UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXIX Ciclo

Architecture and Technologies for the Internet of Things

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Luca Veltri

Dottorando: *Luca Davoli*

Dicembre 2016

To My Family

Contents

List of Figures

List of Tables

List of Acronyms

6LBR	6LoWPAN Border Router
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Network
8PSK	Eight Phase-Shift Keying
AAA	Accounting/Authentication/Authorization
AAI	Authentication and Authorization Infrastructure
ACL	Access Control List
AES	Advanced Encryption Standard
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BLE	Bluetooth Low Energy
BPSK	Binary Phase-Shift Keying
BR	Border Router
C2C	CoAP-to-CoAP
C&C	Command-and-Control
CoAP	Constrained Application Protocol

CoSIP	Constrained Session Initiation Protocol
CSPF	Constrained Shortest Path First
DDoS	Distributed Denial of Service
DGT	Distributed Geographic Table
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Table
DL-SID	Direct-Link Segment Identifier
DLS	Distributed Location Service
DNS	Domain Name System
DoS	Denial of Service
DRD	Distributed Resource Directory
DTLS	Datagram Transport Layer Security
DW	Data Warehouse
ECMP	Equal Cost Multi Path
FQDN	Fully-Qualified Domain Name
H2C	HTTP-to-CoAP
HATEOAS	Hypermedia as the Engine of Application State
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IaaS	Infrastructure-as-a-Service
ICN	Information-Centric Networking

IDaaS Identity-as-a-Service

IdM Identity Management

IdP Identity Provider

IGP Interior Gateway Protocol

IoT Internet of Things

IP Internet Protocol

IPSec IP Security

JSON JavaScript Object Notation

JSON-RPC JavaScript Object Notation Remote Procedure Call

JSON-WSP JavaScript Object Notation Web Service Protocol

LLN Low-Power and Lossy Network

LND Local Node Directory

LPWAN Low-Power Wide Area Network

LRD Local Resource Directory

LSP Label Switched Path

M2M Machine-to-Machine

MAC Medium Access Control

mDNS Multicast DNS

MPLS Multi Protocol Label Switching

MQTT Message Queuing Telemetry Transport

MTU Maximum Transmission Unit

NBI	NorthBound Interface
NFV	Network Function Virtualization
OF	OpenFlow
OFDM	Orthogonal Frequency-Division Multiplexing
OS	Operating System
OSPF	Open Shortest Path First
OTP	One-Time Password
OvS	Open vSwitch
P2P	Peer-to-Peer
PaaS	Platform-as-a-Service
PLC	Power Line Communication
PMSR	Poor Man's Segment Routing
PW	Pseudo-Wire
QoS	Quality of Service
QPSK	Quadrature Phase-Shift Keying
RD	Resource Directory
REST	Representational State Transfer
RFID	Radio Frequency Identifier
ROBO	Robust Modulation
RPC	Remote Procedure Call
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks

S/MIME Secure/Multipurpose Internet Mail Extensions

SaaS Software-as-a-Service

SAML Security Assertion Markup Language

SBC Single-Board Computer

SBI SouthBound Interface

SC Smart City

SCTP Stream Control Transmission Protocol

SDN Software-Defined Networking

SFC Service Function Chaining

SFTP SSH File Transfer Protocol

SG Smart Grid

SID Segment Identifier

SigComp Signaling Compression

SIP Session Initiation Protocol

SLA Service Level Agreement

SO Smart Object

SOA Service Oriented Architecture

SP Service Provider

SPF Shortest Path First

SR Segment Routing

SSH Secure Shell

SSO	Single Sign-On
TCP	Transmission Control Protocol
TE	Traffic Engineering
TLS	Transport Layer Security
TMR	Tone Map Request
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VLL	Virtual Leased Line
VoIP	Voice over IP
VPN	Virtual Private Network
WAYF	Where Are You From
WLAN	Wireless Local Area Network
WoT	Web of Things
WS	Web Service
WSDL	Web Service Description Language
WSN	Wireless Sensor Network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

Introduction

The IoT paradigm can been defined as a *network of networks* of devices interconnected in an Internet-like structure, generally denoted as Smart Objects (SOs), cooperating to collect data and provide services to users. SOs are extremely heterogeneous and differ for connectivity interfaces, battery, processing and memory capabilities, as well as dimensions, costs, and hardware features. They are typically equipped with a microcontroller, a radio interface for communication, sensors and/or actuators and, being typically battery-powered, there is a quest for energy-efficient technologies, communication/networking protocols and mechanisms. The Internet Protocol (IP) [1], and in particular IPv6 [2], have been widely envisaged as the true IoT enablers, as they allow to bring the full interoperability among heterogeneous objects. As part of the standardization process which is taking place, new low-power protocols are being defined in international organizations, such as the IETF [3] and the IEEE [4]. At the application layer, the Constrained Application Protocol (CoAP) [5] has been designed to bring the Representational State Transfer (REST) [6] paradigm, which was originally conceived for applications based on HyperText Transfer Protocol (HTTP) [7], to the IoT and is expected to become the standard communication protocol for constrained applications. Research has now gone beyond the hardware and protocols barriers, providing several solutions for building IoT networks and opening a new challenge: the definition of effective paradigms and mechanisms aimed at integrating the IoT in common people's life. This challenge is very complex from a communication perspective, as it involves all layers of the protocol stack. A few examples of aspects to deal with are: the choice of SOs connectivity; the mech-

anisms for automatic endpoints discovery; the resources representation; final users application design; and the models of interaction between SOs and people [8].

Related to the SOs connectivity, by definition, IoT concept encourage the reuse of existing communication paradigms, adding adaptation mechanisms, when needed. An example is represented by the adoption of the Power Line Communication (PLC) as wired access technology. Another interesting technology for IoT is Software-Defined Networking (SDN), that can help in configuring and managing the backhaul network connecting different access domains.

From the data point of view, an important aspect is the possibility to manage all sensed and gathered data in a smart and practical way. In this context, a possible solution is given by Big Data approaches, developed in the last few years and become popular due to the evolution of online and social/crowd services, which are able to address the need to process extremely large amounts of heterogeneous data for various purposes. However, these techniques typically have an intrinsic inertia (as they are based on batch processing) and focus on the data itself, rather than providing real-time processing and dispatching. For this reason, Big Data approaches might not be the right solution to manage the dynamicity of IoT scenarios with real-time processing. In order to better fit these requirements, it is needed to shift the Big Data paradigm to a new paradigm, denoted as "Big Stream", more oriented to the massive rate of data proper of IoT scenarios.

Thesis Structure

Chapter 1 presents the state of the art of architecture and technologies related to the IoT. Firstly, the main protocols and models related to the low-layer communications in IoT scenarios are presented, then protocols related to application layer are analyzed, in particular with reference to the most relevant existing IoT testbeds. Finally, the main data processing solutions (obtained from IoT scenarios) are analyzed, with particular attention to those characterized by real-time and low-latency constraints, and targeting security aspects of these solutions.

In Chapter 2, details on the introduction of the PLC paradigm as enabler for IoT-related data transmission are presented, thus performing experimental evaluations

of the IoT/PLC architecture. Then, the adoption of the SDN concept as technology for routing management of IoT networks is highlighted, further defining new SDN-related algorithms.

Chapter 3 introduces different enabling technologies aiming at increasing the purposes of the IoT-related application layer, thus validating the integration with experimental evaluations. In particular, the adoption of a constrained signaling protocol in IoT scenarios is detailed, as well as the definition of a Web-oriented testbed, composed of heterogeneous nodes and technologies, is provided, thus integrating it with different wireless communication technologies.

Chapter 4 focuses on the management of huge amounts of informations coming from IoT-oriented scenarios, and introduces a new concept denoted as Big Stream, thus presenting the architectural implementation with open-source technologies and providing performance results obtained through the deployment of a Big Stream-based platform. Moreover, security threats and issues related to the Big Stream concept, in conjunction with security-related technologies that can be adopted in the proposed Big Stream architecture, are described.

Chapter 1

State of the Art

1.1 Internet of Things

The huge number of heterogeneous SOs deployed in an IoT system allows the development of ubiquitous sensing in most areas of modern living. The outcome of this trend is the generation of a huge amount of data that should be treated, aggregated, processed, transformed, stored, and delivered to the final users of the system, in an effective and efficient way, by means of traditional commodity services.

Several architectures for IoT scenarios have been proposed in the literature. The European Union (EU), under its "7th Framework Program" (FP7, 2007–2013), has supported a significant number of IoT-related projects converging in the "IoT European Research Cluster" (IERC), and addressing relevant challenges, particularly from a Wireless Sensor Networks (WSNs) perspective. Among them, it is possible to recall "Internet of Things-Architecture" (IoT-A) [9] and SENSEI [10].

In the IoT-A project, developers focused their work on the design of an IoT architecture, aiming at connecting vertically closed applications, systems and architectures, in order to create integrated and open-interoperable environments and platforms. The main goal of the SENSEI project was to create a business-driven platform that addresses the scalability problems for an amount of globally distributed devices in Wireless Sensor and Actuator (WS&A) networks, enabling an accurate and reliable

interaction with the physical environment, and providing network and information management services.

Another EU FP7 project with relevant implications on the design of IoT platforms was "Connect All IP-based Smart Objects!" (CAL*IP*SO) [11]. The main goal of CAL*IP*SO was to enable IPv6-based connectivity of SOs to IoT networks with very low-power consumption, thus providing long lifetime and high interoperability, also integrating radio duty cycling and data-centric mechanisms with IPv6.

1.1.1 Protocols and Communication Models for IoT

It is a common assumption that, in the IoT, the most prominent driver to provide interoperability is IPv6. Referring to the IP stack, at the application layer developers find a variety of possible protocols applicable to different IoT scenarios, according to specific application requirements. Among many options, the following are the most relevant.

- HyperText Transfer Protocol (HTTP), is mainly used for the communication with the consumer's devices.
- Constrained Application Protocol (CoAP), is built on the top of UDP, follows a request/response paradigm, and is explicitly designed to work with a large number of constrained devices operating in Low-Power and Lossy Networks (LLNs).
- Extensible Messaging and Presence Protocol (XMPP), is based on XML, supports decentralization, security (e.g., TLS), and flexibility [12].
- Message Queuing Telemetry Transport (MQTT), is a lightweight protocol running on top of TCP/IP and following the publish/subscribe paradigm [13]. It is an attractive choice when a small code footprint is required and when remote sensors and control devices have to communicate through low bandwidth and unreliable/intermittent channels. It is characterized by an optimized information distribution to one or more receivers, following a multicast approach. Being based on a publish/subscribe communication paradigm, it acts through

a “message broker” element, responsible for dispatching messages to all topic-linked subscribers.

- Constrained Session Initiation Protocol (CoSIP), is a lightweight version of the Session Initiation Protocol (SIP) [14] aiming at allowing constrained devices to instantiate communication sessions in a standard fashion, optionally including a negotiation phase of some parameters, which will be used for all subsequent communications [15, 16].

1.2 Web of Things

One of the biggest hurdles of the IoT is the monolithic nature and fragmentation of existing vertical closed systems, architectures, and application areas. To foster IoT development and diffusion, applications are increasingly built around the well-known Web model, bringing about the so-called Web of Things (WoT). The Web-based approach helped to greatly expand the Internet, and will likely have the same effect on the IoT. As for IoT-oriented networks, WoT applications rely on specific Web-oriented application-layer protocols similar to HTTP and CoAP, and, more generally, to protocols complying with the REST architectural style.

Whereas simulation tools typically focus on evaluating low-layer communication protocols, in recent years several IoT testbeds have been deployed to evaluate IoT solutions in realistic smart environments, under real-world conditions.

The IoT-Lab [17] is an example of this kind of testbed environment, providing a very large-scale infrastructure with more than 2,700 wireless sensor nodes spread across six different sites in France, and is used to test protocols at the link and network layers and to collect performance results, such as energy consumption or packet delivery ratio. Nevertheless, additional efforts are needed to create new innovative services, promote long-term evolution of systems, and ensure the robustness of applications against changes that might occur over time — thereby furthering WoT development. To easily build applications for IoT and/or WoT scenarios, developers need the ability to work at a high level of abstraction and without worrying about low-level details.

Another relevant large-scale IoT testbed is SmartSantander, consisting of approximately 20,000 nodes deployed in different cities across Europe [18]. With its orientation toward Smart City (SC) services and applications, however, this testbed focuses on environmental data collection and is thus not set up to allow experimentation on a fully addressable and resource-oriented WoT. In particular, SmartSantander does not consider direct and bi-directional interactions between humans and objects.

