of the reduced number of bits, domain of FCS calculation).

![Figure 3.13. Standard and proposed MAC data messages.](image)

Even if the above proposed approach guarantees to receive a bigger amount of information at the cost of accepting isolated bit errors, it does not avoid to lose whole packets with errors in the packet header. To reconstruct parts of lost images three different types of low-complexity concealment algorithms are proposed and discussed in order to improve on the perceived reconstruction quality: (i) **black concealment**, (ii) **copy-background concealment**, and (iii) **copy-image concealment**. All the mentioned techniques consist on replacing lost fragments of a video frame with certain buffers. Consequently in case of black-concealment (see Figure 3.14b) the lost fragment is replaced with black pixels; in case of copy-background concealment (see Figure 3.14c) with the relative background image pixels, in case of copy-image concealment (see Figure 3.14d) with the content of the previous reconstructed image. The “copy-image concealment” algorithm is the most memory consuming and computationally intensive since it requires extra-memory to store the previous image and computational overhead for copying data; from the performance evaluation it results to maximize Peak Signal-to-Noise Ratio (PSNR) (see next section). On the other hand, the “black concealment” algorithm has no memory overhead, it needs to set to zero a certain zone of the current reconstructed image, but its performance in terms of PSNR does not seem promising. Finally the “copy-background algorithm” is the most promising since it shows performance comparable with “copy-image” at no extra-cost in terms of memory and complexity.

Finally, in order to have a fair comparison between the proposed codec and JPEG in error-prone channels, some improvements for the JPEG are presented in order to boost its performance in presence of communication noise. JPEG format is not byte-oriented and intrinsically highly correlated, therefore there is marginal room to introduce both low-complexity error resilience and concealment techniques. For this reason, FCS is applied to the IPv6 header only: if a packet has the IPv6 header corrupted will be discarded, otherwise, even if the payload is corrupted, it will be decompressed at the application layer. In the first situation, an irrecoverable discontinuity inside the JPEG buffer occurs without any possibility of concealment. In this case the entire image is discarded and substituted with the previous retrieved image, using a sort of **global copy-image concealment**. When the bit flips occur in the IPv6 header, the decompressor attempts to reconstruct the image frame: if the errors occur within the JPEG header (where the metadata
3.2. VIDEO STREAMING APPLICATIONS IN IOT SCENARIO

Figure 3.14. Concealment techniques.

as width, height, the Huffman tables, etc. are contained) or a JPEG marker (e.g., the “End of Block (EOB)”), the decompressor cannot reconstruct the frame and the “global copy-image concealment” is applied. In all the other cases (errors occurred in the DCT blocks), the decompressor converges and reconstructs the image. All the performance results presented in the next section refer to this modified version of JPEG.

3.2.4. Performance evaluation

This section comments upon the effects introduced by environmental noise in point to point communication. All the simulations are performed using a loss trace characterized by a Bit Error Rate (BER) equal to $5 \cdot 10^{-5}$ as from the best fit of the data acquisition campaigns of the IPERMOB project. Must be stressed that all the results discussed in this section derives from simulative experiments in which the impact of bit errors is evaluated by using images with a QQ-VGA resolution coming from the dataset described later in this section. Performance results are presented by using the metrics described in Table 3.1 and Table 3.4. The section is organized as follows: in the first part the proposed codec performances are evaluated against the error resilience algorithm improvements described in Section 3.2.3, the best techniques are selected in terms of quality of reconstructed images, complexity and memory footprint.

**Performance evaluation for the proposed codec**

The impact of the FCS based error resilience technique is evaluated using the three types of concealment described in Section 3.2.3. The actual values of PSNR, data rate, and PLR
Table 3.4. Network metrics.

<table>
<thead>
<tr>
<th>Name</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data rate</td>
<td>( Data_Rate = \frac{CompressedImage_Size}{Frame_Period} )</td>
</tr>
<tr>
<td>Packet Lost Rate</td>
<td>( PLR = \frac{#Discarded_Packets}{#Transmitted_Packets} )</td>
</tr>
</tbody>
</table>

retrieved from simulation are shown in Table 3.5. As it is easy to expect a relevant reduction in PLR occurs in the case in which the FCS check is performed only on the packet header (header check): the length of the headers (40 bytes) is considerably shorter than the length of the entire packet (127 bytes) and consequently the probability of error within the headers is smaller than the probability of error occurring within the entire packet (packet check). Since the data rate is not affected by concealment policies (handled at the receiver), the best techniques are selected following arguments related to image reconstruction quality, computational complexity, and memory footprint.

Table 3.5. Performance results comparison between the different composition of the error resilience improvements.

<table>
<thead>
<tr>
<th>BRP</th>
<th>Header check</th>
<th>Packet check</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR black [dB]</td>
<td>PSNR bgnd [dB]</td>
</tr>
<tr>
<td></td>
<td>Data rate [Kbps]</td>
<td>PSNR [dB]</td>
</tr>
<tr>
<td></td>
<td>PLR [%]</td>
<td>PSNR [dB]</td>
</tr>
<tr>
<td>10</td>
<td>25.47 34.13 34.21</td>
<td>46.99 1.44</td>
</tr>
<tr>
<td>20</td>
<td>25.12 33.43 33.53</td>
<td>40.65 1.46</td>
</tr>
<tr>
<td>30</td>
<td>25.29 33.24 33.39</td>
<td>40.04 1.44</td>
</tr>
<tr>
<td>40</td>
<td>25.00 32.92 33.06</td>
<td>38.88 1.43</td>
</tr>
<tr>
<td>50</td>
<td>25.18 32.61 32.80</td>
<td>38.19 1.43</td>
</tr>
<tr>
<td>60</td>
<td>24.70 32.48 32.64</td>
<td>39.07 1.49</td>
</tr>
<tr>
<td>70</td>
<td>25.02 32.43 32.60</td>
<td>40.56 1.44</td>
</tr>
<tr>
<td>80</td>
<td>25.47 32.37 32.53</td>
<td>39.81 1.44</td>
</tr>
<tr>
<td>90</td>
<td>24.86 32.28 32.46</td>
<td>43.33 1.42</td>
</tr>
<tr>
<td>100</td>
<td>24.59 31.52 31.81</td>
<td>42.26 1.47</td>
</tr>
</tbody>
</table>

In Figure 3.15 the PSNR is plotted as a function of BRP for all considered error resilience improvements (i.e., concealment technique and integrity checks). The black concealment produces the worst performance; yet the two curves related to the integrity check techniques (i.e.,
header check and packet check) are clearly separated and highlight the impact of PLR in PSNR (having a minimum of about 20dB in case of packet check, and 25dB in the other case). From the same figure it is possible to state that the best approach in terms of reconstructed image quality is the one based on header check and copy-image concealment. However both the copy-image and copy-background concealment are very effective and produce PSNR ranging from 30dB to 35dB in both header and packet check configurations. Consequently the optimal version of the system is based on header check, because it guarantees better performance in terms of PSNR without any complexity overhead. About concealment the copy-image and the copy-background techniques are both effective: the first one is the optimal in terms of PSNR, and the second one, having a slightly lower PSNR but less computational overhead, represents the best option from cost-benefit analysis arguments.

**Comparative analysis**

In the following, a performance comparison between the proposed codec and the modified JPEG version described in Section 3.2.3 is presented. In Figures 3.16a and 3.16b PLR is plotted versus BRP and JPEG Quality indicator respectively. The values are constant and rather compatible because the header-to-payload ratio are quite similar in the two cases (i.e., 0.31 for the proposed codec and 0.24 for JPEG). This comparable performance is not translated in the perceived quality of image reconstruction (see Figures 3.16c and 3.16d) since a single bit flip either in the IPv6 or JPEG headers corrupts the full image; moreover the high correlation within a single JPEG block and among displaced blocks affects the quality of full image reconstruction also in cases where sporadic bit flips have occurred. When high quality indicators are considered, PSNR has a catastrophic drop because at high sampling frequencies all local defects are
propagated to the full image. In our compressor the PSNR is systematically higher than JPEG and shows a very weak dependence on BRP. Concerning data rate, as shown in Figures 3.16c and 3.16d, our compressor is weakly dependent on BRP; JPEG instead shows a very sensitive drop when lower quality is required. At \( Q = 70 \), the data rate is lower than in our case (33kbps instead of 38kbps); if this residual gain is not considered crucial, the JPEG compressed image can be protected by FEC or EC techniques at the cost of higher complexity. Nonetheless, although totally protecting information the result in terms of PSNR is quite unsatisfactory if compared with ours.

Table 3.6. Performance results comparison between the proposed codec and JPEG.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>34.21</td>
<td>34.13</td>
<td>46.99</td>
<td>1.44</td>
<td>100</td>
<td>21.85</td>
<td>143.46</td>
<td>1.13</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>33.53</td>
<td>33.43</td>
<td>40.65</td>
<td>1.46</td>
<td>90</td>
<td>30.86</td>
<td>58.66</td>
<td>1.12</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>33.39</td>
<td>33.24</td>
<td>40.04</td>
<td>1.44</td>
<td>80</td>
<td>31.21</td>
<td>40.78</td>
<td>1.13</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>33.06</td>
<td>32.92</td>
<td>38.88</td>
<td>1.43</td>
<td>70</td>
<td>31.20</td>
<td>33.31</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>32.80</td>
<td>32.61</td>
<td>38.19</td>
<td>1.43</td>
<td>60</td>
<td>31.12</td>
<td>28.41</td>
<td>1.11</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>32.64</td>
<td>32.48</td>
<td>39.07</td>
<td>1.49</td>
<td>50</td>
<td>30.81</td>
<td>25.63</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>32.60</td>
<td>32.43</td>
<td>40.56</td>
<td>1.44</td>
<td>40</td>
<td>30.55</td>
<td>22.72</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>32.53</td>
<td>32.37</td>
<td>39.81</td>
<td>1.44</td>
<td>30</td>
<td>30.28</td>
<td>19.73</td>
<td>1.11</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>32.46</td>
<td>32.28</td>
<td>43.33</td>
<td>1.42</td>
<td>20</td>
<td>29.58</td>
<td>16.12</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>31.81</td>
<td>31.52</td>
<td>42.26</td>
<td>1.47</td>
<td>10</td>
<td>27.83</td>
<td>11.75</td>
<td>1.08</td>
<td></td>
</tr>
</tbody>
</table>

3.3. WSN and RFID integration in the IoT scenario

Though an integration between RFID and WSN technologies has been already proposed in literature, the problem has been mainly addressed from an hardware point of view, while developing a custom application logic for the interoperability among devices. In the IoT scenario, the devices interoperability is one of the main strengths thanks to which new innovative applications can be developed. In this section a WSN and RFID integration according to the IoT spirit is proposed. The integration is proposed both at hardware and logic levels by discussing the use of IoT protocols at network and application layers. Moreover, the proposed approach is detailed, implemented, and assessed for the Smart Factory use case by developing an advanced safety system able to guarantee a safe access to factory dangerous areas in which safety equipment are required. This section presents an extension of the work presented in at the 20th IEEE International Conference on Software, Telecommunications and Computer Networks [SBP+12]. More
Figure 3.16. Comparison of PSNR, data rate and PLR between the proposed codec and JPEG.

in detail, the integration between RFID and WSN technologies is not addressed only from an hardware point of view, but an IoT based interoperability is proposed, implemented and assessed.

This section proposes and discusses the integration of WSN and RFID technologies in the IoT scenario. The proposed approach is based on the REST paradigm, thanks to which the two technologies can be seamless integrated by representing sensors, actuators and RFID related data as network resources globally addressable through state-of-the-art IoT protocols.
The integration approach is detailed for the Smart Factory use case by proposing and developing an advanced IoT-based WSN and RFID integrated solution aiming at improving safety in industrial plants. The developed system can guarantee a safe access to factory dangerous areas in which safety equipment are required.

3.3.1. Background

The use of standard network protocols (i.e., 6LoWPAN) has been just the first step towards an effective IoT-based network implementation. Another important step is constituted by the RFC of the Constrained Application Protocol (CoAP) [SHB14], developed by the CoRE IETF working group. CoAP permits to create embedded web services running on IoT nodes [She10], thus extending the web architecture, based on the REpresentational State Transfer (REST) paradigm, to resource constrained devices.

REST is an architectural style for distributed systems introduced and defined in 2000 by Fielding in his doctoral dissertation [Fie00]. According to the REST architecture a distributed system is composed by clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of the resource. The most significant example of a system conforming to the REST architecture is the World Wide Web, in which resources are manipulated by means of the HTTP protocol.

As previously introduced, the REST paradigm is also used in the IoT scenario, where resources, identified with URIs, usually represent sensors, actuators or other possible information. However, in IoT-based WSNs, the CoAP protocol is used instead of HTTP. Indeed, CoAP is similar to HTTP, but especially designed for constrained devices. CoAP allows sensor nodes to run embedded web services through which their resources can be manipulated. Specifically, CoAP (like HTTP) provides four methods for manipulating resources: (i) PUT, which requests that the resource identified by the URI specified in the request be updated or created with the transmitted representation; (ii) POST, which requests that the representation transmitted in the request be processed; (iii) GET, which retrieves a representation of the resource identified by the URI specified in the request; and (iv) DELETE, which requests the deletion of the resource identified by the URI specified in the request. In actual practice, sensor are queried using GET requests and actuators are controlled using PUT requests.

CoAP is designed to use UDP as its transport protocol. Indeed, as already mentioned, UDP is more suited to WSNs than TCP. Moreover, UDP allows multicast communication which can be exploited by CoAP to perform group communication [RD14] (i.e., to control a set of resources, such as all the lights in a room, with a single request). The drawback of using UDP is the lack of a reliability which cause CoAP messages to arrive out of order, appear duplicated, or go missing without notice. For this reason, CoAP implements a lightweight reliability mechanism, without trying to re-create the full feature set of a transport like TCP. Moreover, in order to support large resources that cannot fit a single UDP datagram, CoAP can use the block-wise transfer
extension [BS14] which performs fragmentation at application layer. CoAP, unlike HTTP, also provides a resource observation mechanism [Har14] that allows a node to receive notifications about changes in resources it has previously subscribed to.

### 3.3.2. Related works

In the last several years, an increasing number of research activities have started to propose integrated WSN and RFID systems with the aim of providing new services or improving already available applications [LBNS08]. The integration problem has been mainly addressed from an hardware point of view considering the benefits provided by an integrated solutions, and without proposing a full interoperability among devices according to the IoT paradigm. From the hardware point of view the integration between WSN and RFID can be reached by:

- Integrating RFID tags on WSN nodes;
- Integrating RFID readers on WSN nodes.

The former case consists in extending the sensing capabilities of a WSN node with those provided by RFID tags. In this scenario, the RFID tag can be used to provide location-aware services and for minimizing the WSN nodes power consumption giving to the node the capability to wake-up when triggered by RFID readers. If the integration consists in merging the RFID reader with a WSN node the main result is a low-cost pervasive extension of the WSN going towards the full accomplishment of the IoT vision.

A first example of RFID and WSN integration is presented by Chen in [Che10] where a wireless localization system for monitoring children position in theme park is implemented. In the work, the integration is reached by installing an RFID reader on each WSN node, thus creating a hybrid localization system able to estimate the child position with a maximum error of 3 meters. The work presented by Xiong et al. in [XSSG11] can be classified in the same application scenario. In their paper a grid of RFID tags is used to enhance the positioning accuracy reached by standard well-known WSN localization algorithms based on the received signal strength indicator. In this latter case the integration of both technologies is reached again by installing RFID readers on WSN nodes.

The RFID reader usually combines in a single physical device transmission (to the tags) and reception (from the tags) functions. In [DDRC+10] an innovative approach is discussed by introducing a different device, called RFID listener, able to perform receive functions with the aim of realizing a distributed sensing. The integration of RFID listeners on low-cost and low-power WSN nodes allows to realize an architecture with a single transmitter and multiple listeners, enabling a denser deployment and increasing the localization accuracy.

In the above mentioned works, RFIDs are mainly used for implementing a coarse grain localization while trying to optimize the power consumption of each WSN unit. Looking at the power consumption optimization scenario, Jurdak et al. proposed in [JRO08] a low-cost system making use of IEEE 802.15.4 transceiver as a fake RFID tag reader. In particular, their system transmits, through the installed IEEE 802.15.4 transceiver, the electromagnetic energy necessary for triggering a tag and indirectly for waking up the associated WSN node.
More recently, integrated WSN and RFID solutions have been proposed in other application
domains. In [XW08] Xiaoguang and Wei proposed the jointly use of WSN and RFID for the
development of a smart warehouse management system. In the proposed application several
possible network architectures are analyzed and discussed looking at the best trade-off between
system reliability and deployment costs. The use of WSN and RFID in a smart home scenario
is proposed by Hussain et al. in [HSM09]. Leveraging on both WSN and RFID advantages the
main idea provided by the authors is to assist elderly people by tracking their movements while
providing personalized services to increase their comfort. In [NS09] the intelligent transport
system scenario is considered. In their work Nasir and Soong propose to acquire pollutant
emission levels gathered by hybrid WSN and RFID sensors installed near the vehicles. The
pollutant levels are acquired only when an embedded RFID tag receives enough energy for
waking up the sensors. The developed solution permits on one hand to acquire environmental
data, and on the other hand, to correlate the pollutant emission level with a car identification
number (i.e., the car license plate).

In 2012 IEEE 802.15.4f Active RFID System Task Group has begun to work on a new
amendment [Soc12] that provides two PHYs (MSK and LRP UWB) to be used in a wide range
of applications requiring various combinations of low-cost, low-energy consumption, multi-years
battery life, reliable communications, precision location, and reader options. This PHY standard
supports the performance and flexibility needed for future mass deployment of highly populated
autonomous active RFID systems.

Albeit the above mentioned works greatly contribute in the design of an effective solution
aiming at integrating both WSN and RFID technologies, none of them addresses the problem
of a seamless integration going towards the full accomplishment of the IoT vision. The inter-
operability with devices compliant with Internet protocols is not taken into account, as well as
possible solutions for managing resources made available by both WSN and RFID embedded
devices.

### 3.3.3. WSN and RFID integration through the REST paradigm

The possibility of representing sensors, actuators and other possible source of information
as general resources identified by a global URL i.e., the IPv6 address of the network interface
plus a resource identification, allows to abstract the physical components of the system with
a common operation logic. Considering a WSN and RFID integrated system, in which both
RFID-Readers and RFID tags are integrated in hardware with WSN nodes, two new nodes
should be added to those presented in Figure 3.1 presented in Section 3.1 (i) Host Reader
(HR), a WSN node in which an RFID-Reader has been integrated; (ii) Host Tag (HT), a WSN
node in which an RFID-TAG is integrated. The two nodes are depicted in Figure 3.17 as part
of the 6LoWPAN network architecture, while representing the possible resource URIs exposed
by nodes. An Host (H) node can expose a simple CoAP sensor or actuator resource (e.g.,
coap://[aaaa::1]/sensor_resource and coap://[aaaa::2]/actuator_resource), while HR
and HT nodes can expose RFID related resources. More in detail, an HR node can expose
an RFID reader related resource (e.g., coap://[aaaa::3]/reader_resource), while HT can
expose an RFID-TAG related resource (e.g., coap://[aaaa::4]/tag_resource). As matter of example, a reader related resource can be an hardware configuration parameter or an aggregated information obtained by reading tags in the range. A tag related resource can be either the Electronic Product Code (EPC) memory content of a passive tag or a sensor value collected by a semi-passive tag. As matter of example, a reader related resource can be the Electronic Product Code (EPC) of a tag in the reader range or an aggregated information obtained by multiple tags. A tag related resource can be either the user memory content of a passive tag or a sensor value collected by a semi-passive tag.

The proposed approach of extending the 6LoWPAN architecture with HR and HT nodes, while using the REST paradigm through the CoAP protocol is simple in its vision, yet powerful. Indeed, the proposed solution on one hand guarantees a full integration between WSN and RFID technologies, on the other hand achieves a seamless interoperability and integration with Internet according to the IoT vision.

3.3.4. System design

The proposed approach of integrating WSN and RFID technologies by using IoT protocol solutions is detailed for the Smart Factory use case by proposing and developing an advanced safety system able to guarantee a safe access to factory dangerous areas in which safety equipment are required. Considering the industrial plant of a factory, the entire surface can be divided in several restricted areas, each of them characterized by a security access level. The access to each area must be taken under control to avoid possible dangerous situations. A minimum requirement to give access to an Area Under Control (AUC) is to check whether a worker asking for the access is wearing all necessary safety equipment. If the worker request is identified to be safe, then the AUC door can be opened and the worker can enter in the area. Though the
selected use case can be considered quite simple, it is a real scenario in which WSN and RFID technologies can be integrated.

A pictorial sketch of the AUC area with its main components is depicted in Figure 3.18. In the area, several 6LR and H nodes are deployed to collect data from the environment. Each one of them can expose a sensing or actuating resource which can be manipulated through CoAP methods by the Local Server (LS) connected to the 6LBR. Close to the door two main actors of the system are installed: (i) the HR node, in charge of detecting whether a worker identified by its own identity device, HT, is wearing all the necessary equipment, and (ii) the H node able to open the door in case the access is authorized. The HR node is exposing a coap://[aaaa::3]/rfid_reader resource from which the LS can receive detection events. In the system design the HR and H nodes have been kept separate to consider cases in which a remote actuation is required (e.g., two consecutive doors in a corridor must be opened). However, in the case reported in Fig. 3.18 the functionality of the H node could be embedded in HR, thus having a node with multiple functionality.

To better explain the proposed safety access system design a sequence diagram has been reported in Fig. 3.19 to show the main messages exchanged in the setup and operational phases of the system. In the setup phase, LS activates the subscriptions to the necessary resources, i.e., coap://[aaaa::3]/rfid_reader, by sending a CoAP message (a) to the HR node by means of the CoAP observe protocol. The HR node maintains a list of the active subscriptions, while in LS an event handler is installed at run-time and associated to received notification messages. During the operational phase, when the HT node is in the HR range, the worker identity, as well as the list of the equipment to wear, is read by accessing the memory of the tag embedded on it (b and c messages). At the same time a new CoAP resource coap://[aaaa::3]/tag_aaaa_4 is created in HR. The new resource logically represents the memory of the HT tag, and creates
3.3. WSN AND RFID INTEGRATION IN THE IOT SCENARIO

Figure 3.19. Messages exchanged among network devices.

Figure 3.19. Messages exchanged among network devices.

a virtual access to the CoAP resources exposed by HT, i.e., \texttt{coap://[aaaa::4]/identity} and \texttt{coap://[aaaa::4]/log\_location}. When all the required equipment are detected by reading their own passive T tags, the HR node sends an authorized request event (d) to LS using a CoAP method. The event is then automatically handled by LS which sends an open door request (e) to the actuator node exposing the \texttt{coap://[aaaa::2]/open\_door} resource. The list of all necessary equipment is stored in the HT node in order to keep the safety requirements as much as close to the worker, thus enabling multiple checks in case several HR nodes are installed in the AUC area. If the HT node exposes multiple RFID related resources more checks and applications can be realized. In the proposed system, after the authorization phase, the tag\_aaaa\_4 resource is handled (f and g messages) to store in the tag memory of the HT node the identification number of the AUC which the worker is entering, thus creating a per AUC tracking application. The tag\_aaaa\_4 resource virtually access to the \texttt{coap://[aaaa::4]/log\_location} resource.

In the above described system design, malicious cases in which unauthorized workers enter the dangerous area when the door has been opened by a previous successful authorization have not been taken into account. The management of such situations is considered outside the scope of this work. In any case, the problem can be easily managed by installing again several HR nodes in the AUC, thus enabling periodic checks of the safety requirements. The problem of associating a safety equipment to the wrong worker identity has been solved by storing a worker identification number on both HT and T tags. When the identity of a worker is read from HT, the HR node discards all the equipment tags in the same operational range showing a different identification number. In case of several HT nodes have been detected, the check for authorizing the entrance in the AUC area is performed sequentially for each worker, discarding not valid equipment.

**Hardware components**

The main hardware components of the system are, according to Fig. 3.18, the 6LR, the 6LBR, the H node with an actuation resource, and the two new nodes HR and HT. To develop all the integrated hardware components, as well as for the 6LBR and H nodes, the selected WSN
device is the Seed-Eye board. The board, depicted in Fig. 3.20 is equipped with a 32-bit microcontroller, an IEEE 802.15.4 transceiver, Ethernet interface and various expansion connectors. The microcontroller is a 32-bit Microchip™ PIC32MX795F512L microcontroller based on the MIPS architecture and able to reach a maximum clock speed of 80 MHz, it embeds 128 Kbyte of RAM and 512 Kbyte of Flash memory. Wireless communication capabilities are provided by the Microchip™ MRF24J40MB transceiver, characterized by a transmission frequency of 2.4 GHz with a transmission power ranging from -46 dBm to +20 dBm. Seed-Eye presents a small form factor, and can be powered by either batteries or through the USB port. The Seed-Eye, as it is, can be used for the 6LBR node by connecting them to an IPv6 network through the Ethernet interface, while the H node with the actuation capabilities has been created by embedding a relay with two exchange connectors on the board, and operating it via a serial connection. The H actuation node is reported in Fig. 3.21.

The HT node has been developed by embedding a semi-passive RFID-TAG into the Seed-Eye. A picture of the node is reported in Fig. 3.22. The selected embedded tag is the IDS SL900A chip able to communicate in the UHF bands ranging from 860 MHz to 960 MHz, and usable as passive or semi-passive RFID-TAG. Indeed, thanks to an internal real time calendar,
the SL900A chip can be operated as data-logger by connecting external sensors to dedicated pins. The SL900A has been connected to the Seed-Eye board through a serial peripheral interface, from which it is possible to read both the EPC memory of the device and the values of possibly connected external sensors.

![Figure 3.22. HT node embedding a semi-passive tag.](image)

Last key component of the system is the HR node, depicted in Fig. 3.23. It has been created by interconnecting an RFID-Reader to the Seed-Eye. The selected reader is the module Sensor ID Discovery UHF OEM which has been connected to the Seed-Eye by means of a simple serial interface. The reader supports the EPC standard for reading data from tag memories, while reaching a transmission power of 27 dBm. Thanks to the selected omni-directional antenna, and transmission power, the reader is able to read tags at a distance of around 5 meters. The choice of using RFID hardware equipment working in the frequency band ranging from 860 MHz to 960 MHz avoids any possible interference with the selected WSN devices. It must be stressed that the Seed-Eye is equipped with a radio transceiver operating in the 2.4 GHz frequency band.

![Figure 3.23. HR node embedding an RFID-Reader.](image)
Regarding the passive tags used on safety equipment, the Alien ALN-9654 G have been selected. This choice has been mainly done due to their extreme low-cost and compliance with the EPC standard.

**Software components**

To develop the firmware to be installed on all the nodes of the system the ERIKA Enterprise OS [ERI] has been selected. ERIKA OS is characterized by an extremely reduced Flash footprint, although it can provide advanced scheduling policies (e.g., fixed priority, earliest deadline first) for organizing tasks execution, as well as resources and semaphores for implementing preemption policies. ERIKA OS has been chosen, instead of other popular operating systems, such as Contiki and TinyOS, because of its real time features that allow the integration of background monitoring and maintenance tasks on system nodes. These tasks may run in the node at a low priority level, while preserving the required QoS for high priority activities, such as those related to RFID authorization. As matter of example, the HR node can embed additional sensor resources constantly monitored in background without interfering with high priority activities, thus allowing to reduce the number of nodes to be deployed. Furthermore, ERIKA OS comes with a fully compliant, lightweight IEEE 802.15.4 software stack, µWireless, that can be configured for performing time accurate, periodic and aperiodic packet transmissions. The full 6LoWPAN network stack, as well as the CoAP protocol, are part of the ERIKA communication stack. They have been implemented following the IETF recommendations. In the 6LoWPAN stack both flooding and geographical routing protocols have been implemented, namely Ad-Hoc on Demand Distance Vector and SPEED-6LoWPAN/SPEED-3D. Implementation details of all the above protocols can be found in previous works of the authors [BPP+11, MBA+11]. NOTE: SPEED-3D

In the WSN and RFID integrated solution proposed in Section 3.3.3 all the RFID based capabilities have been abstracted as device resources. In order to accomplish such a vision the RFID based software components have been implemented in ERIKA OS as system drivers. For the HR node several functions to set hardware parameters have been implemented, as well as functions for configuring the serial communication interface and for managing the EPC memory of read tags. The HT node driver consists of several functions able to configure the semi-passive tag parameters and to get values from possible sensors embedded into the tag.