1.3 Communication Technologies for the IoT

1.3.1 Wireless Technology in IoT Scenarios

The implementation of IoT systems has the double objective of lowering the entry barrier in connecting things to each other and to the Web, promoting a rapid deployment of Web-enabled objects and giving end-users simple methods by which to access things. To achieve this, one can leverage existing HTTP- and Wi-Fi-based infrastructures, following two different approaches for connecting things to the Web: (i) make a physical object smart by attaching an IoT node to it, thus taking advantage of its built-in sensors and actuators (e.g., a temperature sensor installed on a window); (ii) modify the physical object itself by electrically connecting an IoT node, thereby offering connectivity to actuators of the physical one (e.g., connect a lamp to a Wi-Fi node to create a smart lamp remotely manageable via HTTP). Moreover, research in the IoT field has been first focused on how to build IP-based architectures in WSNs, generally involving IP adaptations to enable resource-constrained things to be seamlessly connected to the Internet through lossy networks.

In the variety of IoT application scenarios, the spectrum of employed communication technologies is wide, with a clear tendency towards the use of wireless communication paradigms. The most representative short-range communication standards can be synthesized as follows:

- **IEEE 802.15.4** devices, normally adopt IPv6 addresses as they are generally employed in application scenarios with networks composed by a huge number of nodes, such as extensive industrial monitoring. For this reason, they need an

adaptation layer (e.g., IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN)) to be able to communicate maintaining low packet size, low-power consumption, and other optimizations related to the limited capabilities of this SOs family [19].

- **Bluetooth Low Energy (BLE)** devices, using the low-power version of the Bluetooth protocol, are one of the latest entries in the IoT arena, being generally deployed in personal area applications (i.e., for proximity sensing or beaconing) [20]. A huge advantage of these devices is the presence of Bluetooth interface on the majority of smartphones.

- **IEEE 802.11** devices, connected in Wireless Local Area Networks (WLANs), are widely used in several IoT testbeds for their easy integration with existing infrastructures and built-in IP network compatibility [21].

These short-range devices are generally organized in subnetworks assuming different topologies (i.e., star, tree-based, ring, mesh). Because of their resource constraints, they typically need to be connected to the rest of the IoT world through a more complex element which acts as a gateway, providing high level functionalities such as: data aggregation, automatic discovery, resource directory interaction, and others.

Considering the long-range communication technologies, there exist two main classes of approaches. The first one relies on the use of cellular networks (e.g., 3G/4G and upcoming 5G) which will likely play a fundamental role in new IoT systems, being able to provide ubiquitous connectivity in wide areas and allowing direct use of smartphones. However, pushing cellular connectivity into SOs presents several limitations, related to the enormous number of IoT SOs that could be simultaneously connected to a single cellular base station, thus compromising the overall system performance. The second one is represented by Low-Power Wide Area Networks (LPWANs), emerging technologies, exploiting Sub-GHz communication bands and guaranteeing long-range communication. LPWANs are a possible alternative to cellular networks to collect IoT data. Sub-GHz-based boards guarantee enhanced coverage: from hundreds of meters to a few Km in urban areas, up to tens of Km in open space. They can be organized in networks with star topologies, avoiding

multi-hop communications. The drawback of the long-range communication technologies is the low data rate, with respect to the short-range ones. Common communication technologies in LPWANs are the following.

- **DigiMesh**: this LPWANs communication technology relies on a proprietary routing protocol developed by Digi [22], that automatically creates a meshed network among all nodes, allowing them to be addressed in an easy and straightforward way. Digimesh-enabled nodes can act as forwarders as well as endpoints, thus allowing both point-to-point and multi-hop communications from source to destination.

- **LoRa**: this LPWANs communication technology has been patented and designed by Semtech Corporation [23]. While the PHY layer of LoRa is proprietary, the rest of the protocol stack, denoted as LoRaWAN, is kept open. LoRa-based networks typically follow a star-of-stars topology, where the endpoints are connected via a single-hop link to one or many gateways that, in turn, are connected to a common Network Server (NetServer) via standard IP protocols. LoRa gateways forward messages between endpoints and the central NetServer. Unlike cellular systems, LoRa endpoints are not required to be associated with a gateway to get access to the network, but only to the NetServer. Gateways thus act only as bridges and simply forward to their associated NetServer all successfully decoded messages sent by any endpoint, after adding some information regarding the quality of the reception.

- **SIGFOX**: is one of the first LPWANs technology proposed in 2009 for IoT scenarios [24]. SIGFOX stack protocol specifications are secret (basically there is no publicly available documentation), but SIGFOX-enabled gateways are claimed to be able to handle up to a million connected objects, with a coverage area of 30-50 Km in rural areas and 3-10 Km in urban areas.

In order to preserve energy and to guarantee long-range communications, SIGFOX devices have some limitations, namely: 96 bits as maximum message size and a maximum of 140 transmitted messages per day. This is also due to the European regulation governing the 868 MHz band, which impose a transmission duty cycle not

higher than 1%. Instead, the LoRaWAN architecture requires the presence of two distinct entities (LoRa gateway and server) that need to be separated and to cooperate through a backhaul. These components can be redundant, as in many IoT scenarios it is possible to define architecture in which the LoRaWAN GW and NetServer are centralized in a unique entity.

In recent years, the theme of integrating Wi-Fi technology in IoT scenarios has been widely analyzed, often in a comparative way with respect to other existing radio frequency techniques. In [25], the ZigBee protocol [26], widely used in WSNs, has been compared with Wi-Fi-based systems in IoT and Smart Grid (SG) scenarios. ZigBee is a low-power, low-rate and short-range technology that relies, at the bottom layers, on the IEEE 802.15.4 standard. The latter, however, suffers from some limitations, such as low data transmission rate. Wi-Fi-based WSNs, instead, provide some significant advantages: (i) high bandwidth, and support for real-time and low-delay communications; (ii) large coverage (e.g., 50 m range for ZigBee nodes, 100 m indoor to 300 m outdoor for Wi-Fi technology); (iii) robustness, with quick installation and reliable fault-recovery; (iv) cost-effectiveness, as hardware cost is reduced using pre-existing Wi-Fi infrastructures. A comparison between IEEE 802.15.4 systems with a 6LoWPAN adaptation layer and low-power Wi-Fi is carried out in [27]. The recent development of power-efficient Wi-Fi components has the advantage of easy integration with existing infrastructures and built-in IP network compatibility, thus offering key cost savings and faster deployment. These features motivate the adoption of Wi-Fi technologies for all applications with real-time requirements, for which response time and reliability become critical. In [28], it is shown that, considering high OSI layers, Wi-Fi modules can achieve long battery lifetime, despite the use of HTTP over TCP/IP. By this, standards and paradigms already employed on the Web can be leveraged to manage resources of SOs, with the further advantage of no need for protocol translation. This fosters the creation of fully Web-enabled devices [29], through the adoption of Internet protocols for constrained devices (e.g., CoAP). Nevertheless, an approach in which an HTTP request obtains, as a response, an HTML document with a graphical representation of the resource, is heavy for both client and server, which must to parse and generate a complete HTML document, re-

spectively. This evidence motivates the need to define lighter paradigms for resource representation. Finally, in [30], WebPlug is proposed to represent HTTP resources introducing some ontology-related concepts. However, this solution seems to be in conflict with known axioms of Uniform Resource Identifier (URI) representation and requires HTTP nodes to handle another REST-like paradigm, instead of reusing those adopted by CoRE WG [31].

1.3.2 Power Line Communication

PLC is manly related to the use of existing electrical cables to transport data and has been investigated for a long time. Power utilities have been using this technology for many years to send or receive (limited amounts of) data on the existing power grid. Although PLC is mostly limited by the type of propagation medium, it can rely on existing wiring in the distribution network. According to EU's standards and laws, electrical utility companies can use PLC for low bit-rate data transfer (with data rates lower than 50 Kbps) in the $3 \div 148$ kHz frequency band. This opens new opportunities and new forms of interactions among people and things in many application areas (e.g., smart metering services and energy consumption reporting, traversing the power line wires to reach data concentrators), making the PLC medium an enabler for sensing, control, and automation in large systems spread over relatively wide areas (e.g., SC and SG scenarios). Thus, on top of PLC, one can adopt also enabling technologies that can improve smart automation processes, such as the IoT. For instance, the adoption of the PLC technology in industrial scenarios (e.g., remote control in automation and manufacturing companies), paves the way to the concept of Industrial IoT [32]. In order to prove the growing interest in the joint adoption of IoT and PLC paradigms to improve the communication system robustness, it can be highlighted that several applications are enabled by the following key feature of PLC technology: its ability to recover from network changes (in terms of repair/improvement, physical removal, and transfer function) mitigating the fallout on the signal transmission.

In last years, the PLC paradigm has been investigated in terms of its applicability to modern communication scenarios. An overview on the role of communication and networking technologies in the transformation of existing electric power systems into

SGs is proposed in [33] where, after discussing on the key drivers for the development of SGs, a data-centric perspective for enhancing communications in this field is then adopted. Even though research challenges (including reliability, timeliness, and data management services) are discussed in conjunction with possible future research directions, in [33] no experimental results are provided.

In [34], the enabling technologies for SGs and a possible roadmap for their profitable evolution are discussed, motivated by the fact that the quest for sustainable energy models is an important research driver for SGs. Furthermore, an analysis on how current standard solutions (carried out by Internet standardization partners such IETF, ETSI [35] and W3C [36]) can be engineered into a system, that fulfills the needs of the SG vision, leads the authors of [34] to the suggestion of using small and resource-constrained devices (namely, IoT), with pervasive computing capabilities, as key components to implement an energy control system. In [34], however, no experimental result is presented.

Another comprehensive overview on PLC in the context of SGs is proposed in [37], where network control problems inherent to SGs are discussed. The PLC channel is modeled as a fading channel and, relying on this approach, control and traffic models are investigated in order to achieve a better understanding of the communications requirements needed in the PLC field. Unfortunately, in [37] only theoretical considerations are carried out, with no experimental evidence.

Investigating the available protocols for PLC, in [38] an overview on the physical layer of two different PLC protocols is presented, trying to address the requests of emerging standards on Narrowband PLC (NB-PLC). The proposed theoretical analysis, aimed at selecting the best PLC protocol, is supported by simulation results. The application of NB-PLC in SGs is also investigated in [39], comparing the advantages and drawbacks of PLC technology with respect to other communication solutions in energy distribution networks. NB-PLC is shown to be suited for medium-voltage (MV) networks, due to their vast and complex geographical extension. In [39], an analysis of the impact of channel and topological characteristics of MV distribution networks on the design and implementation of the PLC infrastructure is also presented. Another overview of PLC for SGs is presented in [40], where current

protocols for PLC scenarios are investigated to derive design guidelines for future standards. However, in [38, 40], no experimental analysis is carried out.

A computational tool, able to simulate PLC systems for access applications in a SG scenario, is proposed in [41]. This tool is an event-based network simulator implemented in C++ language using OMNeT++ [42], and can be used to test a PLC network off-line before a real deployment. However, this tool allows to adopt only a single signal modulation and cannot be used in a real scenario. An extensive overview on PLC technologies and their integrability with IoT is presented in [43]. The authors conclude that there is no need for additional wires to power devices (e.g., smart meters in buildings with their corresponding data concentrator), thus motivating the following choices: adoption of the PLC paradigm, by most electrical utilities, for their SG projects; and adoption, by most cities, of ubiquitous computing and PLC for their smart street lighting projects. However, in [43] no experimental results, that can further motivate a wide adoption of PLC in Smart Cities scenarios, are provided.

An enhanced IoT-PLC concept, denoted as Power Internet of Things (PIoT), is introduced in [44]. PIoT can be applied to different SG scenarios, such as power transformation, distribution and consumption, and relies on three layers: perception, networking, and application. The proposed architecture is intended to directly monitor high-voltage transmission lines (that are weather-sensitive and can paralyze large area power supply systems) through two main components: one component is installed along with the transmission wires to monitor the status of the conductors; and the other component is installed on the transmission towers to monitor the environment and the states of the towers.

1.4 Software-Defined Networking

The basic concept of the SDN is the decoupling of the network control and forwarding functions [45], enabling the underlying infrastructure (e.g., the internal network functions of IoT nodes) to be abstracted and programmable in the control plane (e.g., the layer at which the IoT nodes takes decisions). The network control function is logically centralized in an entity called *controller* that provides an abstract and cen-

tralized view of the overall network to the SDN applications running on top of the control plane. The interface between the application plane and the control plane is called NorthBound Interface (NBI), while the interface between the control plane and the data plane is called SouthBound Interface (SBI). Due to their decoupled nature, SDN solutions can be applied in different scenarios, from IoT-related networks to enterprise data centers, till large carrier networks. In some cases, a pure SDN solution based on L2 switches interconnected to a centralized controller (e.g., in data centers) can be used, while in other contexts a more complex solution including the interaction with standard L3 routing protocols is needed. OpenFlow (OF) [46] is a protocol that has been specifically designed for the SBI.