Regarding the Local Server software solutions, these have been developed as custom applications able to provide web-based services for data storage, resource discovery, nodes configuration, and automatic event handling. Detailed architectural and implementation solutions can be found in [AAB+12].

### 3.3.5. Performance evaluation

The performance of the proposed system have been evaluated by means of a real deployment reflecting the scenario depicted in Figure 3.18. Both HR and H actuator nodes have been installed close to the entering door of an AUC area, while inside the area three 6LR nodes have been deployed, as well as a 6LBR and a laptop working as LS. In the deployed test bed two main experiments have been conducted with the aim of evaluating: (i) the system response time and
its accuracy in authorizing the access in the AUC, and (ii) the system response time in logging
AUC identification data on the HT node.

**System response time and accuracy in authorization control**

The system response time for authorizing a worker to enter in the AUC area has been
evaluated by connecting a laptop to both HR and H actuator nodes. When the HR node sends
an authorization request a timer starts, for being stopped when the open_door request is received
by the H actuator node. The response time has been evaluated as a function of the total number
of transmission hops necessary for sending the various requests from HT to LS and from LS to H,
thus simulating a real system in which multi-hop communications are necessary. The number of
communication hops has been modified by forcing a static routing in the 6LR nodes, i.e., a static
routing table has been written for all nodes for each experiment. In the case of two hops no 6LR
nodes are allowed to forward messages, and HR and H communicate directly with the 6LBR.
Routing protocols such as those previously mentioned (AODV, Speed-6LoWPAN/SPEED-3D)
have not been used to avoid to consider possible routing delays in the evaluation of the system
response time. Results of the performed experiments are summarized in terms of mean (95%
confidence interval) and standard deviation values in Table 3.7. The overall performance result
figures reported in the table have been obtained by performing one thousand authorization
requests for each considered network configuration. In Fig. 3.24 the response time probability
density functions for the cases of 2 and 5 hops are shown.

**Table 3.7.**
Response time as a function of the number of hops.

<table>
<thead>
<tr>
<th>Number of hops</th>
<th>Response time [ms]</th>
<th>( \mu ) ± ( \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>147.59 ± 1.54</td>
<td>24.75</td>
</tr>
<tr>
<td>3</td>
<td>150.79 ± 2.02</td>
<td>29.72</td>
</tr>
<tr>
<td>4</td>
<td>156.18 ± 1.58</td>
<td>32.04</td>
</tr>
<tr>
<td>5</td>
<td>165.32 ± 3.76</td>
<td>58.40</td>
</tr>
</tbody>
</table>

The time values reported in Table 3.7 include all the elaboration times spent in each system
node, and the network stack delays for receiving and sending CoAP messages. As it is easy to
expect, the whole response time increases, in average, as a function of the number of hops. In
any case, the overall time in authorizing the entrance in the AUC is bounded and completely
acceptable by a worker. Regarding the time distribution for each experiment, its behavior shows
a Gaussian shape with a second peak slightly pronounced at higher time values. The observed
behavior has been further investigated, and it is mainly due by ERIKA OS policies in managing
tasks in the network stack. Bigger amount of time correspond to cases in which higher priorities
tasks preempt tasks with lower priority levels, i.e., an incoming packet task preempts the data
transmission task, thus increasing the total system response time. The phenomenon occurs for each experiment, and it is more evident for higher numbers of transmission hops, see Fig. 3.24. The behavior is quantitatively revealed by the standard deviation, which increases according to the number of hops.

In the system design, an event is sent from HR to LS only when a worker is wearing all the necessary safety equipment, which means that all the passive tags T read by HR are in the equipment list stored in the HT node. The event based approach permits to have a certain percentage of false access rejections, while no false access acceptances are allowed. Must be stressed that, while the absence of false acceptances depends on the system design, possible false rejections strictly depends on the hardware equipment used for reading the RFID-TAGs. RFID-Readers able to reach higher output power levels, as well as directive antennas with higher gains can greatly reduce the phenomenon, thus significantly reducing the possibility of a false rejection. However, the detailed phenomenon can be experienced in a real scenario, thus its impact has been evaluated in the case of one or two safety equipment are necessary to enter in the AUC area. Situations in which more equipment are necessary (i.e., a bigger number of tags must be read by the reader) are characterized by worst hardware dependent performance and have not been further analyzed. More in detail, the system false rejection rate has been evaluated by creating an authorization event, i.e., all the tags are installed in the HR range at a distance of one meter from the antenna, and counting the number of reading attempts necessary to generate the event. The results of the performed analysis have been reported in Table 3.8 for the two considered cases. For each of them one thousand experiments have been performed. With the selected hardware equipment the false rejection rate is equal to 0 % after four attempts in case one passive tags plus HT have to be read, while it has a residual value of 1.5 % after six attempts in the case in which two passive tags plus the HT node are necessary to generate the authorization event.
Table 3.8.
False rejection rate vs. number of reading attempts.

<table>
<thead>
<tr>
<th>Number of reading attempts</th>
<th>False rejection rate [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HT=1, T=1</td>
</tr>
<tr>
<td>1</td>
<td>10.8</td>
</tr>
<tr>
<td>2</td>
<td>2.6</td>
</tr>
<tr>
<td>3</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>0.0</td>
</tr>
</tbody>
</table>

False rejections in generating the authorization event increase the overall response time of the system. Combining the response time results in sending the authorization event with the additional delay for generating the event itself, a figure of merit about the overall system response time can be obtained. Considering the maximum delay of a five hop communication, and the highest event generation time experienced in the case of one HT node and one T tag the overall response time is less than 400 ms, which slightly increases in case of two T tags are used. In both cases the value is bounded and completely acceptable by the worker.

**System response time in logging location data**

As previously described, by accessing to the log_location resource of the HT node, a per AUC tracking application can be implemented. In the system design, the AUC identification data are stored in the EPC memory of the HT node when the involved resource is accessed by LS through HR. The log_location request is sent immediately after the open_door request. To evaluate the system response time in logging location data on HT, a laptop has been connected to the HR node. When the log_location request coming from the LS is received by HR a timer starts, for being stopped when HR is able to read a formal acknowledgment for the successfully performed operation in the HT memory. The experimental setup does not take into account transmission delays due to multi-hop communications between LS and HR, in any case an overall value can be extracted by data in Table 3.7 by considering the number of hops from LS to HR. Each measured value is the necessary time for sending an EPC write memory request by the RFID-Reader, and for reading an acknowledgment in the memory of the semi-passive tag embedded in the HT node.

With the adopted hardware solutions, by installing HT at distance of one meter from the antenna, the average experienced logging time is $289.41 \pm 1.73$ ms (95% confidence interval), again completely acceptable by a worker waiting for entering the AUC area. The above reported result has been obtained by performing one thousand experiments. Even in this case several attempts can be necessary before reading from HT the expected acknowledgment. This behavior can be
noticed in Fig. 3.25 where in the pdf distribution two main Gaussian can be noticed, the first with a peak around 280 ms and the second with a peak around 290 ms. Moreover, values around 300 ms have been measured a few times. Better performance results can be reached by using improved RFID hardware equipment.

3.4. Conclusion

This chapter presents two examples related to combine Internet of Things protocols with new technologies where the new possibilities offered by IoT protocols play the major role.

In the first part of this chapter an innovative approach for a streaming technique fitting the usual constraints of a WCN is discussed and an innovative idea about in-node compression and inter-frame removal of redundant information is presented. At the network level, special techniques for implementing error resilience behaviors are suggested and tested against relevant metrics related to computational complexity, bandwidth usage, and perceived reconstruction quality. Realistic simulations compare JPEG, appointed as state-of-the-art solution, with our compressor. The results clearly demonstrate that our techniques outperform JPEG. The approach presented is a possible candidate for streaming applications in WSNs based on 6LoWPAN in which video streams originated by multimedia sensors can be consumed by high-end devices in the spirit of the Internet of Things.

In the second part of this chapter an integration between WSN and RFID technologies in the IoT scenario is presented. The problem is analyzed in respect of IoT enabling solutions, by proposing an integration between WSN and RFID through the REST paradigm. By representing sensors, actuators and RFID related data as network resources, addressable through CoAP protocol methods, a seamless integration between the two technologies can be reached. The result is a final integrated solution in which the outstanding pervasiveness of RFIDs and the advanced sensing and communication features of WSNs are merged together to have pervasive and addressable resources in a worldwide network of objects. To fully support the IoT based integration, the reference 6LoWPAN architecture is extended by defining two new IoT nodes:
(i) the Host Reader, a WSN node in which an RFID-Reader has been integrated, and (ii) the Host Tag, a WSN node embedding an RFID-TAG. The proposed integration approach in the IoT scenario is detailed for the Smart Factory use case by proposing and developing an advanced safety system able to guarantee a safe access to factory dangerous areas in which safety equipment are required. In the selected use case the system design is presented by detailing CoAP resources and describing their interactions. In a real test bed deployment the adopted hardware and software solutions are first described, then the system performance in respect of two main experiments is presented.
STATE-OF-THE-ART solutions in Wireless Sensor Network have several limitations in order to reach a wide adoption in our daily life. In this respect some of the main issues are: interoperability, ease of reprogramming and reliability. New generation of standards for WSN enables interoperability with Internet world (using IP and HTTP-type of protocols) and they need to be adopted in future smart sensors in order to reduce required effort for integration of WSN with other systems. The ease of reprogramming is a main requirement to be taken into account in large-scale systems where the application logic must be changed remotely and without physical access to nodes. Network reliability is another key point to consider, in fact, this issue affects the real capability of the WSN to sense and interact with the physical world. Single point of failure must be avoided in order to prevent the possibility of losing data from several devices deployed in the field.

In respect of the above mentioned issues some progress has been made in WSN interoperability. In particular, interoperability has been improved by adopting standard communication protocols at the physical and medium access control layers (e.g., IEEE 802.15.4 standard), and by adapting the IPv6 to the WSN scenario, thus really enabling the so called Internet of Things vision. The main strength of the Internet of Things is interoperability, which is achieved at network layer through the 6LoWPAN standards that allows WSN nodes to communicate using IPv6, thus making them capable of direct interaction with any other IP-enabled device. As a result, applications can make use of IoT nodes belonging to different WSNs and located anywhere in the world. The IPv6 for WSN (i.e., 6LoWPAN) is only the first step towards a global interoperability, further improvements in this direction have been reached by enabling HTTP-based transactions in WSNs. The CoAP protocol is nowadays a standard solution for enabling the RESTful architecture in IoT-based wireless sensor networks. Progress has also been made in the facilitating reprogramming of the nodes although the proposed approaches are either not so easy, limited to a specific scope, and not really suitable for very constrained devices such those used in wireless sensor networks. In this direction a very promising and challenging approach is that following a virtual machine based design leveraging on RESTful transactions.

Contributions The rest of the chapter discusses the author’s contributions to the topic In-network processing and (Ease of) Reprogramming by presenting a high-level abstractions for IoT-based WSNs, PyFUNS. PyFUNS is a Python framework for ubiquitous sensor networks. By leveraging on IoT-based protocols (i.e., 6LoWPAN and CoAP) PyFUNS guarantees a higher interoperability and reliability with respect to old-style WSNs. Moreover, PyFUNS enables ease
of reprogramming by introducing a virtual machine design based on PyMite, a reduced Python virtual machine for embedded systems.

4.1. Motivations

PyFUNS is motivated by several factors which arise from the existing need to further integrate WSN with other systems and the lack of tools to easily program and deploy new applications on the WSN by the domain experts. They can be organized in the following list:

- The proposed framework has to enable reprogramming of constrained devices using Python scripts. It has been concluded in previous studies [MDD09] that one of the major barriers for domain experts to use WSN is the difficulty to program and acquire concepts adopted by the WSN community. Python has been selected because it is very easy to learn and easy to adopt by no programming experts. It has also a very large and quickly growing community which provides open source libraries from different domains.

- In addition the goal is to abstract embedded systems complexity and networking aspects to give the possibility for the domain experts to focus their effort only on the application development.

- Currently most of the WSN deployments are based on Star topology. It is very difficult to deploy distributed applications with current WSN tool because it requires expertise in application domain, embedded and distributed systems. The framework has to enable easy deployment of distributed applications where processing tasks are shared between network nodes organized in a mesh topology. Distributed applications have higher reliability (as there is no single-point of failure such as a gateway in star topology) and they are also more scalable than centralized approaches due to spectrum reuse and reduced latency [APP13].

- Finally the proposed solution provides flexibility in terms of installing multiple applications in a network and reconfiguring the applications when the network topology changes (in order to optimize network load, application latency or energy consumption).

4.2. Related work

PyFUNS provides a number of features and several relevant solutions have been described in the literature. They have been divided into (i) techniques for remote reprogramming of WSNs, and (ii) frameworks enabling easier programming in WSNs.

4.2.1. Techniques for remote reprogramming of WSNs

System reprogramming

System reprogramming methods consists of replacing node full firmware. They are very inefficient because even minor change in an application requires reloading node binary image. Therefore they require more power and time to reprogram a node than other approaches in which only a reduced set of modules or functions is modified. Moreover, during the updating
process, the new firmware must be stored in an external flash memory before being copied into the internal flash memory when the system restarts. Therefore, the nodes must have available external flash to store full software image. System level reprogramming technique are used in some existing WSN monolithic operating systems (e.g., TinyOS [HC04]) in which the whole application consists of a single image file.

**Modular reprogramming**

In the modular reprogramming approach the node application is composed of independent, re-loadable modules. Contiki [DGV04] is an example of a modular system which consists of two main components: system core and loaded program. The Contiki Core, with the boot loader exception, is a non-reprogrammable component. Therefore, any change in the code of the kernel, program loader, symbol table and communication interfaces is not supported. However, enhanced functionality (e.g., file system support, shell support, power management) are loaded modules and are reprogrammable.

The modular reprogramming approach is suitable for over-the-air reprogramming. Unlike the monolithic method, any system change is local, only the updated modules need to be transmitted. However, a large-memory footprint and slow system execution are inherent disadvantage of any modular system. There are also other solutions implementing modular reprogramming (e.g., Dynamic TinyOS [MALW10] LiteOS [CASH08], RETOS [CCJ+07], but similarly to Contiki their use requires embedded system experts.