1.4.1 Segment Routing

The Segment Routing (SR) paradigm [47] leverages on the *source routing* concept and aims at providing enhanced packet forwarding behavior without requiring per-flow state maintenance within the network, leading to a reduction of the complexity for both control and user planes. With SR the path of a packet can be enforced through an ordered list of processing/forwarding functions, called segments, that may consist of both logical and physical elements: for example a segment may be a packet filter, a network node, or a network link. The chain of these elements forms the *SR path* of a packet, identified by a list of segment identifiers (SIDs). The scope of such SIDs can be global or local. While global SIDs are defined globally and should be recognized by all network nodes, local SIDs are defined locally within a node and the use of local SIDs by other nodes requires the implementation of an explicit distribution mechanism.

SR lends itself to support different applications: Virtual Private Networks (VPNs), protection/restoration, Traffic Engineering (TE), Service Function Chaining (SFC), Network Function Virtualization (NFV), Operation and Management (OAM). The standardization activity on the SR architecture is relatively recent. The status of the draft is mature and different independent implementations are now available. Real world deployments are ongoing, as SR has captured the interest of network providers and of “Over the Top” providers.

SR is a general concept that needs to be mapped onto a specific forwarding technology that supports source routing, for example Multi Protocol Label Switching (MPLS) or IPv6. In case of MPLS, SIDs are represented by MPLS labels, as well as in case of IPv6 choice, in which SIDs equal to IPv6 addresses.

In general the computation of the source routed paths and the configuration of the border nodes can be realized either in a distributed or in a centralized way. In the former case, the control logic of border nodes needs to be further enhanced. In the latter case, the SDN architecture represents a perfect fit: a SDN approach can be used to properly configure the SR services in the border nodes, with minimal or no increase of the complexity of the border node.

1.5 Cloud Computing

Cloud Computing represents the increasing trend moving to the external deployment of IT resources, obtaining them as services [48]. Cloud Computing enables convenient and on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage elements, applications, and services) that could be rapidly provisioned and released with minimal management effort or service provider interaction [49].

At hardware level, a number of physical devices, including processors, hard drives, and network devices, fulfill processing and storage needs. Above this, the combination of (i) software layer, (ii) virtualization layer, and (iii) management layer, allows effective management of servers. In Cloud Computing, available service models are the following.

- **Infrastructure-as-a-Service (IaaS)**: provides processing, storage, networks, and other computing resources, allowing the consumer to deploy and run arbitrary software, including Operating Systems (OSs) and applications. The consumer has control over OSs, storage, deployed applications and, possibly, limited control of select networking components.

- **Platform-as-a-Service (PaaS)**: provides the capability to deploy infrastruc-

ture, consumer-created, or acquired applications. The consumer has no control on the underlying infrastructure (e.g., network, servers, OSs, or storage) but only manages deployed applications.

- **Software-as-a-Service (SaaS)**: provides the capability to use the applications owned by the provider, running on the Cloud infrastructure, by accessing from various client devices through proper client interfaces. The consumer does not manage or control the underlying Cloud infrastructure or individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Cloud Computing is generally complementary to the IoT scenario, as it acts (i) as collector of real-time sensed data, and (ii) as provider of services built on the basis of collected informations. The main need is to be extremely scalable, allowing the support to large-scale IoT applications.

Several research projects address their effort to provide a solution for the deployment and management of a pervasive IoT-Cloud infrastructure. The EU FP7 project OpenIoT [50] aims to provide a systematic and structured solution to the management of utilities based on IoT environments. It can be considered as an extension of traditional Cloud Computing implementations since it is specifically designed to allow access to different and heterogeneous IoT resources and capabilities. To summarize, the main objectives of OpenIoT framework are:

- to create an open-source middleware for getting information from sensor, without having to concern about what exact they are used;

- to explore efficient ways to use and manage Cloud environments for IoT "entities" and resources (such as sensors, actuators and smart devices) and offering utility-based IoT services;

- to provide instantiations of Cloud-based and utility-based sensing services, enabling the concept of "Sensing-as-a-Service" via an adaptive middleware framework for deploying and providing services in Cloud environments.

Another example of this approach is given by the FIWARE project [51], an open Cloud-based infrastructure for cost-effective creation and delivery of Internet applications and services. FIWARE API specifications are public, royalty-free, and compliant to OCCI [52] specifications, driven by the development of an open-source reference implementation which allows developers, service providers, enterprises, and other organizations to develop innovative products based on FIWARE technologies. The FIWARE solution is based on the Openstack project [53], an open Cloud OS that controls large pools of computing, storage, and networking resources for public and private Clouds. The project aims to deliver solutions for all types of Clouds by being simple to implement, massively scalable, and feature-rich. While OpenStack can be seen as a framework following a vendor-driven model, OpenNebula [54] represents an open-source Cloud platform focused on the user, aiming at delivering a flexible and simple feature-rich solution to build and manage virtualized data centers and enterprise Clouds.

1.5.1 Security Issues in Cloud and IoT Scenarios

Cloud Computing is constantly growing and so are concerns about security and privacy of data, that give rise to additional Cloud-specific vulnerabilities (i.e., the possibility that a component will be unable to resist against the actions of a threat agent). Following the public, shared, and virtualized nature of the Cloud Computing paradigm, the migration of assets (e.g., data and applications) from an administrative control environment to a shared environment where many users are collocated, highly increases security concerns [55, 56].

Similarly to other paradigms, Cloud Computing needs to manage the digital identity of its users. A possible solution is represented by Identity Management (IdM) [57], corresponding to a set of capabilities, such as maintenance, administration, authentication and policy enforcement, used to ensure identity information and guarantee security. In order to allow access control in open environments, IdM can be implemented in different ways: (i) *in-house* implementation, in which identities are issued and managed by single companies, which have complete responsibility on them; and (ii) *managed* implementation, in which the Identity-as-a-Service

(IDaaS) concept is adopted and the IdM is outsourced to external companies. Regardless of the chosen configuration, an IdM system (IMS) controlling the platform identities always involves three main types of entities: (i) the user; (ii) the Identity Provider (IdP), responsible for issuing and managing user identities and credentials; and (iii) the Service Provider (SP), responsible for providing services based on identities/attributes of the users. An IMS must always provide the following functions: (i) provisioning of identities related to the different types of accounts adopted by the organization (e.g., administrator, IT consultant, developer, end user); (ii) authentication and authorization functions, identifying individuals through various mechanisms (e.g., username/password, X.509 certificates, One-Time Password (OTP), biometrics) and providing them different access levels for different operations within a computing infrastructure; and (iii) grouping SPs into a federation and, then, establishing a trust that allows sharing of identities information among them.

Even if the topic is not fully explored, some works have recently appeared in the literature to address the problem of applying security mechanisms to the IoT-Cloud environment. In [58], security issues in an IoT-based e-Health scenario are analyzed, providing a framework mainly focused on vulnerabilities associated with: (i) endpoint access; (ii) Cloud services; and (iii) partners and providers. In [59], a SC scenario is explored, focusing on the problem of securing Cloud API for sensing and actuation in an open sensor platform. Regardless of the selected use case, the most relevant technologies employed for implementing an IMS can be summarized as follows.

- **Active Directory (AD)**: a directory service created by Microsoft for Windows domain networks.
- **Security Assertion Markup Language (SAML)**: a XML-based open standard for exchanging authentication and authorization data between security domains (i.e., between an IdP and a SP).
- **Single Sign-On (SSO)**: a mechanism for access control of multiple (related, but independent) software systems, in which a user logs in only once and gains access to all related systems without the need to log in again.

Storing and managing the identities presents crucial security concerns for Cloud providers, because stored information can be tampered or modified by malicious or unauthorized users.

Several technologies implement federated identity; among them, one of the most important is Shibboleth [60], which is a widely employed Authentication and Authorization Infrastructure (AAI) based on SAML that allows to create a safe structure that simplifies the management of identities; moreover, it provides the user a SSO layer for different organizations belonging to the same federation, in which identity informations are shared. A Shibboleth system is built on two main entities, the IdP and the SP, and on an optional one, the Where Are You From (WAYF) module. The Shibboleth IdP manages the authentication of the users, maintaining credentials and attributes. The SP is located where the resources are stored and accessed by the user. The optional WAYF component allows an association between a user and multiple organizations. When the access to a protected resource is tried, the user is forwarded to an interface that asks him/her to choose the organization which he/she belongs to; after this choice, the user is redirected to his/her correct IdM interface to start the authentication process. The WAYF service can be distributed as part of a SP, becoming quite useful when it is used with SPs offering resources for registered users in several IdPs.

In addition to these IdM-based solutions, several methods and strategies to enable confidentiality in IoT infrastructures (especially publish/subscribe) are proposed. IoT systems have to avoid security threats, providing strong security foundations built on a holistic view of security for all IoT elements at all stages: from object identification to service provision; from data acquisition to stream processing. All security mechanisms must ensure: resilience to attacks, data authentication, access control, and client privacy.

In [61], IoT systems are expected to bridge the physical and the “virtual” worlds, using a novel broker that supports protocols such as HTTP and MQTT, adhering to the REST paradigm and allowing developers to easily and responsively expose fundamental entities as REST resources. This broker does not address any security issues, claiming that possibles solutions could include: plain authentication, VPNs,

Access Control Lists (ACLs), as well as OAuth2 [62], a new type of authorization which is used to grant access to personal data by third-party applications.

In [63], the authors examine roles of different actors comprising an inter-domain publish/subscribe network, along with security requirements and minimal required trust associations between entities, introducing and analyzing an architecture that secures both data and control planes. The main security goals for a publish/subscribe architecture are: (i) integrity, (ii) scalability, (iii) availability, and iv) prevention of underived traffic. Finally, in [63], different actors and security mechanisms are identified. The main mechanism is Packet Level Authentication (PLA) which, combined with cryptographic signatures and data identifiers tied to secured identifiers, creates a strong binding between data and traffic, thus preventing Denial of Service (DoS) attacks.

The work of [64] treats security issues relying on the requirements of a particular application and on an external publish/subscribe infrastructure. General security needs of the applications include confidentiality, integrity, and availability. At the opposite, the security concerns of the infrastructure focus on system integrity and availability. Security issues in publish/subscribe platforms rely on authentication, information integrity, subscription integrity, service integrity, user anonymity and information confidentiality, in addition to subscription confidentiality, publication confidentiality, and accountability.

In [65], a study of confidentiality in Content-Based Publish/Subscribe (CBPS) systems is presented, defined as an interaction model storing the interests of subscribers in a content-based infrastructure, to guide routing of notifications to interested subjects. In agreement with the approach in [64], confidentiality aspects are decoupled into two facets, namely notification and subscription, suggesting that: a confidential CBPS (C-CBPS) must satisfy correctness and notification; whereas a subscription CBPS must satisfy unforgeability and security, and match isolation. A high level approach to obtain C-CBPS relying on notifications using simple blocks, that may be controlled and checked better than a whole encryption of those ones, is proposed.

In [66], the use of Federated Identity and Access Management (FIAM) in IoT is

analyzed, following a consumer-oriented approach, where consumers own data collected by their devices, having a control over entities who access these data. Traditional security models, based on the concept of roles with a hierarchical structure, are not applicable for IoT scenarios (because of the billions of devices involved, the impossibility to adopt a centralized model of authentication, and the necessity to support mechanisms for delegation of authority). The authors propose OAuth2 as a possible solution to achieve access management with IoT devices which support the MQTT protocol. The overall system consists of: (i) a MQTT broker, (ii) an Authorization Server supporting OAuth2, (iii) a Web Authorization tool, and (iv) a device.

The work of [67] tackles the problem of application security in the Cloud, aiming at incorporating end-to-end security, so that Cloud providers not only can isolate their clients from each other, but can also isolate the data generated by multiple users which access a particular service provided by the Cloud. An approach called "application-level virtualization", which consists of (i) removing from applications all the details regarding security and flow control, (ii) placing the security management logic in the Cloud infrastructure, and (iii) allowing providers to permit only the interactions that the clients specify, is proposed.