**Virtual Machine**

In Virtual Machine (VM) based WSN, every node runs an instance of the virtual machine. The VM is used for the execution of both on-network application packets and byte code instructions. In the literature there are several VM based approaches proposed for WSN [LC02, KP05b, LGC05, DFEV06, BLC09, SCC+06]. Mate [LC02] is a VM built on TinyOS which uses the concept of capsules - a small set of high level primitives of up to 23 bytes. Mate-based applications are composed of several capsules which can propagate throughout the network to deliver an objective. Another VM for WSN is Squawk [SCC+06] - a scale-down version of Java VM that runs without an OS on memory constrained devices. Squawk allows deployment and execution of multiple, isolated applications on a node. The use of a VM-based approach requires sensor nodes with improved resources with respect to well-known target platforms. This is because the virtual machine could be demanding in terms of CPU and memory. Considering the general trend in providing sensor nodes with higher performance at lower costs, the VM approach can be nowadays considered an effective and powerful solution in the WSN scenario.

**Differential**

The use of a differential reprogramming scheme is mainly based on the use of code patches: a user generates a patch using the difference between the old and the updated program. Rsync [TM98] is a well-known differential update scheme, and its functionality has been demonstrated in WSNs [JC04]. As working principle, Rsync divides the program into different blocks and calculates the hash value of those blocks. The hash values are then matched to determine the
block insertion, deletion, or modification. There are many other examples of differential reprogramming systems \cite{KP05a, RL03}, and in general it has been shown that the size of the deltas produced by the differential-based approaches is very small compared to the full binary image. However, most of them poorly perform when there is a change of both program and variable layout. This is because such update requires full flash memory writing, and large amount of additional external flash memory. Also in this case, differential solutions can be used only by embedded system experts.

### 4.2.2. Frameworks enabling easier programming of WSN

Many solutions for enabling an easier WSN reprogramming have been described in the literature \cite{MP11, SG08}. They were designed with different objectives, including application energy-efficiency, scalability, failure-resilience or collaborative data processing. In this respect it must be underlined that one of PyFUNS main goals is to reduce required expertise in embedded systems for reprogramming WSNs, as this has been previously identified by domain experts \cite{MDD09} as one of the major barriers for deploying WSNs. In that study the authors implemented the BASIC programming language for sensor networks and conducted a user study with novice programmers. Half of users with no previous programming experience of any kind were able to program simple network tasks using developed BASIC programs while only 0-17% could do so in TinyScript. Therefore the authors concluded that current WSN languages require knowledge of either very low-level systems development (including the details of sensor hardware and embedded system design), or high-level programming concepts and abstractions that are not obvious to most application domain experts. And because application domain experts have little programming experience, most of which is with simple single-threaded imperative programming models, the authors have ported a small BASIC interpreter to a WSN platform. Authors’ motivations are coherent with ours, although the proposed solution provides more features (e.g., interoperability due to IPv6 and CoAP protocols) and is based on Python interpreter.

Recently several publications described solutions to program WSNs by using Python language due to popularity and ease-of-use of this language \cite{APP13, CP10}. However, either they are at the early stage of development and incomplete to be used in real world applications \cite{CP10}, or like T-Res \cite{APP13} they have limited flexibility to be applied in a broad range of scenarios. T-Res enables reprogramming of the node to execute simple data-processing tasks performing the following actions: (i) monitoring one or more resources, (ii) executing some data-processing on their values, and (iii) sending the resulting output to other resources.

Another approach to simplify configuration of WSN is based on visual programming language. An example of solution for visually reprogramming WSNs is Clickscript \cite{GTW10} which enables creating Web mashups by connecting resources and operations (e.g., greater than, if..then, loops) building blocks together. However, the code is executed centrally and WSN nodes are only exposing theirs resources (e.g., sensor readings, actuation interface). Therefore it cannot be used in distributed processing scenarios or when the software image of the devices must to be changed.
4.3. PyFUNS design

Having identified the limitations of literature systems aiming at enabling remote reprogramming in WSNs, a new framework called PyFUNS that can be used in an easy way to reprogram wireless sensor networks has been designed. The proposed framework leaves to the user only the application development task in the form of Python scripts, while abstracting low level and networking functionality.

4.3.1. Dynamic services over WSN

Traditional WSN solutions enable the development and deployment of pervasive networks aiming at providing many simple services, such as the monitoring of the surrounding environment or the basic actuation control through basic operations. With the introduction of the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) standard protocol and Constrained Application Protocol (CoAP), following the Internet of Things (IoT) vision, WSNs have acquired enough smartness to accomplish more complex services, such as the capability of exposing equipped sensors in Internet to perform automatic control operations. The next natural step in the WSNs domain is to build a smart management of dynamic services, thus enabling the possibility of remotely reprogramming the services provided by an IoT-based WSN.

In general terms a service provided by a WSN is a set of operations to be performed with the aim of performing a specific task. For instance, a service can be the automatic light control in a room and the operations to be performed are: (i) check the light value periodically, (ii) check the presence of people in the room, (iii) switch on the lamp while setting the power according to the desired light value, and (iv) switch off the lamp when people leave the room.

As previously stated, PyFUNS enables the management of dynamic services in WSNs. In the rest of chapter the aforementioned definition of service (i.e., a set of operations) is followed, calling the operations to be performed applications.

4.3.2. Application components

An application deployed on a sensor node has several components:

- **Name**: string of characters that uniquely identifies the application;
- **Period**: it is related to the periodicity of the application execution. Values bigger than zero mean periodicity, equal to zero is for one time executions, while less than zero mean application blocked waiting for an answer. Application flow changes based on the Period value;
- **Timer**: used in case of periodic application, and when it fires the application is executed;
- **State**: indicates the current state of the application in its internal Finite State Machine (FSM);
- **Script**: it contains the Python byte code performing the specific task which the application has to provide;
- **Variables**: list of variables required to store data to be exchanged among different scripts of the same application or among different applications;
• Requests: list of active requests. A request is used to retrieve the current representation of a resource through network messages. Each request is associated to both a callback function, called when a reply is received, and a variable, which is used to store the received data.

To the end of building an abstract framework that allows to implement applications able to perform data communication through the network (i.e., request/reply paradigm), the application has been decomposed with three sub-scripts: PreScript (optional), MainScript (mandatory) and PostScript (optional). PreScript allows to send data request messages to a specific node in the network, and the answer will be processed in the MainScript. Moreover, it allows to set up the application environment (i.e., to create the variables required), and to retrieve the resource representation. PostScript is executed when the application has been stopped, and is mainly used to clean the application environment (i.e., to delete variables or active requests). PreScript runs once at the application start, whereas PostScript runs once at the application stop. MainScript is the only mandatory byte code to be installed on the nodes, and represents the core of the application. It can be run once or several times according to the Period value. The execution of the MainScript can be triggered by a periodic event, the expiration of a timer, or by a sporadic event, the reception of a message. Figure 4.1 illustrates the script flow for an application using all the three described scripts.

![Figure 4.1. Scripts flow chart.](image)

### 4.3.3. Application life-cycle

The finite state machine model has been used to implement the life-cycle of the application that can be dynamically installed, started, stopped, updated and uninstalled. To enable the aforementioned operations, five different states have been defined: (i) new, all the memory required to store the application structure has been allocated successfully; (ii) installed, scripts have been installed on the node; (iii) resolved, application is ready to execute; (iv) running, application is active and performs its operations; and (v) uninstalled, the application structure has been deleted and the memory has been released. Figure 4.2 depicts the application life-cycle and the possible transitions among states.

The application life starts from the new state, in which the necessary memory is allocated to store the components described in Section 4.3.2. All the components are set to a default value, except for the name which is filled when the application is created. In the new state it is possible to install PreScript, MainScript and PostScript on the node. As previously stated MainScript
is mandatory for each application and installing it implies a change of state to \textit{INSTALLED}. In the \textit{NEW} state it has been enabled the possibility to uninstall the application through a defined \textit{uninstall} event. In the \textit{INSTALLED} state all necessary components for the application are set, even though they are still waiting for a control check aiming at verifying the compatibility among scripts (e.g., compare scripts version). The check is triggered by a defined \textit{check} event, and in case all the tests are passed, the state changes to \textit{RESOLVED}. Also in the \textit{INSTALLED} state it is possible to trigger an \textit{uninstall} event to delete the application. Once the application reaches the \textit{RESOLVED} state it has been successfully checked and it is ready to be executed. Three different events can be triggered from this state: (i) \textit{start}, to run the application, \textit{PreScript} is executed in case it is present, otherwise \textit{MainScript} is interpreted, as result the state moves to \textit{RUNNING}; (ii) \textit{update}, to perform any changes concerning the scripts (i.e., install, update or delete Python scripts on the node), in this case the state moves to \textit{INSTALLED} and the check compatibility on the new installed scripts must be redone; and (iii) \textit{uninstall}, to remove the whole application and deallocate the memory occupied by the application, next triggered state is \textit{UNINSTALLED}. In \textit{RUNNING} state the application can be executed one or many times according to the Period, and can be stopped through a dedicated \textit{stop} event. \textit{PostScript}, if present, is executed during the transition from \textit{RUNNING} to \textit{RESOLVED}. Last state is \textit{UNINSTALLED} where the application, and all its components, are deleted from the node. Table \ref{tab:state-transitions} summarizes the state transitions of the above described finite state machine.

### 4.3.4. Application flow

As mentioned in Section 4.3.2 the application flow, in particular when \textit{MainScript} is executed, depends on the value of the application period. Two different period categories have been defined: period equal to zero when \textit{MainScript} runs one time, and period not equal to zero when \textit{MainScript} can run zero, one or many times. Figure 4.3 shows different flow chart depending on the value of period, Figure 4.3a is for the first category, while Figure 4.3b and Figure 4.3c for the second.
Table 4.1. Application state transition.

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>install</td>
<td>INSTALLED</td>
<td>At least MainScript has been installed</td>
</tr>
<tr>
<td></td>
<td>uninstall</td>
<td>UNINSTALLED</td>
<td>Application deleted</td>
</tr>
<tr>
<td>INSTALLED</td>
<td>check</td>
<td>RESOLVED</td>
<td>Application ready to execute</td>
</tr>
<tr>
<td></td>
<td>uninstall</td>
<td>UNINSTALLED</td>
<td>Application deleted</td>
</tr>
<tr>
<td>RESOLVED</td>
<td>start</td>
<td>RUNNING</td>
<td>PreScript executed, if installed</td>
</tr>
<tr>
<td></td>
<td>update</td>
<td>INSTALLED</td>
<td>Changes in installed scripts</td>
</tr>
<tr>
<td></td>
<td>uninstall</td>
<td>UNINSTALLED</td>
<td>Application deleted</td>
</tr>
<tr>
<td>RUNNING</td>
<td>run</td>
<td>RUNNING</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>stop</td>
<td>RESOLVED</td>
<td>PostScript executed, if installed</td>
</tr>
</tbody>
</table>

Figure 4.3. Script flow chart for period equal to zero (a), period not equal to zero (b) and period less than zero, particular implementation (c).

In case of period equal to zero (Figure 4.3(a), the application goes from PreScript to PostScript directly, running MainScript one time. It is not possible to stop the application once it is started. This setting of period is useful for applications with the purpose of changing the resource representation only one time.

With period not equal to zero (Figure 4.3(b), after PreScript, the application waits for an event to continue its execution. Two types of events that trigger MainScript have been defined: periodic and sporadic. Periodic applications have period value greater than zero and they wait the expiration of a timer before to interpret MainScript. This setting is useful to implement applications with the purpose to change the resource representation periodically. In case of sporadic application the period value is set with at a value less than zero and MainScript is called when an sporadic event happens (e.g., message received). This type of setting is useful to implement applications that perform activities in case of observed resources changes. A particular application flow based on a period less than zero has been implemented, Figure 4.3(c), in order
to provide applications with the purpose to run MainScript once after resources representation are retrieved.

### 4.3.5 Application RESTful interface

The goal of PyFUNS is to enable easy managing of dynamic application installed in ubiquitous networked sensors. To reach a seamless integration of the framework in tiny motes it is necessary to abstract the application and its attributes. This can be done by using the REST paradigm in the context of IoT-based WSNs, or in other words by using the CoAP protocol, thus allowing sensor nodes to abstract resources and run embedded web services. Abstracting application and its attributes as CoAP resources enables the use of well-known HTTP methods, GET, PUT, POST and DELETE, to administer code installed in a WSN. Moreover management of the application, (e.g., start, stop) can be performed by a user through a web site, or by another application through simple CoAP messages.

As described in Section 4.3.2, an application is defined by its components which are managed in PyFUNS as sub-resources of /apps. The resulting application structure is shown in Figure 4.4. Resource /apps is created statically during startup phase. This resource is the container of all applications installed on the node and it can be managed through CoAP methods to list currently installed applications and check the validity of one of them specific. The methods of /[app_name] provide the services to create/delete a specific application, retrieve the current state of the application, and start/stop its execution. The /[app_name] resource and its sub-resources are created by allocating the required memory only once, when the application is installed. The use of CoAP methods to manage the execution of a specific application (start/stop) enables the possibility to install on a node several applications related to each other in order to implement complex services.

```
/apps # list currently installed apps [GET]
    # check a specific app [PUT]

/[app_name] # retrieve the application state [GET]
    # create/delete a specific app [POST|DELETE]
    # start/stop a specific app [PUT]

/period # retrieve/update the period [GET|PUT]

/preScript # retrieve/update/delete the PreScript [GET|PUT|DELETE]
  /version # retrieve/update the PreScript version [GET|PUT]

/mainScript # retrieve/update/delete the MainScript [GET|PUT|DELETE]
  /version # retrieve/update the MainScript version [GET|PUT]

/postScript # retrieve/update/delete the PostScript [GET|PUT|DELETE]
  /version # retrieve/update the PostScript version [GET|PUT]

/variables # list currently variables [GET]

  /[var_name] # retrieve/observe/update the value [GET|PUT]
```

Figure 4.4. The structure of an application resource.

Resource /period represents the current application period value, and must be set following the rules described in Section 4.3.4. A set of methods are provided to manage the scripts: for
each script, *PreScript*, *MainScript* and *PostScript*, it is possible to retrieve/update/delete the byte code and retrieve/update the version of them. */variables* resource is the container of the variables used by the application to accomplish its functionality. By interacting with it, the list of currently variables can be retrieved. For each variable a new resource is created and it is possible to retrieve/update the value. The purpose of this resource is to exchange data among different scripts of the same application, or among different applications. Each */[var_name]*/ resource can be observed, even by other applications, enabling a smart functionality to be used in complex systems.

### 4.3.6. PyFUNS implementation

Native code replacement and loadable modules on the one hand enable services updates, on the other hand imply a higher cost since downloaded modules are more coarse-grained compared to a virtual machine application. Moreover, these methods require to maintain information about the software version in each node of the network, and the implementation is highly hardware dependent. Therefore, to fully decouple applications from the sensing infrastructure, to run the applications a virtual machine has been adopted.

Most of the virtual machine based approaches enable highly efficient updates: low cost for transmitting new code and abstraction from the platform. The software updates sent from front-end-device to different nodes (based on different platform) are always the same. However, VMs introduce overhead in term of memory and computational overhead, which is overcome by more powerful devices present on the market. Python, Java and JavaScript are the most common interpreted languages used for virtual machine approaches with substantial libraries of pre-written code. The last two are object-oriented languages; whereas Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. Java script is too big to be installed in a WSN node and it cannot be compiled into byte code. Using byte code for reprogramming leads to an extremely powerful system in which microcontroller can be programmed interactively without the typical compile/link/flash/run cycle. Both Python and Java allow for platform-independent processing functions that can be freely exchanged among nodes, but the former approach has been preferrend because programming in Python is really simple and supports multiple programming styles.

A prototype of PyFUNS has been implemented on top of the Contiki Operating System [DGV04] that provides native support for 6LoWPAN and CoAP. A Python interpreter has been ported to the target operating system to enable script interpretation on constrained devices. PyMite [PyM] has been ported. It is a reduced Python interpreter that runs a significant subset of the Python language on microcontroller with very few resources.

PyFUNS provides a set of APIs, summarized in Table 4.2, that can be used in Python scripts to implement applications. Such APIs allow: (i) to manage variables (create/delete/get/set); (ii) to send a generic CoAP message specifying the method (GET, POST, PUT, DELETE), the node address, the URI of target resource, the eventually payload and the eventually variable where store the result of the operation; (iii) to set/unset observation to a specific resource defined by its IPv6 address and URI; and finally (iv) to stop the execution of the application. The IPv6
address parameters is expressed without the prefix (e.g., [0,0,0,2]), as the messages exchanged among different applications, in this version of PyFUNS, can be performed only inside the same network. Notice that sendMsg and obs functions have a parameter var to be associated with the request. In case of var is not present, it is automatically created inside the functions.

4.3.7. Example of usage

To better understand how PyFUNS works, this section presents two examples of the framework usage: Security service, composed of one application, and Light Control service, composed of two applications.

**Security service example**

Security service has purpose to detect any motion in a room and trigger an alarm. Network is composed of three PIR sensors, on nodes 2, 3 and 4 with the URI coap://[aaaa::2]/sen/pir, coap://[aaaa::3]/sen/pir and coap://[aaaa::4]/sen/pir respectively, and one alarm, on node 5 with URI coap://[aaaa::5]/act/alarm. The application implementing the service can be installed in any node inside the network using the RESTful interface defined in Section 4.3.5. The intent of Security service is to observe the PIR sensors, and trigger the alarm whenever a notification of motion detection is received. To implement such envisioned application it is necessary to write and install the PreScript, MainScript and PostScript. PreScript, Figure 4.5, issues OBSERVE messages to all three PIR sensors and associates the requests to variables, p1, p2 and p3, used to maintain the representation of the sensors. Since the MainScript is run whenever a notification is received, the period of the application is set with a number less than zero: execute MainScript after a sporadic event happens (Figure 4.3.c).

As mentioned, MainScript, Figure 4.6, is called whenever a notification from observed sensors is received. The operations carried out are very simple: retrieve the representation of the variable associated to each PIR sensors and issue a PUT request to coap://[aaaa::5]/act/alarm to trigger the alarm, if one of the variables is equal to one, or to stop the alarm otherwise. Figure 4.7
from pyfuns import *

obs([0,0,0,2], "sen/pir", "p1")
ob([0,0,0,3], "sen/pir", "p2")
ob([0,0,0,4], "sen/pir", "p3")

Figure 4.5. The PreScript of Security application.

shows the Python script related to *PostScript*. It sends messages to the PIR resources in order to delete the subscription when the application has stopped. The byte code of scripts to be installed on the node can be obtained by compiling the presented Python scripts.

from pyfuns import *

if getVar("p1") or getVar("p2") or getVar("p3"):
    sendMsg(3, [0,0,0,5], "act/alarm", "1")
else:
    sendMsg(3, [0,0,0,5], "act/alarm", "0")

Figure 4.6. The MainScript of Security application.

Light Control service example

*Light Control service* performs the lamp management based on the presence of person inside a room and on the value of bright. In this example, network is composed of two PIR sensors, on nodes 2 and 3, with the URI *coap://[aaaa::2]/sen/pir* and *coap://[aaaa::3]/sen/pir* respectively, one light sensor, on node 4 with URI *coap://[aaaa::4]/sen/light*, and one lamp, on node 5 with URI *coap://[aaaa::5]/act/lamp*. To accomplish *Light Control service* it is necessary to implement two different applications interacting between them. The first one with the purpose of detecting any motion inside a room, called *Motion Detection application*, and the second one with the aim to retrieve the light value and turn on the lamp, called *Lamp Control application*. Figure 4.8 shows the PreScript of Motion Detection application: it issues OBSERVE messages to PIR sensors and associates the requests to variables, p1 and p2, used

from pyfuns import *

delObs([0,0,0,2], "sen/pir")
delObs([0,0,0,3], "sen/pir")
delObs([0,0,0,4], "sen/pir")

Figure 4.7. The PostScript of Security application.
to maintain the representation of the sensors. *MainScript*, Figure 4.9, is executed whenever a change in the representation of PIR resources happens. It retrieves p1 and p2 value and if at least one of them is equal to one, motion detected inside the room, issues a PUT request to coap://[aaaa::5]/app/lum to start the execution of Lamp Control application, otherwise, any motion detected, issues a PUT request to coap://[aaaa::5]/act/lamp to switch off the lamp. *PostScript*, Figure 4.10, simply remove the subscription sending messages to PIR resources. As in the example of *Security* service, the period of Motion Detection application must be set with a number less than zero: run the *MainScript* after a sporadic event happens.

Lamp Control is the second application of the service with the aim of checking the light value and, depending on it, it set the brightness of the lamp. The activation of Lamp Control is performed by Motion Detection through a CoAP message. The application is composed of a *PreScript*, Figure 4.11 and *MainScript*, Figure 4.12. The aim of Lamp Control is to retrieve the light value and switch on the lamp setting the brightness of the lamp. After sending a message to get the value of the light to coap://[aaaa::4]/sen/light, in *PreScript*, the application waits the replay to proceed the execution (Figure 4.3.a). Once *MainScript* is activated, it retrieves the light value by calling *getVar*. If the light is less than 100 lux, PUT request is issued to switch on the lamp with brightness 1, if the light is less than 50 lux, PUT request is issued to switch on...
from pyfuns import *

sendMsg(1, [0,0,0,4], "sen/light", ",", "light")

Figure 4.11. The PreScript of Lamp Control application.

from pyfuns import *

if getVar("light") < 100:
    sendMsg(3, [0,0,0,5], "act/lamp", "1")
elif getVar("light") < 50:
    sendMsg(3, [0,0,0,5], "act/lamp", "2")

Figure 4.12. The MainScript of Lamp Control application.

the lamp with brightness 2, if the light is greater than 100 no action are performed. Given the nature of Lamp Control application, run only one time, the period is set to 0. Both applications can be installed wherever inside the WSN. The only main requirement in writing PreScript of Motion Detection, that sends a message to activate another application, is to indicates the right address of the node in which the Lamp Control application is installed. This limitation can be overcome in future implementation by using the Resource Directory provided by Constrained RESTful Environments (core) IETF working group.

4.4. Performance evaluation

To evaluate PyFUNS performance, it has been implemented on top of Contiki Operating System by integrating/porting PyMite on two constrained platforms: (i) WiSMote, which is equipped with a MSP430F5 microcontroller having 16 kB of RAM and 256 kB of flash memory, and (ii) CC2538dk, which is equipped with an ARM Cortex™ M3 microcontroller having 32 kB of RAM and 512 kB of flash memory. In the following of this section the feasibility of PyFUNS has been proved by checking that in both selected target platforms the performed implementation requires flash memory and RAM which are within the physical limits. Then, PyFUNS overhead has been evaluated in terms of run time and energy consumption. Finally an extensive evaluation of PyFUNS framework is presented by implementing two real services: Security and Light Control. To deploy the system bases on real platform, and test it in a real life scenario, the following components has been integrated: (i) sensors, such as PIRs, and (ii) actuators, such as alarms, on target platforms. All sensors and actuators drivers have been implemented in Contiki. Figure 4.13 shows several platforms running PyFUNS, some CC2538dk platforms equipped with PIR sensors and alarms.
4.4. PERFORMANCE EVALUATION

4.4.1. Flash and RAM requirements

In order to assess the feasibility of PyFUNS on the selected hardware platforms suitable for WSN deployments, both the Flash and RAM occupation have been measured. Table 4.3 shows the memory occupied by the software for both platforms, the WiSMote and the CC2538dk. The software installed on each WSN node includes the Contiki Operating System, the PyMite interpreter, PyFUNS, plus the possibly required memory to install two PyFUNS applications.

<table>
<thead>
<tr>
<th>Node type</th>
<th>RAM</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[bytes]</td>
<td>[bytes]</td>
</tr>
<tr>
<td>WiSMote</td>
<td>14,918 (93%)</td>
<td>98,077 (38%)</td>
</tr>
<tr>
<td>CC2538dk</td>
<td>19,904 (62%)</td>
<td>96,732 (19%)</td>
</tr>
</tbody>
</table>

In case of WiSMote platform the whole firmware occupies 93% of the available RAM and 38% of the available Flash. Considering the CC2538dk platform the firmware requires the 62% of the available RAM and the 19% of the available Flash. Such a notable occupation of memory, especially RAM, is mainly due to PyMite, which alone requires 45 kB of flash memory and 8 kB of RAM. In order to reduce the RAM occupation, it is possible to implement in next version of the framework a tool to store Python byte codes into the flash memory, since the current version of PyFUNS stores the Python scripts in RAM, which is usually constrained respect to the Flash memory.
4.4.2. Native code versus Python script

PyFUNS overhead in terms of run time and energy consumption has been evaluated with respect to a native code solution. Both performance figures have been measured by using two different set of benchmarks: (i) five test applications implementing algorithms showing a different complexity level; (ii) three applications implementing CoAP methods. As previously stated, each benchmark test has been executed by considering a native code solution written in C language, and its Python version as required by PyFUNS.

The first benchmark set is composed of five algorithms, characterized by different complexity levels, and chosen from “*dada’s perl lab*” More in detail, the following algorithms have been selected, listed in function of their complexity (from lower to higher): (i) ACK - Ackermann’s Function(3, N) that is a classic recursive function with N=3; (ii) FIB - Fibonacci Numbers(N) that computes the Fibonacci sequence with N=17; (iii) MAT - Matrix Multiplication(N) that performs the multiplication between two matrices with size 5 and N=10; (iv) HEAP - Heapsort(N) that sorts a vector with a size N=100 of integer numbers, and initialized with strictly decreasing value; and (v) MET - Method Calls(N) that implements activation of class methods using object-oriented style.

The second benchmark test, instead, includes: (i) an application that issues a PUT request to a resource installed in a neighbor node (PUT); (ii) an application that issues a PUT request to a resource installed in a neighbor node and waits the acknowledgment message from the resource (PUT2); and (iii) an application that issues a GET request to one resource installed in a neighbor node, waits the reply, processes it and sends a PUT request to another resource installed in a neighbor node (GET). All performance results for both set of applications are reported in Table 4.4.

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>Python</th>
<th>Python/C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (ms)</td>
<td>Energy (µJ)</td>
<td>Time (ms)</td>
</tr>
<tr>
<td>ACK</td>
<td>4.08</td>
<td>0.029</td>
<td>645.25</td>
</tr>
<tr>
<td>FIB</td>
<td>9.95</td>
<td>0.072</td>
<td>1344.84</td>
</tr>
<tr>
<td>MAT</td>
<td>5.06</td>
<td>0.037</td>
<td>687.31</td>
</tr>
<tr>
<td>HEAP</td>
<td>1.95</td>
<td>0.014</td>
<td>379.68</td>
</tr>
<tr>
<td>MET</td>
<td>1.16</td>
<td>0.009</td>
<td>207.28</td>
</tr>
<tr>
<td>PUT</td>
<td>1.22</td>
<td>0.009</td>
<td>5.35</td>
</tr>
<tr>
<td>PUT2</td>
<td>8.61</td>
<td>0.328</td>
<td>12.68</td>
</tr>
<tr>
<td>GET</td>
<td>17.26</td>
<td>0.604</td>
<td>26.19</td>
</tr>
</tbody>
</table>

1A benchmark comparison of a number of programming languages: [http://dada.perl.it/shootout/craps.html](http://dada.perl.it/shootout/craps.html)
All data reported in the table have been obtained by running each test 1000 times in Cooja, the Contiki network simulator. Cooja allows to run the same binary files to be used on real platforms while enabling a quick testing and debugging of the system. In the simulator all tests have been performed by using only the WiSMote platform (CC2538dk is not supported at time of writing), moreover to prove the Cooja accuracy, two benchmark tests have been ran also on a real WiSMote platform. In Table 4.4 the C and Python columns show the run times (in ms) and the energy consumption (in $\mu$J) for all benchmark applications, while the last column labeled as Python/C reports the ratio between PyFUNS and native code approaches. For the first benchmark set the time performance penalty of PyMite is between 135 and 195, while showing a performance gap between 137 and 198 in energy consumption. Such a notable difference between C and Python is mainly caused by the extensive use of the heap memory in PyMite when performing complex operations such as recursive calls. On the contrary, in CoAP methods tests (second benchmark set) the run time performance penalty is between 1.5 and 4.4 with with an energy consumption performance gap between 1.1 and 4.3. Considering such a results, the use of PyMite to perform CoAP methods in WSNs leads an insignificant penalty in terms of run times and energy consumption, while enabling a powerful tool providing platform abstraction at the cost of a bigger overhead in running complex algorithms.