1.6 Fog Computing

In the area of user-driven and Cloud IoT architectures, in [68] is proposed the concept of Fog Computing as a novel and appropriate paradigm for a variety of IoT services and applications that require mobility support, low latency, and location awareness. The Fog Computing can be described as a highly virtualized platform that provides computing, storage, and networking services between end-devices and the Cloud. In other words, the Fog Computing is meant to act as an extension of the Cloud, operating at the edge of the network to support endpoints by providing rich services that can fulfill real-time and low-latency consumer's requirements. The Fog Computing paradigm has specific characteristics, which can be summarized as follows:

- geographical distribution, in contrast with the centralization envisioned by the Cloud;

- subscriber model employed by the players in the Fog;
- support for mobility.

The architecture described in [68] is based on Fog and Cloud interplay: the former provides localization, low-latency, and context awareness to endpoints; the latter provides global centralization functionalities. In the presented IoT/Fog scenario, collectors at the edge of the network manage the data generated by sensors and devices: the portion of these data that require real-time processing (from milliseconds to tenths of seconds) are consumed locally by the first tier of the Fog. The rest is sent to the higher tiers for operations with less stringent time constraints (from seconds to minutes). The higher is the tier, the wider is the geographical coverage and the longer the time scale. As a result, the Fog must support several types of storage: from ephemeral, at the lowest tier, to semi-permanent, at the highest tier. The ultimate and global coverage is provided by the Cloud, which is used as repository for data with a potential duration of months or years. In Fig. 1.1 the hierarchical structure of layers involved in data collection, processing, and distribution in IoT scenarios, is shown.

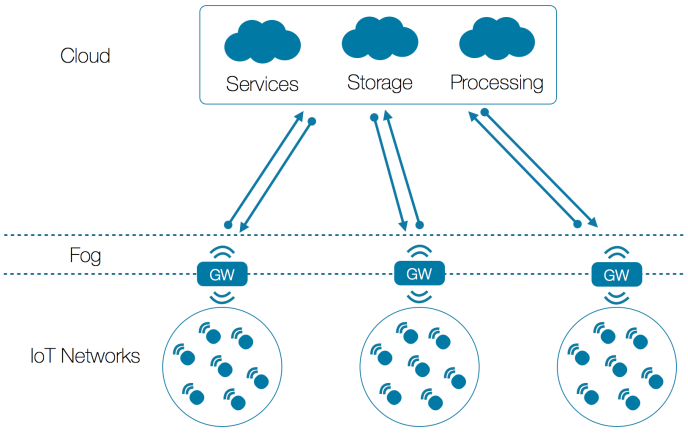


Figure 1.1: The hierarchy of layers involved in IoT scenarios: the Fog Computing works as an extension of the Cloud Computing to the network edge to support data collection, processing, and distribution.

1.7 Big Data Processing Pattern

From a business perspective, managing and gaining insights from data is a challenge and a key to competitive advantage. Analytical solutions that collect structured and unstructured data are important, as they can help companies to gain cross-related information not only from their privately acquired data, but also from large amounts of data publicly available on the Web, social networks, blogs and, thus, coming from IoT-oriented architectures. Big Data opens a wide range of possibilities for organizations to understand the needs of their customers, predict their demands, and optimize the use of evaluable resources. The work of [69] illustrates that the Big Data notion is different and more powerful with respect to traditional analytic tools used by companies. As analytic tools, Big Data can find patterns and glean intelligence from data translating, and this translates into business advantage. However, Big Data is powered by what is often referred as a "multi V" model, in which "V" stands for:

- **Variety**: to represent the data types;
- **Velocity**: to represent the rate at which the data is produced and processed and stored according with further analysis;
- **Volume**: to define the amount of data;
- **Veracity**: refers to how much the data can be trusted given the reliability of its sources.

Big Data architectures generally use traditional processing patterns with a pipeline approach [70]. These architectures are typically based on a processing perspective where the data flow goes downstream from input to output, to perform specific tasks or reach the target goal. Typically, the information follows a pipeline where data are sequentially handled with tightly coupled pre-defined processing sub-units (static data routing). The described paradigm can be defined as "process-oriented": a central coordination point manages the execution of sub-units in a certain order and each sub-unit provides a specific processing output, which is created to be used only within the scope of its own process without the possibility to be shared among different

processes. This approach represents a major deviation from traditional Service Oriented Architectures (SOAs), in which the sub-units are external Web Services (WSs) invoked by a coordinator process rather than internal services [71]. Big Data applications generally interact with Cloud Computing architectures which can handle resources and provide services to consumers. In [72], the authors presents a survey on approaches, environments, and technologies on key-areas for Big Data analytics capabilities, investigating how they can contribute to build analytics solutions for Clouds. A set of gaps and recommendations, for the research community, on future directions on Cloud-supported Big Data computing are also described.

1.8 Stream and Real-Time Management

In literature there have been presented different platforms abiding by low-latency and real-time requirements. Possible examples of such solutions are Apache Storm [73] and Apache S4 [74].

Storm is a free and open-source distributed real-time computation system to reliably process unbounded streams of data. The system can be integrated with different queuing and database technologies and provides mechanisms to define topologies in which nodes consume data streams and process them in arbitrarily complex ways. S4 is a general purpose, near real-time, distributed, decentralized, scalable, event-driven, and modular platform that allows programmers to implement applications for processing streams of data. Multiple application nodes can be deployed and interconnected on S4 clusters to create more sophisticated systems. The most relevant use cases for Storm and S4 are stream processing and continuous computations on data stored in databases (e.g., message processing for database update).

In [75] it is addressed the problem to process, procure, and provide information related to the IoT scenario with almost zero latency. As an example, it is considered a taxi fleet management system, which has to identify the most relevant taxi in terms of availability and proximity to the customer's location. The core of the publish/subscribe architecture proposed in [75] is the Mediator, which encapsulates the processing of the incoming requests from the consumer side and the incoming events

from the services side. Services are publishers (taxis in the proposed example) which are responsible to inform the Mediator if there is some change in the provided service (e.g., the taxi location or the number of current passengers). Thus, instead of pulling data at consumer's request time, the Mediator knows at any time the status of all services, being able to join user requests with the event stream coming from the taxis, using temporal join-statements expressed through SQL-like expressions.

Chapter 2

Access and Backhaul Technologies for IoT

The gigantic information exchange coming from IoT-oriented scenarios enables new opportunities and new forms of interactions among things and people, and requires solutions about the transmission mediums that can be used to handle IoT-related data rates. These solutions need to be adopted at different layers of the communication, trying to reuse, if possible, already available transmission architectures, as well as combining existing paradigms with IoT concept. In this context it is interesting to analyze the effects and the consequences of the joint adoption of IoT and PLC.

2.1 IoT on Power Line Communication

As known, the IoT can be defined as a "network of networks" of physical devices connected in an Internet-like structure, thus enabling them to collect, exchange and process data. In the last years, the services over the IoT have evolved due to the needs identified by the new interactions (e.g., people-to-people, people-to-machine and Machine-to-Machine (M2M) interactions). In this way, the IoT paradigm allows to join real and virtual worlds, especially when combined with other technologies, such as mobile and sensing technologies, and home networking applications (e.g.,

smart metering).

Thus, it is interesting to evaluate how the joint adoption of IoT and PLC concepts allows to rely on IoT protocols (e.g., CoAP, HTTP, CoSIP) over power lines, since this has relevant implications on smart infrastructure management [76].

2.1.1 Power Line Communication Modulations

Over the past several years, there have been intense research activities on modeling powerline channels. Due to a significant interest in adopting the low-frequency bands for communication in PLC scenarios (from 20 KHz to 500 KHz), various standardization institutes have defined several PLC bands to regulate the frequency utilization, as illustrated in Fig. 2.1. Furthermore, the European Standard CENELEC EN50065 [77] has divided the low frequency power line spectrum, between 3 KHz and 148.5 KHz, into four different frequency bands, referred to as, respectively:

- "CENELEC A" band: 3 KHz $\leq f \leq$ 95 KHz;
- "CENELEC B" band: 95 KHz $< f \leq$ 125 KHz;
- "CENELEC C" band: 125 KHz $< f \leq$ 140 KHz;
- "CENELEC D" band: 140 KHz $< f \leq$ 148.5 KHz.

In Japan, the regulatory entity ARIB has defined an available PLC transmission band between 10 KHz and 450 KHz [78]. In the United States, the whole spectrum between 10 KHz and 490 KHz has been allocated to one wideband channel by the Federal Communications Commission (FCC) [79].

Following these band allocations, different physical layer protocols have been recently defined, to support data transmission in the low frequency bands, and further adopted by different PLC modems vendors (e.g., Texas Instruments (TI), Maxim Integrated, STMicroelectronics). The main protocols to date are here summarized.

- *PRIME* [80]: intended for PLC-based modems operating in the frequency range between 42 KHz and 88 KHz using Orthogonal Frequency-Division Multiplexing (OFDM).

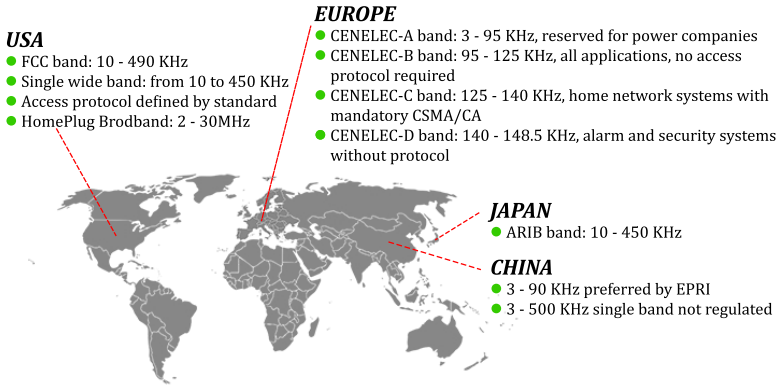


Figure 2.1: PLC bands.

- *HomePlug* specification [81]: operating at frequencies up to 400 KHz.
- *G3-PLC* [82]: intended for PLC-based modems operating in a sub-frequency range of the CENELEC A band, from 35 KHz to 91 KHz.
- *G.hnem* specification [83]: drafted by ITU and selecting G3-PLC and PRIME specification as two annexes to its main body.
- *IEEE P1901.2* [84]: defined by IEEE and adopting G3-PLC and PRIME specifications as two profiles for PLC communication.

2.1.2 Experimental Evaluation

In order to investigate the joint integration of the PLC paradigm in an IoT scenario and to evaluate the performance of the overall communication architecture, an experimental evaluation has been carried out on a residential electrical line, in order to verify the capabilities to exchange data collected from different IoT devices (through their on-board sensors, e.g., temperature, humidity, brightness, and proximity sensors). In particular, the "CENELEC A" mask has been selected, in its 36 sub-carriers version (with adjacent sub-carrier separation $\Delta f = 1.5625$ KHz), since the other masks did not work properly on the electrical line, as verified by further experimental

tests.

Among many PLC modems produced by different vendors, a PLC kit able to support both PRIME and G3-PLC protocols and provided by TI has been selected, namely the TMDSPLCKIT-V3 kit [85]. This modem allows to use different modulations (*Robust Modulation (ROBO)*, *Binary Phase-Shift Keying (BPSK)*, *Quadrature Phase-Shift Keying (QPSK)*, *Eight Phase-Shift Keying (8PSK)*) and can operate in a double transmission mode (*Stream*/*NO Stream*). Assuming a transmitter module (TX) and a receiver one (RX), the transmission modes work as follows:

- if the *Stream* mode is deactivated (*NO Stream* mode), the RX module has to send back an ACK message to the TX module for every received packet (slow transmission);
- if the *Stream* option is activated, the RX module has not to send back any acknowledgment message to the TX module (fast transmission).

The experimental evaluation has been carried out by adopting the G3-PLC transmission protocol, due to its features and manageability in the chosen TI PLC modem, and has been mainly split into two phases.

The former experimentation involves the adoption of an application provided by the vendor of the chosen PLC modems. The latter is based on the adoption of a self-developed software library that allows to interact with the chosen PLC modems and to transmit data (obtained by on-board sensors equipping the IoT devices) through the electrical line.

Experimental Evaluation using PLC Modems as "Black Boxes"

In order to test the functionalities of the chosen PLC modem and of its APIs, an external Java-based library, with which it is possible to interact with PLC modules, has been developed. This library is based, for its serial communication features, on the jSSC library [86] and needs to adhere to the TI-defined request/response "HostMessage" protocol, by which it is possible to initialize a G3-PLC-based PLC network and to communicate over the power line. The configuration of a PLC modem is carried out through the following steps: (i) system initialization (e.g., check for existing

configurations, current configuration loading, system reboot); (ii) network configuration, required to make the modem part of an IP power line-based network (network parameters configuration, Base Node (BN) discovery, PLC module attachment to the BN); (iii) data transmission, in which the user transmits a "DATA TRANSFER" command to the PLC modem, which replies back with a confirmation message and starts sending the message, properly encoded, on the power line.