To prove the validity of the aforementioned results obtained with Cooja simulator, the Python version of Ackermann’s Function and PUT method have been ran also on a real WiSMote platform. The obtained results are reported in Table 4.5, and they are very similar to those obtained by using Cooja simulator and reported in Table 4.4.

<table>
<thead>
<tr>
<th></th>
<th>Time (ms)</th>
<th>Energy ($\mu$J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACK</td>
<td>649.79</td>
<td>4.799</td>
</tr>
<tr>
<td>PUT</td>
<td>5.52</td>
<td>0.040</td>
</tr>
</tbody>
</table>

**Table 4.5. Performance benchmarks on WiSMote.**

4.4.3. Real case evaluation

PyFUNS performance in providing real services, such those presented in Section 4.3.7, has been evaluated in terms of energy consumption, actuation delay and network traffic by using the already presented Python scripts.

To better assess the performance of the proposed framework, the impact due to in-network application deployment by considering various network topologies has been evaluated. The reported energy consumption represents the total amount of energy consumed by all network devices, the actuation delay represents the elapsed time between the detection of the event and the associated actuation, while network traffic measures the total amount of bytes exchanged in the network. To better understand PyFUNS impact in all the above performance figures only the CoAP messages have been considered, thus avoiding to consider both management and routing messages (e.g., RPL messages). In considering in-network application deployment...
strategies two cases have been considered: (i) one in which PyFUNS is installed inside the WSN (distributed approach) and (ii) another one in which PyFUNS is installed in the WSN Border Router (centralized approach). With the aim of evaluating the total energy consumption, actuation delay and network traffic in a comparable manner for several network topologies and in-network application deployments, all presented results have been obtained by using the Cooja simulator. Such a study is not feasible in a real scenario because of the required additional equipment to be installed on each node for measuring necessary parameters.

Both services presented in Section 4.3.7: Security and Light Control, have been simulated by considering a multi-hop IoT-based WSN by considering the nine network topologies reported in Figure 4.14: three star topologies with 5, 9, and 13 nodes, one mesh topology with 13 nodes, and 5 tree topologies each one of them with 9 nodes and different transmission links. At a first stage, all simulations performed to compare the performance among the various topologies have been obtained by keeping fixed the transmission power. Only for the topology reported in Figure 4.14(a) the additional case in which an higher transmission power is used has been considered. This is with the aim of comparing in terms of energy consumption topology (a) with the increased transmission power with respect to topology (b) using the basic transceiver power transmission. As previously stated, for each topology several in-network application deployment strategies have been simulated (i.e., PyFUNS framework installed in different nodes of the network). In this respect, it must be underlined that an application sending a message to a resource exposed by the same node does not generate any network traffic (loopback feature provided by the IPv6 protocol).

![Diagram](image)

Figure 4.14. Network topologies: star (a-b-c), mesh (d) and tree (e-f-g-h-i).

**Security service**

In the simulated Security service scenario all topologies reported in Figure 4.14 have been taken into account, while considering for each one of them node 1 as network Border Router (BR), nodes 2, 3 and 4 as devices equipped with the PIR sensor, and node 5 as the actuator device equipped with the alarm. Regarding all the other nodes of the network, these are simple 6LoWPAN routers without additional sensors or actuators.
For each topology the case in which the application is installed in several in-network positions has been considered. Table 4.6 reports all the nodes in which the Security service has been installed. As general rule we can say that in all cases the configuration in which the Security application runs on the BR has been considered (centralized approach), while testing several solutions in which it runs on sensor and actuator nodes or close to them, one or two hop distance (distributed approach).

Table 4.6. Security Control service deployment configurations.

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(f)</th>
<th>(g)</th>
<th>(h)</th>
<th>(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR (1)</td>
<td>BR (1)</td>
<td>BR (1)</td>
<td>BR (1)</td>
<td>PIR2 (2)</td>
<td>PIR2 (2)</td>
<td>PIR2 (2)</td>
<td>PIR4 (4)</td>
<td>Node 6</td>
</tr>
<tr>
<td>Alarm (5)</td>
<td>Node 6</td>
<td>Node 9</td>
<td>Node 10</td>
<td>Node 13</td>
<td>Node 10</td>
<td>Node 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 6</td>
<td>Node 10</td>
<td>Node 8</td>
<td>Node 8</td>
<td>Node 8</td>
<td>Node 8</td>
<td>Node 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node 9</td>
<td>Node 9</td>
<td>Node 9</td>
<td>Node 9</td>
<td>Node 9</td>
<td>Node 9</td>
<td>Node 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figures 4.15 and 4.16 show the energy consumption measurement for star/mesh and tree topologies respectively. As expected, in the star topology the minimum energy consumption is obtained by installing the PyFUNS application on the network Border Router (purple columns in Figure 4.15). This is because in this configuration the amount of data exchanged within the network is less than all the other cases, as is possible to see in Figure 4.17. In fact, when the transmission is between nodes distant more than one hop an additional 6LoWPAN header overhead (due to the addressing and hop limit fields) can be experienced. It must be noticed that in case a higher transmission power is used, topology (a) at 5 dBm (red columns), a significant energy consumption reduction can be experienced with respect to topologies with a bigger number of nodes and lower transmission power, topology (b) (yellow columns), although they present the same coverage area. The additional energy consumed by increasing the transmission power in all nodes is less than the total energy consumed by adding four more nodes. Regarding the mesh topology, reported in the same graphs of star topologies, in most the cases an in-network approach reduces the energy consumption respect to a centralized solution (application on BR). In any case the number of transmission hops play an important role in the total amount of network traffic, and consequently in energy consumption. Such a behavior can be noticed also for the tree topologies, see Figures 4.16 and 4.18. For instance, by considering topology (e) (purple columns) it is possible to see that the energy consumed when the application is installed in nodes PIR2 and 6 is bigger than a centralized approach (application on the BR node). By considering only the energy consumption parameter the best choice for (e) (f) (g) is node 8, with a consumed energy equal to 7.47 mJ, 7.12 mJ and 5.43 mJ respectively, for (h) is node 4 with 5.36 mJ, and for (d) (i) is node 5 with 5.85 mJ and 7.39 mJ respectively.
Last metric to be discussed for evaluating PyFUNS is the delay time introduced by the framework in triggering the actuator node when a motion detection event happens. The delay time is a significant metric, especially in services where the actuation time plays an important role...
4.4. PERFORMANCE EVALUATION

Figure 4.18. Security - Network traffic in tree (e-f-g-h-i) topologies.

(i.e., security services). Figures 4.19 and 4.20 present the Delay time for the considered Security scenario, in any case it strictly depends on the total amount of transmission hops between sensor and actuator.

Figure 4.19. Security - Delay time in star (a-b-c) and mesh (d) topologies.

From all the above presented results in terms of energy consumption, network traffic, and actuation delay, two main considerations can be carried out: (i) a mesh topology is preferable to star and tree topologies because distributing the service logic inside the WSN can result in lower actuation time values as well as lower values for network traffic and energy consumption, (ii) though such a trend is true in general, it must be taken into account that a careful deployment planning in both node position and application logic installation is necessary. In the respect of this last point PyFUNS can be considered a valuable framework to be used by integrating managing resources in future versions.

Light Control service

The Light Control service, instead, is an interesting scenario to evaluate PyFUNS performance in case a service is composed of different applications. In this scenario the whole set
of topologies analyzed in the Security scenario have been again considered (Figure 4.14). All networks are composed of: a Border Router (node 1), two PIR sensors (node 2 and node 3), one light sensor device (node 4), one lamp (node 5). All the other nodes of the network are simple 6LoWPAN routers without any sensor or actuator. Since both applications can be installed on any WSN node, a subset of deployment configurations have been selected according to the following rules: (i) both applications on installed on the BR, (ii) both applications on the light sensor, (iii) both applications on the lamp device, (iv) both applications on the node with the minimum energy consumption measured in the Security service, and (v) Motion Detection application always on the node with the minimum energy consumption measured in Security service while moving the Lamp Control application on BR, PIR sensor (node 2), light sensor and lamp device. Table 4.7 summarizes all the selected configurations by dividing them for topologies.

Table 4.7. Light Control service deployment configurations.

<table>
<thead>
<tr>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(e)</th>
<th>(f)</th>
<th>(g)</th>
<th>(h)</th>
<th>(d)</th>
<th>(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR - BR</td>
<td>N8 - BR</td>
<td>Light - BR</td>
<td>Lamp - PIR2</td>
<td>BR - BR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR - PIR2</td>
<td>N8 - PIR2</td>
<td>Light - PIR2</td>
<td>Lamp - PIR2</td>
<td>BR - BR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR - Light</td>
<td>N8 - Light</td>
<td>Light - Light</td>
<td>Lamp - Lamp</td>
<td>BR - BR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR - Lamp</td>
<td>N8 - Lamp</td>
<td>Light - Lamp</td>
<td>Lamp - Lamp</td>
<td>BR - BR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Light - Light</td>
<td>N8 - N8</td>
<td>Lamp - Lamp</td>
<td>Lamp - Lamp</td>
<td>BR - BR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this second scenario all results are presented only in terms of energy consumption for the sake of brevity. Figure 4.21 shows the results for the three star topologies (a-b-c). As showed in the Security service scenario the total energy consumed increase according to the number of transmission hops. Regarding the simulation with topology (a) and increased transmission power, the test confirms that the consumption increase with the number of nodes in the network,
and that it is better to expand the network coverage by increasing the transmission power instead of adding additional nodes (red and yellow columns). Interesting considerations can be carried out from the evaluation of Light Control performed in (e) (f) (g) topologies (Figure 4.22). The results show that by distributing the logic in the WSN network lower energy consumption than centralized approaches can be experienced, even though this is not true for all configurations. Indeed, using N8 - BR and N8 - PIR2 configurations a higher energy consumption is experienced in respect to BR - BR for (e) and (f) topologies. However, when Lamp Control and Detection Motion applications run on the same node where Security application showed the minimum energy consumption, in this scenario such a value shows a minimum, and this is true for all three considered topologies, having 13.29 mJ, 12.58 mJ and 9.13 mJ for (e), (f) and (g) respectively. Same results are obtained in topology (h), as it is depicted in Figure 4.23 where both applications are installed in the light node instead of N8. In this latter case the best the total energy consumed by all nodes in the network is equal to 7.27 mJ. In topology (i) the same behavior has been experienced (Figure 4.24), even though in this case both applications are installed in the lamp node. Among all tree topologies the one that shows the lowest energy consumption is the (h), because it has a bigger number of links with respect to all the others, which results in the possibility of performing communications among nodes with a shorter path. In the mesh topology the minimum energy consumption is reached in any case by installing both applications in the same node (see Figure 4.24), while the lowest value can be noticed in the case in which they are installed in the light node.

4.5. PyFUNS extension

The use of PyFUNS simplifies reprogramming of WSN has been described. The proposed solution hides communication details and does not require developers to be experienced with embedded programming. PyFUNS is a framework that allows to deploy dynamically distributed services, however it does not preclude centralized approaches, such as running the entire application logic in external servers or in the Cloud. Indeed, PyFUNS can be used as a complementary tool of a bigger framework that allows to the user easy writing of python scripts (e.g., through a
Graphical User Interface), and easy installation/update of services on a wireless sensor network,
hiding from the user the installation process. This feature can be addressed by PyoT, a system for macro-programming and managing IoT-based WSNs.

PyoT abstracts the WSN as a set of software objects that can be manipulated and combined in order to perform complex tasks. Specifically, PyoT allows the user to: (i) automatically discover available resources; (ii) monitor sensor data; (iii) handle its storage; (iv) control actuators; (v) define events and the actions to be performed when they are detected; and (vi) interact with resources using a scripting language (macro-programming).

The high-level abstraction provided by PyoT completely hides the nodes and the network, letting users and application developers focus on the sensing and actuation capability of the system. Moreover, this approach allows for a seamless federation of multiple WSNs as long as such WSNs are based on the IoT protocols (i.e., 6LoWPAN and CoAP). Indeed, integrating a new WSN into PyoT just means adding its resources to the set of available ones.

The development of applications involving large groups of nodes is further supported by the distributed and scalable architecture of PyoT, which, for example, allows application developers to parallelize operations on sets of nodes. Moreover, PyoT can integrate reprogramming frameworks that enable “in-network processing” in IoT-based WSNs, i.e., the possibility to run part of the application logic directly on sensor/actuator nodes. PyoT already provides such a kind of feature, integrating T-Res, however it can be easily extended adding the management of the PyFUNS application by using the application RESTful interface presented in Section 4.3.5.

PyoT also provides a storage mechanism that is used for both caching purposes and long term storage. Sensor data caching is performed automatically by PyoT in order to reduce the number of network interactions with IoT nodes thus extending their life (nodes are usually battery-powered and network communication is their main source of energy consumption) and improving application performance (radio duty-cycling policies commonly used by IoT nodes lead to slow network communication).

Finally, PyoT does not define a custom language for its macro-programming mechanism; on the contrary, it uses a popular high-level programming language (i.e., Python) that features a large set of libraries, including scientific ones (e.g., SciPy, NumPy, matplotlib, pandas). The use of a high-level language makes it possible to quickly prototype complex applications in a few lines of code. Moreover, PyoT macro-programming mechanism is not just targeted to application development, indeed, it can also be used to control IoT nodes “on the fly” by means of interactive shells.

Other important feature of PyFUNS is decoupling of IoT services from the sensing infrastructure, allowing deployment of application tasks at any network node. This possibility enables an infinite number of application deployment configurations. Next step can be the study of application deployment, performed both locally (i.e., by the application itself) and remotely (i.e., by an external frameworks). Locally means that the application itself has the knowledge of parameters that describe the state of the node in which is installed (e.g., remaining battery lifetime, local network congestion) and take management decisions on them (i.e., migration to another node of the network, put itself in a sleep state). To enable this feature, a remote framework that has the knowledge of the whole network is required. It can manage the applications deployment
by computing the optimal configuration, in terms of energy consumption and actuation delay, and set the network configuration.

4.6. Conclusions

As wireless sensor networks moved from the academic world to the industrial scenario new challenges have been raised up in order to reach a wide adoption of the WSNs in several domains. Some of the main issues nowadays considered are: interoperability, ease of reprogramming and reliability. To address such issues, PyFUNS, a Python framework for ubiquitous sensor networks, is presented. By leveraging on IoT-based protocols (i.e., 6LoWPAN and CoAP) PyFUNS guarantees a higher interoperability and reliability with respect to old-style WSNs. Moreover, thanks to its adopted virtual machine based design based on PyMite, a reduced Python virtual machine for embedded systems, PyFUNS enables easy reprogramming capabilities in wireless sensor network.