Among other features, the Java-based library includes some APIs for managing the PLC modules in two distinct ways, denoted as "Point-to-Point" and "Service Node" configurations, as follows.

- "Point-to-Point" mode: the PLC module registers itself with the power line and waits for an input from the user; in the meantime, the PLC module can receive, in an asynchronous way, messages from another PLC module.
- "Service Node" (SN) mode: the PLC module registers itself on the power line network, waiting for a joining acknowledgment from a running BN. Once it has completed this joining step, it belongs to G3-PLC network and is addressable with an IPv6 address released by the BN.

Considering the Point-to-Point configuration, different scenarios with two TI PLC modules have been tested, on both cold and hot lines, managing them with the Java-based library. In these tests, a transmitter module sends a HostMessage-based packet to a second PLC module which receives it and, then, replies to the TX with another HostMessage-based packet.

Regarding the SN configuration, an experimental scenario composed by a single SN (corresponding to a PLC modem) and by a BN (whose features are provided by the TI Data Concentrator TMDSDC3359 [87]) is considered. In order to prevent any damages to the BN module, the SN scenario has been deployed on a cold line. With this configuration, a successful communication has been experimented, in which the BN initializes the G3-PLC network and the SN correctly joins the G3-PLC network, becoming an active member of the system. Another experimental scenario, composed of a TI Data Concentrator TMDSDC3359 as BN and with two PLC modules as SNs, has been successfully investigated. The obtained results show that, by

using the Java-based library, is possible to both identify the SNs through their IPv6 addresses (assigned by the BN) and let them exchange HostMessage-based packets.

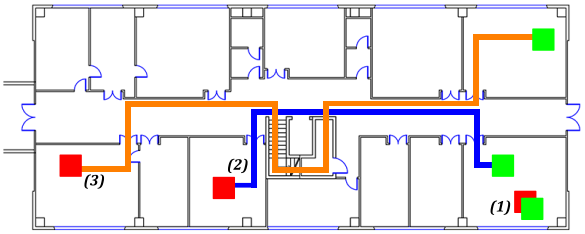


Figure 2.2: Map of a scientific building of the School of Engineering of the University of Parma.

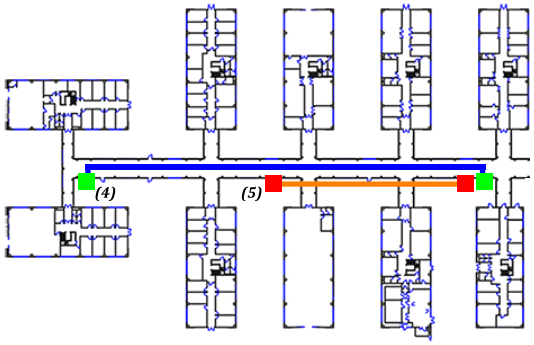


Figure 2.3: Map of the corridor between the different scientific buildings of the School of Engineering of the University of Parma.

In Table 2.1, the results of several experimental tests, based on both Point-to-Point and SN configurations, are reported. The performance metrics of interest in each test are: the measured distances between PLC endpoints and the throughput. As expected, the main causes of communication failures (namely, electrical line noises and misconfigurations) are unpredictable and out of direct control.

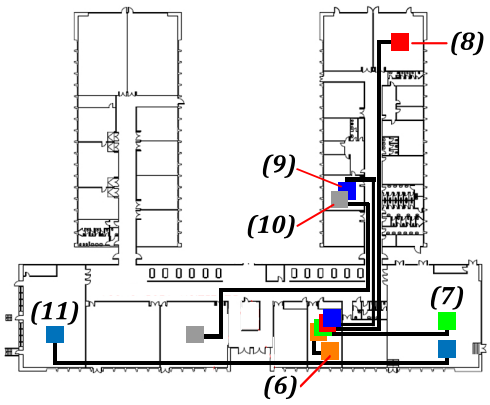


Figure 2.4: Map of the learning buildings of the School of Engineering of the University of Parma.

Evaluation on a Supervised Electrical Line

Another experimental evaluation has been carried out by transferring between the two PLC modems an 88 Kbyte file. Since each PLC transmission packet (at physical layer) has to be 256 byte long, the transmission of an 88 Kbyte packet requires transmitting 351 PLC packets. In this case, the evaluation has been carried out, through a proper application available from TI, on a close-set electrical line. Different physical layer settings have been considered to test the communication on both a cold line (a power strip disconnected and isolated from the electrical residential line) and a hot line (a power strip connected and powered by the electrical residential line). In particular, the considered configurations are the following: “CENELEC A 36”, “CENELEC A 25” (use of the “CENELEC A” mask with 25 sub-carriers), “CENELEC B”, “CENELEC BC” (associated with a broader frequency range given by the union of “CENELEC B” and “CENELEC C” frequency bands), and “CENELEC BCD” (obtained by the union of the “CENELEC B/C/D” masks). The performance results, associated with the test on a hot line, are shown in Table 2.2, in columns 1-4.

Considering the values obtained with the inclusion of “CENELEC B” mask, it can be observed that the majority of the configurations with “CENELEC BC” and

Table 2.1: Performance results of experimental tests, using PLC modems as "black boxes", in both Point-to-Point and SN configurations.

Configuration	Line	Description	Throughput	Reference
Point-to-Point / SN	COLD	Zero distance between the PLC modems.	100%	Fig. 2.2 (1)
SN	HOT	Same building, with 5 m between each line entry points.	100%	Fig. 2.2 (2)
SN	HOT	Same building, two different floors, with 15 m between each line entry points.	100%	Fig. 2.2 (3)
SN	HOT	Corridor between 2 buildings, with 50 m length.	0%	Fig. 2.3 (4)
SN	HOT	Corridor between 2 buildings, with 25 m length.	0%	Fig. 2.3 (5)
SN	HOT	Same building.	100%	Fig. 2.4 (6)
SN	HOT	Same building, with 10 m length.	100%	Fig. 2.4 (7)
SN	HOT	Same building, with 40 m length.	0%	Fig. 2.4 (8)
SN	HOT	Same building, with 15 m length.	100%	Fig. 2.4 (9)
SN	HOT	Same building, with 20 m length.	0%	Fig. 2.4 (10)
SN	HOT	Corridor between 2 classrooms at 50 m length.	0%	Fig. 2.4 (11)

"CENELEC BCD" do not provide any data transmission. Moreover, the throughput obtained with the "CENELEC B" mask, for each configuration, is lower than the corresponding one obtained with the "CENELEC A 25" and "CENELEC A 36" masks. This is likely due, on one end, to a reduced frequency bandwidth and, on the other end, to the incompatibility between the available modulations and masks different from "CENELEC A".

Although the best performance has been obtained with the "CENELEC A 36" mask, a further experimental test has been carried out considering the same settings assumed in the previous tests, except for the activation of the Tone Map Request (TMR) option (a TI-provided setting that allows to tune the tone map of OFDM for each currently selected frequency range). The obtained results are shown in Ta-

Table 2.2: Performance results on hot lines, using default parameters and adopting different modulations: “CENELEC A 36”, “CENELEC A 25”, “CENELEC B”, “CENELEC BC”, “CENELEC BCD” masks.

CENELEC Mask		“B”	“BC”	“BCD”	“A 25”	“A 36”			
Packet Length		256 bytes				256 bytes		1024 bytes	
Modulation	Stream	TMR OFF				TMR ON	TMR OFF	TMR ON	TMR OFF
ROBO	OFF	729	1358	1617	1335	3389	1975	4901	2634
	ON	902	1691	2115	1648	4976	2775	6076	3060
BPSK	OFF	2984	3928	✗	3772	3418	4568	4894	6661
	ON	4200	5846	✗	5600	4968	7074	6069	8769
QPSK	OFF	4013	✗	✗	4669	3483	5586	4866	8142
	ON	5874	✗	✗	7374	4987	8959	6089	11212
8PSK	OFF	2501	✗	✗	5148	3440	5985	4906	8766
	ON	5846	✗	✗	8247	4971	9807	6089	12344

ble 2.2, in columns 5-6. Comparing the results obtained with activated TMR (column 5) with those with deactivated TMR (column 6), it is possible to observe that the activation of the TMR option only increases the transmission throughput when the *ROBO* modulation is adopted (improving it by almost 70%). At the opposite, in all other cases the throughput decreases by approximately 45%.

Considering the “CENELEC A 36” mask (which guaranteed the highest throughput), it has been performed another test by increasing the PLC packet size from 256 byte to 1024 byte. The transmission of an 88 Kbyte file thus requires 88 PLC packets. The obtained results, with activation and deactivation of the TMR option, are reported in the columns 7-8 of Table 2.2. Comparing the results with activated TMR with those obtained with deactivated TMR (more generally, with the results highlighted in columns 1-6), it can be concluded that the highest throughput (around 12 kbps) is achieved transferring a file in *Stream* mode, using the “CENELEC A 36” mask with *8PSK* modulation and disabling the TMR option.

Experiments on a Hot Electrical Line

Subsequently, the focus has been placed on an experimental campaign carried out on a hot electrical line between two offices of the scientific buildings of the School of Engineering of the University of Parma. Following the guidelines suggested by the results previously highlighted, the experimental tests have been carried out using the "CENELEC A 36" mask and transferring, between the two PLC modules, an 88 Kbyte file with 351 256 byte TMR transmission packets (at physical layer). The results obtained with this physical layer setting, activating and deactivating the TMR option, are shown in Table 2.3, columns 1-2.

Table 2.3: Performance on real hot line, transferring packets and using "CENELEC A 36" and "CENELEC A 25" masks, and enabling and disabling the TMR option.

CENELEC Mask		"A 36"				"A 25"	
Packet Length		256 bytes		1024 bytes		1024 bytes	
Modulation	Stream	TMR ON	TMR OFF	TMR ON	TMR OFF	TMR ON	TMR OFF
ROBO	OFF	1972	1949	2549	2584	✗	✗
	ON	2723	2730	2990	2991	1875	1977
BPSK	OFF	1939	✗	2585	✗	✗	✗
	ON	2715	✗	3020	✗	1888	✗
QPSK	OFF	1966	✗	2578	✗	✗	✗
	ON	2719	✗	2930	✗	1955	✗
8PSK	OFF	1944	✗	2617	✗	✗	✗
	ON	2715	✗	2971	✗	1942	✗

Analyzing results in Table 2.3, one can observe that the activation of the TMR option allows the communication through a real electrical line with all the tested modulations, with an average throughput on the order of 2.7 kbps when the *Stream* mode is enabled.

The performance results obtained increasing the TMR packet size from 256 byte to 1024 byte (an 88 Kbyte file requires 88 PLC packets), activating and deactivating the TMR option, are shown in Table 2.3, columns 3-4. Comparing the results with 256 byte long PLC packets with those with 1024 byte long PLC packets, it is possible

to observe that, in the latter case: (i) with deactivated TMR option, the majority of the modulations still do not allow a communication between the PLC modules, showing only a little increase in the case of *ROBO* modulation: this means that the packet size increase does not improve the communication performance; (ii) the activation of the TMR option, instead, allows the communication with all available modulations, with highest throughput (3.0 kbps) in the case of *BPSK* modulation.

Furthermore, comparing these results with those in Table 2.2, it is possible to observe that: (i) with "CENELEC A 36" mask, activated TMR option, and 256 byte packets, the performance results in Table 2.2 (column 5) are higher (almost twice) than those in Table 2.3 (column 1); (ii) with "CENELEC A 36" mask, deactivated TMR option, and 256 byte packet, the only operational modulation in Table 2.3 (column 2) is *ROBO*, while in Table 2.2 (column 6) all modulations allow data transmission; (iii) with "CENELEC A 36" mask, activated TMR option, and 1024 byte packets, the performance results in Table 2.3 (column 3) are lower (almost half) than those in Table 2.2 (column 7); and (iv) with "CENELEC A 36" mask, deactivated TMR option, and 1024 byte packet, the only operational modulation in Table 2.3 (column 4) is *ROBO*, while in Table 2.2 (column 8) all modulations allow data transmission.

Further experimental tests have been carried out maintaining the "CENELEC A" band and changing the mask from "CENELEC A 36" to "CENELEC A 25", thus activating and deactivating the TMR option. The corresponding results are reported in Table 2.3, columns 5-6. Adopting the "CENELEC A 25" mask and disabling the TMR option, most of the modulations do not allow communication on the hot electrical line. The only operational configuration is the one with the *ROBO* modulation: however, in this case as well, the performance degrades with respect to that guaranteed by "CENELEC A 36". It can be observed that the activation of the TMR option increases the number of operational modulations (especially when the *Stream* mode is enabled). Furthermore, by comparing the results in Table 2.3 with those in Table 2.2, it can be observed that on a real hot line, the only operational modulation is *ROBO*, while in Table 2.2 all modulations allow data transmission.

Other communication tests on hot electrical lines have led to the following results.

1. Adoption of the “CENELEC B/C/D” masks, using the same configurations of previous tests on supervised electrical line: no data transmission on these bands was successfully carried out, regardless of the used modulation.

2. Adoption of different masks in TX and in RX, looking for combinations which would allow the communication on the electrical line: with the “CENELEC A 36” mask in TX and the “CENELEC A 25” mask in RX, no configuration worked; with the “CENELEC A 25” mask in TX and the “CENELEC A 36” mask in RX, no communication was allowed as well.

Experiments on Electrical Lines with Different Lengths

It has been preliminary recalled that it is not possible to control the exact extension and the loads on a real (walled) electrical line, as the one between different departments of the School of Engineering of the University of Parma. Therefore, in order to overcome this constraint, it has been assembled a handmade 300 m electrical line, composed by 6 50 m-long electrical cables, as shown in Fig. 2.5.

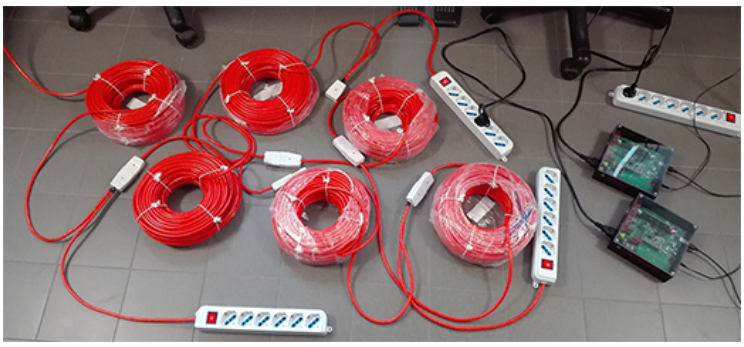


Figure 2.5: Real deployment of an electrical line composed by different pieces of electrical cable.

Adopting the “CENELEC A 36” mask, tests were carried out with activated TMR option and using a PLC packet size equal to 1024 byte. According to this setting, the transmission of an 88 Kbyte file requires 88 PLC packets on the electrical line.

Tests on a cold electrical line As in the test on a close-set electrical line previously described, in this case experimental tests were carried out increasing progressively the length of the cold electrical line from 50 m to 300 m, with a 50 m step. In Table 2.4, the obtained results, considering various modulations available in adopted tool, are shown. In Fig. 2.6, the corresponding throughput is shown.

Table 2.4: Transmission data rate (dimension: [bps]) on a cold cables, with length from 50 m to 300 m, transferring 1024 byte packets and using "CENELEC A 36" mask.

Modulation	Stream	50 m cable	100 m cable	150 m cable	200 m cable	250 m cable	300 m cable
ROBO	OFF	6276	6241	5736	5720	5613	5591
	ON	8752	8360	8205	8160	8005	7935
BPSK	OFF	6289	6250	5744	5742	5677	5645
	ON	8754	8382	8204	8187	8137	8055
QPSK	OFF	6276	6259	5760	5769	5795	5652
	ON	8777	8383	8210	8194	8115	8062
8PSK	OFF	6375	6272	5980	5890	5859	5759
	ON	8769	8467	8416	8277	8216	8093

It is possible to observe that, on a cold electrical line, the best performance is achieved transmitting the packets (e.g., data collected by IoT devices) with the *Stream* option enabled and *8PSK* modulation.

Tests on a hot electrical line The previously described handmade 300 m electrical line has thus been connected to a real hot line, attaching the electrical cables on two distinct wall outlets, at a distance of approximately 6 m, as shown in Fig. 2.7.

Experimental tests similar to those described for cold line case were performed, increasing the length of the out-of-wall line from 50 m to 300 m, with a 50 m step. The performance results, in terms of transmission data rate (dimension: [bps]), are listed in Table 2.5. In Fig. 2.8, the corresponding throughput is shown.

As in Fig. 2.6, in this case as well it can be observed that, on a hot electrical

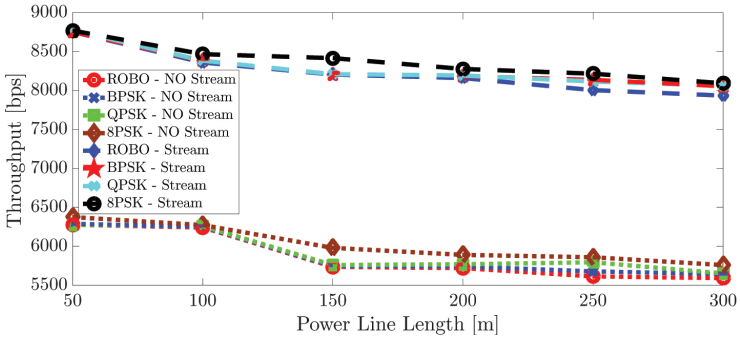


Figure 2.6: Performance obtained on a cold electrical cable, transferring 1024 byte packets and using "CENELEC A 36" mask.

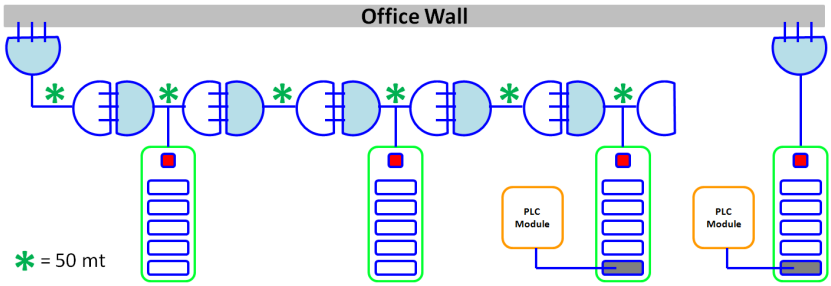


Figure 2.7: Deployment of an electrical line composed by different pieces of cable, attached into two different wall outlets.

line, the best performance is achieved by enabling the *Stream* option and selecting the *8PSK* modulation scheme. Moreover, comparing the results in Fig. 2.6 (cold electrical lines) with those in Fig. 2.8 (hot electrical lines), it can be concluded that: (i) it is advisable to transmit 1 Mbyte PLC packets, instead of 256 Kbyte packets; and (ii) there is an obvious performance degradation, due to the unpredictable noises in a real (hot) electrical line. However, even in the latter case, the transmission rate remains acceptable(namely, 6 kbps) for relevant IoT scenarios (e.g., distributed monitoring).

Table 2.5: Transmission data rate (dimension: [bps]) on a hot cables, with length from 50 m to 300 m, transferring 1024 byte packets and using “CENELEC A 36” mask.

Modulation	Stream	50 m cable	100 m cable	150 m cable	200 m cable	250 m cable	300 m cable
ROBO	OFF	4600	4601	4594	4606	4595	4601
	ON	6067	6076	6047	6078	6091	6077
BPSK	OFF	4612	4602	4580	4601	4622	4608
	ON	6077	6075	6090	6031	5916	6081
QPSK	OFF	4579	4549	4610	4604	4610	4568
	ON	6046	6071	6086	6087	6070	6081
8PSK	OFF	4620	4600	4611	4613	4592	4609
	ON	5899	6088	6086	6093	6064	6087

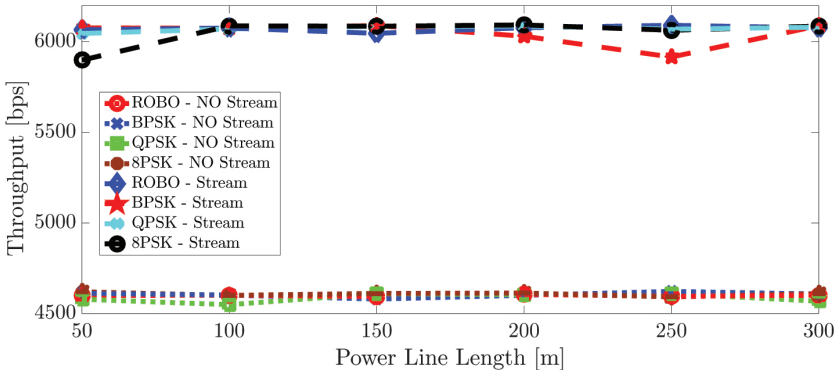


Figure 2.8: Performance obtained on a hot electrical cable, transferring 1024 byte packets and using “CENELEC A 36” mask.

2.2 IoT Backhaul

Analyzing the possibility to spread data (obtained by IoT sensors and collected by their proper SOs) over the electrical lines through the PLC paradigm, it can be concluded that a communication network built over an electrical grid is quite static and not often subject to topology changes. Contrariwise, IoT scenarios will involve, in the majority of the cases, dynamic topologies that require scalability and solutions

for efficient routing of information. Others problems and challenges that need to be addresses are: (i) problems with end-to-end IP networking to resource-constrained IoT devices, since there is a need of adaptation and management of a large number of devices with variety of IoT protocols; (ii) mismatched capability between devices, in terms of protocol stacks support, MTU differences, processing and communications bandwidth; and (iii) rapid interaction needs between services and infrastructure.

Moreover, due to the highly concentration of SOs that need to be power safer (in terms of energy consumption and processing power), it can be imagined that some external components, providing higher processing capabilities and communication interfaces, are needed to collect harvested data. These external devices can be identified with the generic term of "IoT Gateways", with different scopes, one of which is the management of routing policies internal to the managed IoT networks. An example of an IoT Gateway can be the PLC-related Base Node, whose purpose is to initialize, manage and control the IPv6 network built over the electrical grid.

In this way, it is interesting to define and deploy solutions that can help in controlling, with an high-layer view, different IoT-oriented networks.

2.2.1 Software-Defined Networking for IoT

A possible approach is represented by the adoption of the Software-Defined Networking (SDN) and the Network Function Virtualization (NFV) paradigms, which allows to decouple the data forwarding plane from the control plane, at which routing policies are defined and injected into routing elements from logically-centralized layers. Moreover, this operational decoupling is useful in handling security threats, that cannot be directly managed by constrained IoT SOs, but that need to be handled by IoT Gateways. Thus, SDN approach can represent a useful paradigm to be applied to the management of routing and data forwarding in IoT scenarios, and for the definition of smart strategies for the communication between the SOs composing different IoT environments.

Thus, the joint adoption of SDN and NFV paradigms can solve this challenges, in the way that SDN and NFV can represent enablers for IoT-oriented infrastructures [88], that may become more agile and rapid.

2.2.2 Traffic Engineering with Segment Routing

In IP networks (e.g., IoT-oriented networks), Traffic Engineering (TE) is one of the control strategies that can benefit from the SDN paradigm. By logically centralizing the control of the network, it is possible to "program" per-flow routing based on TE goals. Traditional per-flow routing requires a direct interaction between the SDN controller and each node (e.g., IoT Gateway) that is involved in the traffic paths. Depending on the granularity and on the temporal properties of the flows, this can lead to scalability issues for the amount of routing state that needs to be maintained in core IoT network nodes and for the required configuration traffic. On the other hand, Segment Routing (SR) is an emerging approach to routing that may simplify the route enforcement delegating all the configuration and per-flow state at the border of the network (e.g., an IoT Gateway that resides on the frontier of its proper IoT subnetwork). Due to the different features of TE and SR paradigms, it is interesting to center the analysis on the more powerful IoT Gateways managing the different IoT network, equalizing these devices to nodes composing a wide ISP network. Starting from this assumption, in the following it is proposed an architecture that integrates the SDN paradigm with SR-based TE, for which: (i) is provided an open source reference implementation; (ii) is designed and implemented a simple TE/SR heuristic for flow allocation; and (iii) are shown and discussed experimental results.

2.2.3 Reference Architecture for IoT Backhaul

The considered architecture corresponds to an IoT network managed by a (logically) centralized SDN controller, as shown in Fig. 2.9. Network nodes are classified in Access Routers (AR) and Core Routers (CR); in the following, both types are assumed to be MPLS nodes. As in a traditional MPLS network, AR nodes (that can correspond, as example, to IoT Gateways) are capable to originate and terminate connections (Label Switched Paths (LSPs)), while both AR and CR nodes are capable of switching labels, i.e., they can be in the middle of a path. The AR nodes are connected to different IoT-like Access Networks, which are the external traffic sources and destinations.