PyFUNS is first presented by detailing its design and implementation choices by carefully explaining its usage in building simple and complex services. Two detailed examples are described, labeled as: Security service and Light Control service. Then, it has been evaluate PyFUNS performance considering the WiSMote and CC2538dk platforms with the aim of proving its feasibility in real constrained devices, and its overhead in terms of run time and energy consumption with respect to native code solutions. Finally PyFUNS performance in star, mesh and tree network topologies are evaluated for both Security and Light Control services by considering both centralized and distributed application logic solutions.
CHAPTER 5

Application scenario

In the last several years a large number of projects have been realized with the aim of creating effective Internet of Things deployments in which useful and cost-effective services can be provided to the final users. As matter of example it is possible to cite the already mentioned GINSENG \[OBB^{13}\] and SmartSantander \[SMG^{14}\] projects where the potential of Wireless Sensor Networks have been proved through real large-scale deployments. In the envisioned Smart City scenario, one of the key components are distributed smart sensors which are able to interact with the physical world exchanging data through wireless communications. In this direction the European Union (EU) has progressively increased its contribution in funding research activities in the Intelligent Transport Systems field aiming at: (i) making optimal use of road, traffic and travel data; (ii) guaranteeing continuity of traffic and freight management in ITS services, and (iii) creating ITS road safety and security applications. Such requirements have been specified by the EU in the Directive 2010/40/EU \[EUd^{10}\].

In order to create effective ITS solutions able to meet the EU requirements it is necessary to develop layered architectures \[PPAS^{13}\] where a data collection infrastructure is able to collect data pervasively in the monitored area. Although the pervasiveness is a key point for ITS collection layers, such a requirement can be reached in reality only by using low-cost hardware solutions able to communicate among them wireless through standard protocols. Nowadays the low-cost condition can be reached by using embedded devices based on low-complexity microcontroller, while providing them advanced communication capabilities able to support well-known Internet based protocols. In this direction the most suitable technology solution is based on Wireless Camera Network based on communication standards recently developed for supporting the IoT paradigm in resource constrained devices used in WSNs. The use of low-complexity devices coming from IoT-based WSNs and equipped with vision capabilities (i.e., IoT-based WCNs) on the one hand requires the development of advanced low-complexity computer vision algorithms able to extract mobility related data (e.g., traffic flow, parking lots occupancy levels), while on the other hand strongly reduces installation costs in the ITS scenario. Camera Nodes can be installed on existing poles, thus avoiding expensive infrastructure works as required by state-of-the-art solutions based on invasive installations (e.g., inductive loops). Moreover, the use of standard protocols derived from the Internet world, and adapted to the WCN scenario, enable the system pervasiveness as well as the possibility of creating open and interoperable ITS systems acting as part of a bigger IoT network. By merging together advanced solutions in WCN and IoT research fields innovative cost-effective and optimized collection layers for ITSs can be
created. As matter of example, each node of the data collection infrastructure can extract mobility related data performing an on-board processing of captured images with the aim of exposing them as resources to be used by other nodes or high level entities to generate complex events (e.g., traffic queue detection by merging features of several nodes) or to feed running applications (e.g., traffic prediction services by sending raw resources to a control center acting in the cloud).

According to the above described vision, the intelligence of the whole system is distributed among all the nodes of a global network, inside the WCN acting with IoT protocols, and outside, in remote high-end devices able to elaborate the resources provided by the CNs. Inside the IoT-based WCN the resources exposed by the nodes can be both scalar and vector features obtained by an on-board processing of the captured images. In this way each visual sensor provides features which reflect its environmental understanding related to its field of view. Whenever an in-network processing of such features is necessary by performing their composition, elaboration, and aggregation, this operation can be delegated to a middleware solution able to: (i) manage network transactions among nodes; (ii) manage node functionality by interacting with the exposed operating system interfaces; and (iii) provide enough flexibility to change at run-time the internal Resource Processing Engine (RPE), thus without changing the node firmware, and enabling a reconfigurable in-network processing in visual sensor networks.

A pictorial view of an IoT-based WCN exposing mobility data as resources is depicted in Figure 5.1. In the picture three nodes are monitoring the same parking spaces. While two nodes are considering the highlighted parking space full with a probability equal to 50% and 40% because a car passing in their field of views and in front of the space, the other node is exposing an occupancy level resource suggesting that the monitored parking space is empty. All three resources can be aggregated in any node of the network where a RPE gathering all the occupancy levels, and other possible information, can take the right decision.

![Figure 5.1. Wireless Camera Network exposing parking space occupancy levels as resources.](image)
5.1. INTELLIGENT TRANSPORT SYSTEM AND INTERNET OF THINGS

Contributions The rest of the chapter discusses the integration of two application domains that are especially challenging: Wireless Camera Network and Intelligent Transport System. This chapter first introduces WCNs, by focusing on already developed applications targeted to the ITS scenario, then it describes IoT enabling protocol solutions and the REpresentational State Transfer (REST) paradigm on which IoT-based WSNs, as well as IoT-based WCNs are designed (Section 5.1). The architecture of the proposed middleware able to support reconfigurable in-network processing in IoT-based WCNs is presented in Section 5.2, while in Section 5.3 the middleware working principles are detailed for the “parking lot monitoring” use case by considering node resources in an IoT network, and a possible RPE module. The feasibility of the proposed solution in low-cost WCN devices based on microcontroller is proven through a real implementation of the middleware in the Contiki OS [DGV04] running in the Seed-Eye board [SEE]. In all the following sections the terms IoT-based WSNs and IoT-based WCNs are used with the same meaning by considering a WCN as a WSN in which nodes embed vision capabilities.

5.1. Intelligent Transport System and Internet of Things

First part of this section introduces WCNs by presenting state-of-the-art already developed applications targeted to the ITS scenario. In the second part the use of the REST paradigm in IoT-based WCN is discussed.

5.1.1. Wireless Camera Network Applications in the ITS scenario

Although WSNs have been already proven to be an effective tool for supporting next generation ITS, WCNs capabilities in such a scenario have not been fully exploited. Indeed, even if several real deployments of sensor networks based on scalar sensors (e.g., acoustic sensors, ultrasonic sensors, magnetometers) can be found in literature (e.g., VTrack [TRL+09], ParkNet [MJK+10], WITS [CCCT06] projects.), WCNs test beds are not much common. A recent example in this direction is the test bed developed during the Ipermob project 1, where a real WCN has been installed at the Pisa International Airport landside to monitor and control urban mobility in real-time. The main difficulty in developing real WCNs targeted to ITS is in adapting already developed computer vision applications to the new resource-constrained scenario. Indeed, the use of low-cost hardware, having limited computational capabilities and onboard memory, makes unfeasible the use of state-of-the-art computer vision applications, which has to be redesigned or modified for the new application scenario, as proposed by [SPdR+14].

In recent years, several ITS-related applications based on embedded CNs have been proposed. As a matter of example it is possible to cite [MMN+11,SPGP12b], where two embedded low-complexity computer vision applications have been developed with the aim of detecting the status of parking spaces, and [MSP+12,SPB+14], where CNs are used to count passing vehicles as well as to measure car speed. Looking forward at future applications, the use of pervasive WCNs will allow the realization of a set of interoperable systems capable to solve several open issues

1-Infrastruttura Pervasiva Eterogenea Real-time per il controllo della Mobilità (“A Pervasive and Heterogeneous Infrastructure to control Urban Mobility in Real-Time”). [http://www.ipermob.org]
in the ITS field. While on one hand a pervasive deployment of CNs able to extract mobility related parameters will be able to generate open data to be consumed by municipalities for traffic planning purposes, on the other hand they can retrofit old road illumination systems for creating advanced services in the so called Smart Cities. Moreover, in the same direction, more complex and distributed applications can be deployed exploiting the computational power of the above mentioned FPGA based CNs and the flexibility of the wireless networks: a real example is the tracking of pedestrians while they are crossing a road, thus improving safety in the road of the future.

5.1.2. REST paradigm in IoT-based WSNs

IoT-based communication rely on standard protocol solutions covering all the layers of the well-known Internet Protocol suite. The standard protocol, specifying both the Physical (PHY) and Medium Access Control (MAC) sub-layers of the ISO/OSI communication model, is the IEEE 802.15.4 [Soc11]. The standard has been released in its first version in the 2003 with the aim of enabling energy-efficiency communications in Low-Rate Wireless Personal Area Networks (LR-WPANs). In the 2007 Kushalnagar et al. [KKGB12] proposed the adaptation of the 6LoWPAN, thus specifying a Network (NET) layer for enabling Internet like communication in IEEE 802.15.4-based networks. The 6LoWPAN concept comes from the idea that ”the Internet Protocol could and should be applied even to the smallest devices” [Mul07], and that low-power devices with limited processing capabilities should be able to participate in the envisioned IoT. 6LoWPAN defines the frame format for the transmission of IPv6 packets, as well as mechanisms for header compression, and formalizes how to create IPv6 global addresses on top of IEEE 802.15.4 networks. The last protocol to be cited for enabling IoT solutions is the Constrained Application Protocol (CoAP) [SHB14], which is a standard solution working at the Application (APP) layer, and currently being defined within the CoRE working group of the IETF. It aims at providing a REST-based framework for resource-oriented applications by designing a protocol stack able to cope with limited packet size, low-energy devices and unreliable channels. CoAP is designed for having an easy stateless mapping with HTTP, and for providing Machine-2-Machine interaction. HTTP compatibility is obtained by maintaining the same interaction model, but using a subset of the HTTP methods. Any HTTP client or server can interoperate with CoAP-ready endpoints by simply installing a translation proxy between the two devices. The REST paradigm in IoT-based WSNs will be detailed in the following, while in Figure 5.2 the protocol stack for the IoT is reported, by comparing it with the classical Internet protocol suite.

REpresentational State Transfer (REST) [Fie00] is an architectural style for distributed systems introduced and defined in 2000 by Fielding in his doctoral dissertation. RESTful architectures basically consist of clients and servers. Clients send requests to servers, which reply with appropriate responses. Requests and responses are built around the transfer of representations of resources, where a resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of the resource. The most relevant example of a system conforming to the REST
architectural style is the World Wide Web, in which resources are manipulated using the HTTP protocol.

The REST paradigm can be successfully used in IoT-based WSNs, where resources usually represent sensors, actuators or other possible information. However, as previously introduced, the CoAP protocol is used instead of HTTP, thus allowing sensor nodes to run embedded web services through which their resources can be manipulated. Specifically, CoAP provides four methods for manipulating resources: (i) PUT, which requests that the resource identified by the URI specified in the request be updated or created with the transmitted representation; (ii) POST, which requests that the representation transmitted in the request be processed; (iii) GET, which retrieves a representation of the resource identified by the URI specified in the request; and (iv) DELETE, which requests the deletion of the resource identified by the URI specified in the request. CoAP also provides a resource observation mechanism [Har14] (OBSERVE) which allows a node to receive notifications about changes in resources it has previously subscribed to.

Figure 5.3 depicts a simple example of an IoT-based WSN with vision capabilities able to act as an ITS collection layer. In the picture the network is composed by two nodes: one is
a vision sensor able to count vehicles and detect possible traffic congestion by analyzing the speed of vehicles (on the left), while the other one is an actuator able to control the messages of a Variable Message Sign (VMS). According to the REST working principles each resource is identified by the node IPv6 address and a symbolic name, and is managed through PUT, POST, GET and DELETE methods. For example, the CAM node can expose a “queue” resource that can be retrieved by issuing a GET request on the URI \texttt{coap://[aaaa::1]/trafficdata/queue}. Similarly the VMS exposes a “message” resource that can be controlled by sending a PUT request to the URI \texttt{coap://[aaaa::2]/trafficdata/message} containing the message to be displayed in its payload. In such a way it is possible to create an application that collects traffic-congestion notifications (sending GET requests to the “queue” resource) and uses them for suggesting alternative routes to drivers (sending PUT requests to the “message” resource).

The use of standard protocols (i.e., 6LoWPAN and CoAP) allows the nodes to be used for many different applications. However, such protocols alone do not provide a way for changing the devices application logic once nodes are installed and configured for a specific task. As previously stated, such a feature must be demanded to a middleware solution running in visual sensors and able to provide enough flexibility in changing at run-time the internal RPE, as well as to manage network transactions and resources (i.e., through exposed program interfaces the middleware must be able to get data from running computer vision algorithms in order to update the exposed resources).

5.2. Proposed Middleware Architecture for IoT-based WCNs

A middleware is basically a software solution able to interact with both high level applications and Operating Systems (OSs). Its main purpose is to uniform heterogeneous systems by hiding their specific complexity and by providing a common unified software abstraction. Moreover, middleware usually provide common configuration and maintenance services, thus enabling an easier management of complex systems. In a WSN scenario a middleware is usually designed as a tool to bridge the gap between the high level requirements of the applications and the low level hardware complexity. WSN middleware are supposed to help programmers by providing an adequate system abstraction, and allowing them to focus on the high level application logic without caring about low level implementation details.

In IoT-based WCNs in which low-complexity computer vision algorithms are executed on CNs with the aim of providing both scalar and vector resources (i.e., scalar and vector features extracted by processed images) to other network nodes, a middleware must be able to:

(i) manage network transactions among nodes by leveraging on IoT protocols while using the REST paradigm; (ii) manage CN functionality permitting to configure on-board computer vision algorithms through exposed operating system interfaces, and (iii) provide enough flexibility to change at run-time the CN RPE. The last requirement is a key point to consider to permit the composition, elaboration, and aggregation of image-related resources, and to enable a reconfigurable in-network processing in WCNs. Such middleware high level requirements can be easily mapped onto middleware components able to interact among them through common interfaces. At a system design level three main middleware components can be identified:
5.2. PROPOSED MIDDLEWARE ARCHITECTURE FOR IOT-BASED WCNS

(1) RESTful Web Service (RWS);
(2) Configuration Manager (CM);
(3) Resource Processing Engine (RPE).

The three components are graphically reported in Figure 5.4 while their functionality are detailed in the following of the section. The first two are identified as OS components, this is because they mainly interact with OS interfaces to configure the embedded vision logic as well as to use network services to support the REST paradigm. The last is identified as Virtual Machine component, because, as it will be better clarified in the following, it requires to be implemented as an application on top of an intermediate software layer running in the operating system. According to the classification presented in [HM06] this approach is considered a Virtual Machine based design.