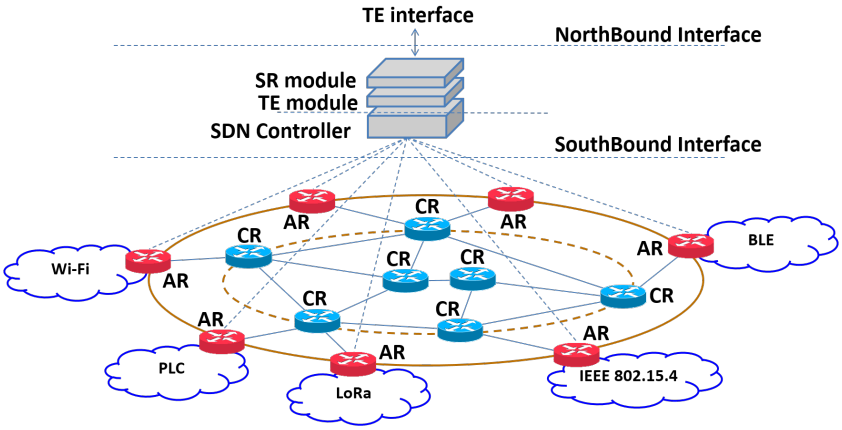


Figure 2.9: Traffic Engineering in a SDN-enabled network.

The SDN controller is in charge of setting up the *edge-to-edge* services, by configuring the ingress and egress AR nodes to support a given flow. Thanks to the SR approach, no configuration of internal nodes is needed to support a specific flow.

The local SR daemon is in charge of configuring the flow tables to support the MPLS labels representing the SIDs. The SR daemon locally interacts with the routing daemon in order to obtain routing table updates and consequently to program the switch flow tables. In this way, the SR solution relies on the functionality of the interior routing protocol (Open Shortest Path First (OSPF) in this case), complying with the requirement to minimize the enhancements of the routing protocol to support SR.

Further requirement is the coexistence of SR-based forwarding with traditional *hop-by-hop* MPLS-based LSP. This way, it is possible to offer IP Virtual Leased Line (VLL) and Pseudo-Wire (PW) services based either on *hop-by-hop* LSP or on SR forwarding.

With reference to Fig. 2.9, a SDN controller will be enhanced with TE/SR modules. The TE/SR features could be offered to applications through a proper NBI, while, on the SBI, the OF protocol is considered for the implementation. It is assumed that the SDN controller is requested to allocate a set of traffic flows with a

specified bit rate, knowing the link capacity. The TE/SR modules will first allocate *hop-by-hop* TE *paths* solving a classical *flow assignment* problem. Then, for each TE path, it will compute a corresponding *SR path* (*SR assignment* problem) for instructing the flow packets through the assigned TE path. A fundamental assumption is to execute *flow assignment* and *SR assignment* algorithm in sequence without interaction, knowing that a combined procedure could achieve further optimization of some parameters.

2.2.4 Basic Traffic Engineering Algorithm

In order to deploy a SDN controller with TE/SR modules, a classical *flow assignment* problem is considered: given a set of (unidirectional) flows between nodes with their expected rate (b/s) and given the link capacities (b/s), find the "optimal" paths for all the flows. It is assumed that there can be more than one flow between the same (source, destination) node pair, and that each of those flows can be individually routed through a different *hop-by-hop* path, without the possibility to split a single flow into multiple paths. Moreover, a feasibility/admission control check is included: a solution is admissible only if, on each link, the sum of the rates of the allocated flows does not exceed the link capacity. Therefore, the "optimal" solution could also include only a subset of the input flows.

The input of the heuristic consists in the topology, the capacity of the links, and the traffic flows, represented as triples (source, destination, bit rate). The heuristic is divided in two phases: (i) a Constrained Shortest Path First (CSPF) phase, in which a first allocation of the flows is realized (flows are rejected if they cannot be allocated in this phase); and (ii) a heuristic re-assignment phase, which tries to re-assign all admitted flows one-by-one, in order to minimize the global network crossing time T_{avg}. The second phase is executed multiple times until no improvement is achieved, in order to obtain a TE algorithm that allocates flows into *hop-by-hop TE paths* to be transformed into *SR paths*.

2.2.5 Segment Routing Assignment - First Algorithm

In order to enforce TE through SR, the *SR assignment* algorithm is performed on the TE paths obtained from the heuristic.

As explanatory example, the topology shown in Fig. 2.10, and the TE path indicated with thick arrows and composed by the node sequence $\{n_1, n_2, n_3, n_5, n_6\}$, are considered. It is assumed that there is a single IP link between two nodes, so that the sequence of nodes unequivocally identifies the path. Since the goal is to find a SR path, i.e., a list of segments that enforces the same route in the network, a suitable SR path is $\{n_1, n_3, n_5, n_6\}$, where n_3, n_5 and n_6 are node segments or Node-SIDs. The node segments are global SIDs which simply instruct an intermediate node to send the packet following the shortest path towards the node corresponding to the SID.

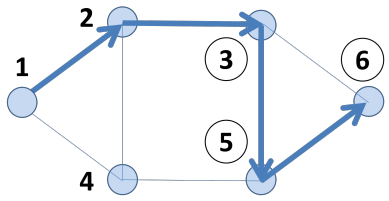


Figure 2.10: Example: network topology and TE/SR path.

In the example shown in Fig. 2.10, a sequence of node segments was able to represent the given TE paths, but this is not always the case. It is thus consider the topology represented in Fig. 2.11 and the TE path $\{n_1, n_3, n_5, n_7\}$. The SR path $\{n_1, n_3, n_5, n_6\}$ is not able to enforce the same route: node n_3 will find equal cost paths towards n_5 over the links $n_3 \rightarrow n_4$ and $n_3 \rightarrow n_5$ and node n_5 will send the packet towards n_6 on the link $n_5 \rightarrow n_6$ instead of using the link $n_5 \rightarrow n_7$. In the SR architecture, this issue is solved with the use of Adj-SID. This segment needs to be advertised by the nodes and can be of local or global significance. For example, it is possible to respectively represent with SIDs $l_{3>5}$ and $l_{5>7}$ the segments corresponding to the direct links $n_3 \rightarrow n_5$ and $n_5 \rightarrow n_7$. Using Adj-SIDs with local significance, a suitable SR path is: $\{n_1, n_3, l_{3>5}, l_{5>7}\}$. In principle, it is also possible to use Adj-SIDs with global significance, leading to the following shorter SR path: $\{n_1, l_{3>5}, l_{5>7}\}$. The price is

the need to install one additional rule in all nodes for each global Adj-SID to enforce the proper routing.

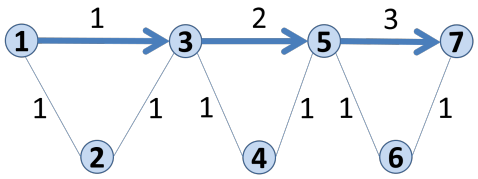


Figure 2.11: Network topology in which SR node segments are not able to enforce arbitrary TE paths (one example shown in the figure).

The use of Adj-SIDs is not desirable for the following reasons: the dissemination of Adj-SIDs introduces complexity and is based on enhancements to Interior Gateway Protocol (IGP) protocols; if local significance SIDs are used, the handling of node/link failures is more complex or less efficient; if global significance SIDs are used, the state information in nodes grows and the scalability decreases. For this reasons, it is needed to propose a solution with a new type of SID that has global significance but does not need to be disseminated, as it can be automatically derived and handled by all nodes. This type of SID is denoted as "*direct-link SID*" (*DL-SID*) and represented with the notation $n_k{}^*$. A *DL-SID* instructs a node to send the packet on the direct link toward a target node, if the direct links exists. Otherwise, the regular shortest path forwarding towards the target node is used. The *DL-SID* $n_k{}^*$ has global significance and, in particular, it can be used by different neighbors of a node n_k to send the packet on the direct link towards n_k, even if this link is not the shortest path toward n_k.

Note that there are topological conditions under which only node SIDs are enough to enforce any TE path, and the use of *DL-SID*s is not needed. In particular, it is needed to assume that when a direct link exists between two nodes, it matches the shortest path between those nodes. This is easily verified in case all links have the same cost, which constitutes a sufficient condition for not having to use the *DL-SID*s.

The objective of the *SR assignment* algorithm is to find the minimal-length SR

paths corresponding to the TE paths, i.e., the shortest list of SIDs that allows the packets to follow the assigned TE path.

In the following, the Node-SID associated to a node n_k will be simply referred through the name of the node n_k, while the *DL-SID* for n_k is represented with ${n_k}^*$. Thus, it can be supposed that a given flow f, characterized by the ingress AR node I and the egress AR node E, is requested to be set up with an assigned *hop-by-hop* TE path, specified by the complete list of all intermediate nodes R_1, R_2, ... , R_{N-1}.

The complete *hop-by-hop* TE path is:

$$P = \{R_0{=}I, R_1, R_2, ... , R_{N-1}, R_N{=}E\}$$

The *SR assignment* problem consists in finding a sequence of SIDs that can be pushed into a packet by the ingress AR node I, forcing the packet to follow a given route; this sequence of SIDs is the SR path of the packet. In the following a *SR assignment* algorithm is proposed, using the following notation:

- G: the graph of the network;
- P: the assigned TE path;
- $tep(n_s, n_d)$: portion of the TE path starting from node n_s and ending with node n_d. As particular case, $tep(I, E) \equiv P$;
- $SP(n_s, n_d) = \{spi(n_s, n_d),\ i = 1, ..., M\}$: the set of equal-cost shortest paths $spi(n_s, n_d)$ in G from n_s to n_d, based on the current routing tables; routing tables are considered to be already set up using a link-state routing protocol (e.g. OSPF), using Shortest Path First algorithm;
- $SP^*(n_s, n_d) = \{spi^*(n_s, n_d),\ i = 1, ..., M^*\}$: the set of *direct-links biased* equal-cost shortest paths in G from n_s to n_d;
- $prec(p, n)$: the preceding node of node n along a path p;
- srp: the SR path containing the list of assigned SIDs.

The pseudo-code representation of the *SR assignment* algorithm is reported in the Algorithm 1. The algorithm takes as input the graph of the topology and the assigned TE path, and returns as output the assigned SR path (the segment list). It starts by considering as current source node n_s the ingress node I of the TE path and as current target node n_d the egress node E of the TE path. Then, the set of equal-cost shortest paths from n_s to n_d is considered. If only one shortest path exists and it equals the TE path (from n_s to n_d), then the node n_d is used as Node-SID from n_s to n_d, and the algorithm ends. On the contrary, if the number of equal-cost shortest paths is greater than 1 or the shortest path differs from the TE path, the set of *direct-links biased* equal-cost shortest paths is considered. If the size of the set is 1 and the only path equals the TE path (from n_s to n_d), then the node n_d is used as *DL-SID* from n_s to n_d and the algorithm ends. On the contrary, if the two paths differ or the size of the set is greater than 1, no single SID exists for the entire path from n_s to n_d, and the previous procedure is repeated considering as target node the preceding node of n_d (that is new n_d=$prec(p, n_d)$). If it succeeds in finding a SID, then the SID is added to the segment list and the procedure is repeated from the current n_d to E (that is n_s=n_d, n_d=E); otherwise, if it fails, n_d is replaced with the preceding node (as above), and the procedure is repeated. The algorithm ends when a SID is found with n_d=E. As explanatory example, the topology of Fig. 2.10 can be considered, and supposing that the goal is to find a SR path for the TE path p=$tep(n_1, n_6)$=$\{n_1, n_2, n_3, n_5, n_6\}$. Since there are more than one shortest between n_1 and n_6, the sub-path p=$tep(n_1, n_5)$=$\{n_1, n_2, n_3, n_5\}$ is considered. Since the shortest path between n_1 and n_5, that is $sp(n_1, n_5)$=$\{n_1, n_4, n_5\}$, differs from p, the sub-path p=$tep(n_1, n_3)$=$\{n_1, n_2, n_3\}$ is considered. Now, since p coincides with the shortest path $sp(n_1, n_3)$, n_3 is added to the SR path and the algorithm restarts between n_3 and n_6. At the end, the returned SR path will be srp=$\{n_1, n_3, n_5, n_6\}$. In this example, all the steps can be enforced with Node-SIDs (as it was expected, considering that all links have the same unitary cost).