![Figure 5.4. Middleware components.](image)

5.2.1. RESTful Web Service

The RESTful Web Service component handles all network data inputs and outputs through CoAP transactions, moreover it acts as resource directory service. In fact, each resource exposed by a certain node to other entities of the network is internally registered by such component through RESTful program interfaces. In addition the component has the knowledge of the internal exposed resources, both simple (e.g., the output of an on-board running algorithm) and complex (e.g., the result of a resource aggregation process made by the RPE). In IoT-based WCNs the internal resource exposed by the node through the RWS component can be both scalar and vector features extracted by processed images. As depicted in Figure 5.4 the component mainly interacts with the OS network communication stack, and also with the other two components with common Application Program Interfaces (APIs). In fact, along the services provided to the RPE module to gather and publish resources, it can manage network transaction in behalf of the CM component, while exposing again as resources the CM configuration parameters.
5.2.2. Configuration Manager

The Configuration Manager component is an OS-based application running on top of the OS. The component is responsible of the configuration of some part of the platform. It is in charge of the RPE configuration by changing upon request the RPE logic for the resource elaboration. In IoT-based WCNs a main feature requested to such a component is the possibility of changing configuration parameters for the running computer vision applications. In CNs based on embedded processors (e.g., WiCa, MeshEye, Seed-Eye) simple parameters can be easily configured at run-time. For instance, the resolution of the acquired image, the frame rate, the region of interests to be used by the on-board computer vision applications. In such a category of CNs the CM can be even used to change the running computer vision algorithms, even if this would require firmware updating policies. In CN based on FPGA and internal programmable logic (e.g., FPGA-based CN presented in the previous section) the CM can be used to configure and compose the whole computer vision pipeline, thus changing at run-time the whole on-board processing application. Moreover, because of the interaction with the RWS component the output of any single selected step of a computer vision pipeline can be abstracted as a resource, thus enabling the possibility of having multiple distributed applications inside the network (i.e., the output of a step can be used by the RPE of a node for a certain application, while another node can use the same value for further elaborations).

5.2.3. Resource Processing Engine

The RPE component is the core technical module of the middleware. To the component is requested to: (i) monitor one or more resources; (ii) execute some data processing on their values; and (iii) send the output to other resources. To gather and publish resource values RPE strongly interacts with WBS, while interacts with CM for changing the RPE processing logic. The RPE of the proposed middleware architecture for IoT-based WCNs is mainly based on a Reprogramming Framework (RF) that abstract the tasks performed by a device as a resource. Examples of such a framework can be PyFUNS, presented in Chapter 4, or T-Res [AP13] where the main idea beyond them is to represent resource processing tasks as CoAP resources. In such a way a processing task resource, like any other classic resource, can be manipulated using CoAP methods: it can be created, deleted, modified, or even retrieved (duplicated).

To fully decouple task processing functions from the sensing infrastructure, as well as to permit to change them through another entity (e.g., CM component) without requiring all the firmware updates, the processing functions must be platform independent. Therefore, they cannot be written in languages that must be compiled into native code (e.g., C, C++), instead, languages that can be compiled into bytecode or directly executed by an interpreter must be used. In this direction PyFUNS and T-Res use Python for defining processing functions, while providing APIs that can be used in Python scripts to define processing functions. Because they uses Python bytecode running on a Python interpreter hosted by the operating system they are considered a virtual machine component, and consequently the RPE component.
5.3. Middleware Instantiation for the Parking Lot Monitoring Use Case

In this section the middleware working principles are detailed for a “parking lot monitoring” use case by considering CN resources for each middleware component, and by detailing the Python code performing resource processing in RPEs. Moreover, the feasibility of the proposed middleware solution in IoT-based WCNs in proven through a real implementation on the Seed-Eye board.

5.3.1. Use case scenario, exposed resources and their interaction

The considered use case is that of an ITS collection layer based on an IoT-based WCN in which CNs are deployed on the field to evaluate the status of several parking spaces. By referring to Figure 5.1 it is possible to consider to have three nodes monitoring (for sake of simplicity) a total amount of four parking spaces which identification numbers are 23, 25, 26 and 27. Because of their deployment in the field, all three nodes have in their field of view the parking space 25. Moreover, each CN runs an on-board computer vision algorithm giving as output the occupancy level for each parking space as a value ranging from 0 to 255. Such a value intrinsically reflects the uncertainty of the decision regarding the state of the parking space. Indeed, considering values in the range [0, 128) associated to an empty state, values close to 128 can indicate a bigger uncertainty. The same consideration is valid for a full state in which the occupancy level is in the range [128, 255]. During the installation phase each CN can be remotely configured through a possible gateway entity to set up, for each parking space, a Region Of Interest (ROI) to be considered by the algorithm, as well as a fixed weight which is a value a used by the algorithm to evaluate the occupancy level. For instance, during deployment the field of view of a camera can partially frame a parking space, in this case the algorithm can run on the configured ROI by using a low weight, thus meaning a bigger uncertainty associated to the occupancy level.

Considering of having the middleware previously described running in each CN, the resources exposed for each component are:

- Parking resources through RWS component
  - /space_xx_level;
- CM component
  - /space_xx_roi;
  - /space_xx_weight;
- RPE component
  - /apps/parking and its sub-resources.

The exposed resources are detailed for the three nodes in Figure 5.5 along with a possible interaction model performing an in-network processing to take a global decision on the parking space 25.

As it is possible to see in the picture Node 1 is exposing Parking resources for parking spaces 25 and 27, as well as CM resources related to them. In the example such a node is not exposing RPE resources because it does not provide any processing on them. The same is for Node 3 which only exposes Parking and CM resources for the parking spaces 25 and 26. Node 2, instead, exposes, along with Parking and CM resources, also RPE related resources. Indeed,
Node 2 is performing a processing task on resources related to the parking space 25. The Node 2 RPE periodically requires as input to GET all `space_25_level` resources. When all the values have been collected the processing task is activated and the jointly decision on the occupancy level of parking space 25 (e.g., through an average evaluation of acquired values) is sent to the Gateway for a possible communication in a parking control center. Following the RPE input resources to perform the processing:
- `coap://[aaaa::1]/space_25_level`
- `coap://[aaaa::2]/space_25_level`
- `coap://[aaaa::3]/space_25_level`
while the output resource:
- `coap://[aaaa::100]/parking/space_25`

The Python pseudo-code that generates a possible bytecode installed on the RPE component are shown in Figure 5.6. Where the PreScript (Figure 5.6a) retrieve the value of each resource defined as input, and MainScript (Figure 5.6b) performs a PUT on the output resources. In a real scenario the bytecode of the Python script must be transferred to the CN through a Programming module running on an external device and able to interact with the CM component of the middleware. In the presented use case the Programming module can easily run on the Gateway and generates the bytecode to be transferred to the node once a new Python script is available. Moreover, in more powerful scenario, it can even generate scripts to distribute the complexity of the processing among several nodes.

5.3.2. Middleware implementation

To prove the feasibility of the proposed middleware solution its requirements have been evaluated in both code size (flash memory occupancy) and memory requirements (RAM occupancy) by implementing its components on Contiki OS, and by considering the Seed-Eye board as target device. As first step PyFUNS has been ported on the Seed-Eye, thus using again PyMite as Python interpreter. PyMite is a reduced Python virtual machine suitable for embedded systems. At a second stage a basic CM component with reduced functionality has
from pyfuns import *

sendMsg(1, [0,0,0,1], "sen/space_25_level", ",", "space25_1")
sendMsg(1, [0,0,0,2], "sen/space_25_level", ",", "space25_2")
sendMsg(1, [0,0,0,3], "sen/space_25_level", ",", "space25_3")

(a) PreScript.

from pyfuns import *

sum = getVar("space25_1")
sum += getVar("space25_2")
sum += getVar("space25_3")

avg = sum / 3

if avg < 128 :
    sendMsg(3, [0,0,0,100], "parking/space_25", "0")
else :
    sendMsg(3, [0,0,0,100], "parking/space_25", "1")

(b) MainScript.

Figure 5.6. Parking slot application: PreScript and MainScript.

been implemented, while RWS is intrinsically furnished by Contiki OS thanks to its internal 6LoWPAN and CoAP support.

The resulting implementation on the already presented Seed-Eye platform has been tested, which is equipped with a PIC32MX795F512L microcontroller having 128kB of RAM and 512kB of flash memory. Table 5.1 shows the RAM and flash memory occupancy by reporting in percentage their impact on the total microcontroller resources. The proposed middleware is a real feasible solution for low-complexity and low-cost CNs. In the Seed-Eye board it requires only the 14% of the available RAM and the 31% of the available flash memory. Because of the limited amount of occupied RAM, more processing tasks (Python bytecode) can be instantiated in the RPE component, thus enabling much more complex in-network processing functionality. As matter of example a node can host several resource composition logics to evaluate possible better composition strategies. Moreover, thanks to the reduced amount of flash memory occupied in the tested implementation more complex CM components can be developed, thus fully supporting more flexible configuration policies required by next generation FPGA-based CNs.

5.4. Conclusions

IoT-based WCNs are a suitable solution for the development of low-cost, pervasive and interoperable ITS collection layers able to fully support cost-effective services to be provided
to final users. According to such a vision, CNs run on-board computer vision algorithms to extract mobility related features that can be represented as device and network resources thanks to the use of the REST paradigm enabled by IoT protocols. In IoT-based WCNs whenever an in-network processing of exposed resources is necessary by performing their composition, elaboration, and aggregation, a middleware can be instantiated to hide all the complexity as well as to abstract network functionality. This chapter presents a middleware solution suitable for IoT-based WCNs and able to: (i) manage network transactions among nodes by leveraging on IoT protocols while using the REST paradigm; (ii) manage CN functionality permitting to configure on-board computer vision algorithms through exposed operating system interfaces, and (iii) provide enough flexibility to change at run-time the internal resource processing engine by using a virtual machine based solution. In the chapter the middleware components are first presented before to detail the its working principles for a “parking lot monitoring” use case where several CNs monitor overlapped parking spaces. The feasibility of the proposed middleware is proven in the chapter by reporting objective data obtained through a real implementation based on the Contiki OS running in the Seed-Eye board.

<table>
<thead>
<tr>
<th>Device</th>
<th>RAM</th>
<th>Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[bytes]</td>
<td>[bytes]</td>
</tr>
<tr>
<td>Seed-Eye</td>
<td>17652 (14%)</td>
<td>154,984 (31%)</td>
</tr>
</tbody>
</table>
CHAPTER 6

Conclusions

This thesis discussed the evolution of Wireless Sensor Networks into the IoT world by addressing several topics. Starting from the WSNs’ limitations at the beginning of the IoT era, the thesis pointed out the main challenges and the advances of the current systems that must be taken into account in next generation WSN in order to exploit WSN technology in our daily life. The main changings treated in this thesis are listed as following: (i) Protocol adaptation, (ii) Combine IoT protocols with new technologies, (iii) In-network processing, and (iv) (Ease of) Reprogramming.

Regarding Protocol adaptation, namely the adaptation of protocols designed for traditional WSN to WSN adopting the new IoT standard protocols, this thesis discussed the adaptation of Speed geographic routing protocol algorithm in a 6LoWPAN scenario. Two new adaptations have been presented: Speed-6LoWPAN and Speed-3D. Supporting soft real-time, load balancing and flow shaping mechanisms, they are effective solutions in supporting packet routing in 6LoWPAN networks.

In the topic of Combine IoT protocols with new technologies, this thesis discussed the integration of two main technologies in the IoT context: image sensing in WSN and Radio-Frequency IDentification. These integration open WSN world to interesting applications scenarios that can change our daily life. As a matter of example video streaming or object tracking can be low-cost and deployed in a pervasive manner. Regarding image sensing, an innovative approach for a streaming technique fitting the usual constraints of a WCN, and an innovative idea about in-node compression and inter-frame removal of redundant information is presented in this thesis. The approach presented is a possible candidate for streaming applications in WSN based on 6LoWPAN in which video streams originated by multimedia sensors can be consumed by high-end devices in the spirit of the Internet of Things. In the RFID field, an integration between WSN and RFID technologies in the IoT scenario is presented. The integration has been analyzed in respect of IoT enabling solutions, by proposing a representation of RFID related data as network resources through the REST paradigm. Image sensing and RFID technologies have been mapped in the 6LoWPAN architecture. The result is a final integrated network in which the outstanding pervasiveness of RFIDs and cameras merged together the advanced sensing and communication features of IoT networks.

PyFUNS is a Python framework for ubiquitous sensor networks with the aim to enables In-network processing and (Ease of) Reprogramming in the IoT domain. The use of in-network processing in WSNs is an important step to be accomplished, in fact it allows to reduce network traffic and energy consumption, moreover it improves the reaction time and the scalability of
the systems. Reprogramming is another requirement to be taken into account in order to enable the possibility of change remotely the services provided by a node during the WSN life cycle. Moreover, in order to open WSN technology to a wider audience, the reprogramming task must be as simple as much as possible. With this aim, this thesis presented PyFUNS: it enables reconfigurable in-network processing since it leverages on the resource abstraction provided by CoAP to represent the tasks performed by IoT nodes and simplifies the development of in-network applications by handling low level and networking functionality.

The thesis also investigated the possible use of IoT protocols in one of the most promising use cases of the smart cities scenario: Intelligent Transport System. The use case proves that the use of IoT abstractions is not confined to IoT-based WSNs, but can also be beneficial in other systems that likewise employ more powerful sensing, i.e. image sensing. Indeed, the parking slot scenario highlights that IoT-based applied to Wireless Camera Networks is a suitable solution for the development of low-cost, pervasive and interoperable ITS collection layers able to fully support cost-effective services to be provided to final users.

Generally speaking, IoT protocols and abstractions introduce the concept of generality, pervasiveness, hardware-independence and interoperability in the WSNs world, making the WSN technology a perfect candidate to a wide range of application scenarios. Therefore, a sub-optimal efficiency is one of the main drawbacks of using IoT protocols. Indeed, energy consumption, task execution time, congestion and saturation of the available bandwidth are the main disadvantages in the IoT-based WSNs. Despite the limitations considered, the IoT vision announces a technological revolution representing the future of the Internet.
Bibliography