In the implementation, there is an automatic derivation of both the global Node-SID and the *DL-SID* from the loopback interface address that can univocally identify a (MPLS) router in an ISP network. This is an important simplification, as there is no

need to distribute SIDs through extensions of the routing protocols.

Algorithm 1 Second phase of the algorithm.

```
function SEGMENTALLOCATION(G, P)
    n_s = I, n_d = E, srp = { };
    BEGIN:
    p = tep(n_s, n_d);
    //check if p is equal to the (only one) shortest path
    if ((sizeof(SP(n_s, n_d)) == 1) AND (sp_1(n_s, n_d) == p)) then
        ADD n_d to srp;
        if n_d == E then GOTO FINISH;
        else
            n_s = n_d, n_d = E, GOTO BEGIN;
        end if
    else
        //p is not the (only one) shortest path, try with direct-links
        if ((sizeof(SP*(n_s, n_d)) == 1) AND (sp_1*(n_s, n_d) == p)) then
            ADD n_d* to srp;
            if n_d == E then GOTO FINISH;
            else
                n_s = n_d, n_d = E, GOTO BEGIN;
            end if
        else
            //p is not the (only) direct-link shortest-path,
            //try with a shorter segment
            n_d = prec(p, n_d), GOTO BEGIN;
        end if
    end if
    FINISH: return srp;
end function
```

2.2.6 Implementation

The proposed architecture and the algorithms previously described have been implemented and tested in conjunction with the University of Rome "Tor Vergata". Fig. 2.12 shows the developed software components and their interactions, whose source code has been made freely available [89]. The topology parser can acquire topology information from the REST API of Ryu controller, from the GraphML representation used in the Topology Zoo project [90], or from the self-made Topology 3D GUI [91]. A random demand generator can be used to create a list of traffic flows between AR nodes (or the list of flows can be manually created with the Topology 3D GUI). Two Java modules implement the *flow assignment* heuristic and the *SR assignment* algorithm. Their output is a list of SR paths. Both the input to the *flow assignment* heuristic and the output of the *SR assignment* algorithm are represented as JSON files [89]. The SR paths list is processed by the *OSHI_SR_pusher* Python script. Using the Ryu REST API, this script configures the ingress and egress operations in AR OSHI nodes. The data plane is emulated in Mininet [92], with OSHI nodes extended by the OSHI-SR-dataplane [91].

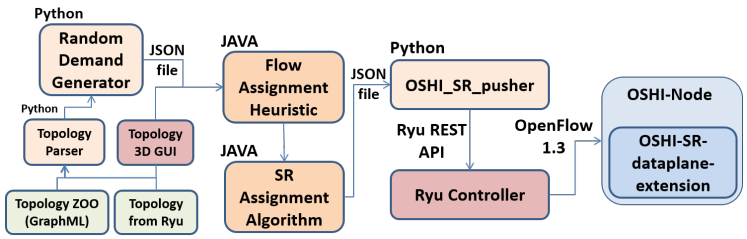


Figure 2.12: Software components of the implementation.

2.2.7 Evaluation of the First Segment Routing Assignment Algorithm

An experimental analysis of the proposed architecture and algorithms has been carried out, with the following two main goals:

- testing the *SR assignment* algorithm;

- testing the overall implementation of the solution, from the SDN-based control plane to the MPLS-based SR forwarding in the data plane.

To accomplish the first goal, a relatively large scale topology with 153 nodes and 354 unidirectional links, the "Colt Telecom" topology, included in the Topology Zoo dataset, has been considered, and assuming that all links have the same capacity. Thus, a random demand generator has been developed as follows. It randomly selected 40% of the nodes to be AR (e.g., ingress/egress), then it randomly selected 20% of the AR couples to be active source/destination of traffic flows. For each active couple of ARs, in each direction there is an average of 3.5 flows (the number of flows has a geometrical distribution) with the sum of the flow rates equal to 10% of the capacity of a link and the size of each flow has a negative exponential distribution. With these parameters, the random generator produced a list of 2460 flows with their bit rate.

As shown in Fig. 2.13, it has been run a set of allocation experiments by selecting an increasing number of flows out of the full demand of 2460 flows and it has been collected the number of allocated flows (the flows are selected following the same order). The straight line shows the ideal case with infinite link capacity.

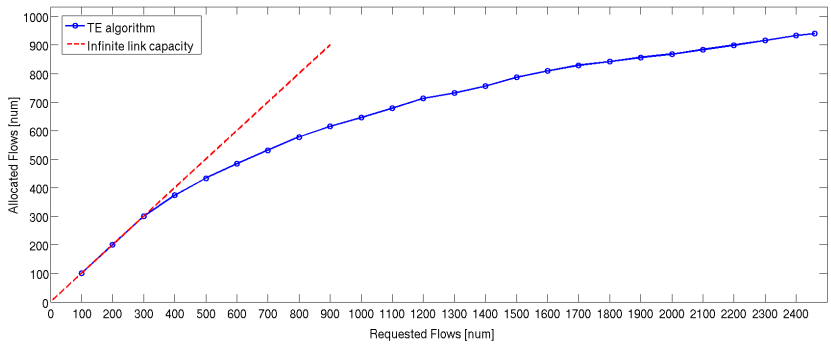


Figure 2.13: Allocated flows vs. requested flows.

This demand has been generated to largely overcome the network capacity, so that only a subset of the flows can be allocated. In this condition, the flow allocation algorithm has been stressed, obtaining TE paths that diverge from the shortest path.

In the first experiment, the full set of requested flows (2460) is considered, resulting in the allocation of 940 flows. For each allocated flow, the TE path and SR path have been computed according to the *flow assignment* heuristic and *SR assignment* algorithm. Fig. 2.14 reports the distribution of path lengths for the TE paths and for the "natural paths". The term "natural path of a flow" refers to the shortest path from ingress AR to the egress AR (with no capacity constraints). As expected, TE paths are longer than natural paths (in this experiment the mean length is 8.17 vs 6.63).

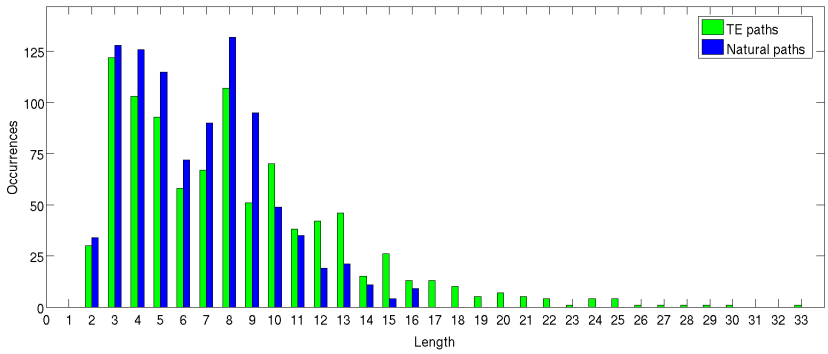


Figure 2.14: Distribution of path length for TE and natural paths.

The result of the *SR assignment* algorithm is shown in Fig. 2.15, which provides the distribution of SR path lengths. It is interesting to note that for more than 50% of the flows the SR path has only one SID , i.e., the egress node (this means that the flow is surely allocated on a natural path). When the SR path has more than one SID, it can be a natural path in which it was needed to differentiate among multiple equal cost shortest path, or a TE path different from the natural path. It is interesting to note that the mean number of hops in a SR path is low: the average number is 1.7 and the maximum number is 6.

The *flow assignment* and *SR assignment* algorithms have been also tested varying the number of allocated flows. In Fig. 2.16 is shown the mean length of TE paths, natural paths, and SR paths for different number of allocated flows. Increasing the number of flows (e.g., the traffic load), the mean length of the TE paths increases from ~8.2 to ~9.7. When the number of allocated flows further increases, the mean

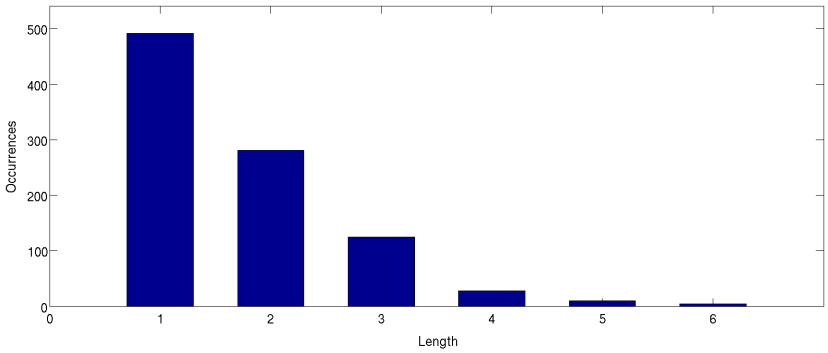


Figure 2.15: Distribution of SR path length (number of SIDs in the path).

length starts to decrease. This depends on how the sets of allocated flows starting from the full list of requested flows (2460) has been selected. As shown in Fig. 2.13, larger subsets from the full list has been extracted and then selected the admitted flows from this subset. This creates a bias in the selected flows: when there have already admitted several flows and loaded the network, the flows with distant ingress and egress ARs can hardly be allocated, therefore the flows that are accepted are on average shorter. The mean number of SR-hops starts from ~1.3 when the network is not loaded and TE paths correspond to natural paths, because of the segments used to differentiate among multiple equal-cost shortest paths.

Moreover, it has been made another simple experimental evaluation of the processing time of the proposed TE/SR algorithms. Fig. 2.17 reports the time spent for the computation of TE paths (*flow assignment* heuristic) and of SR paths (*SR assignment* algorithm). It has been used a PC with an Intel Core i7 @ 2Ghz and 6GB RAM. Note that processing time of the *flow assignment* heuristic depends on the number of iterations of the optimization cycle, therefore a set of seemingly parallel lines can be appreciated in the Fig. 2.17 (each one corresponds to a given number of cycles).

The processing time of the *SR assignment* algorithm is negligible with respect to the *flow assignment* heuristic. In the considered range (up to 900 admitted flows) it was possible to run both algorithms and allocate the flows in less than 8 seconds. This performance seems adequate for periodic (e.g. nightly) reallocation procedures

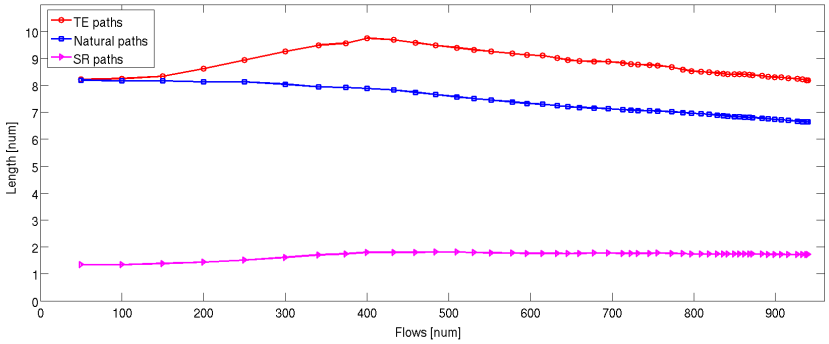


Figure 2.16: Mean lengths of TE paths, natural paths, and SR paths vs. the number of allocated TE flows.

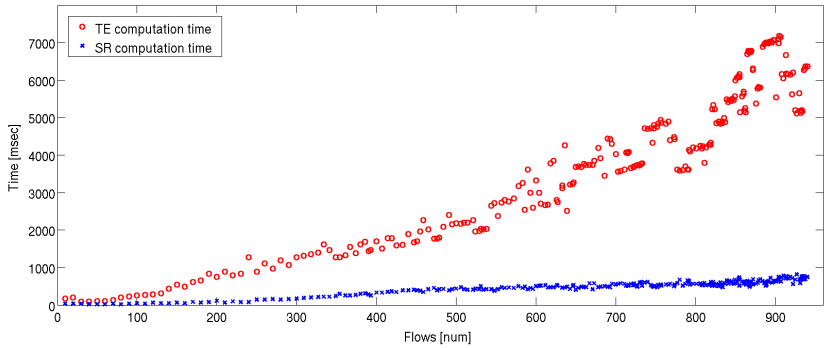


Figure 2.17: Execution time of the algorithms.

that aim to evenly redistribute the load on the network links.

2.2.8 Poor Man's Segment Routing

As previously highlighted, in its current specification, the SR architecture proposed in [47] requires enhancements to routing protocols (e.g., [93, 94]) in order to distribute the SIDs across the networks. Thus, there is a certain interest in finding a minimalistic approach that does not need to explicitly distribute information among nodes and hence does not require enhancements to the routing protocols. This solu-

tion has been defined and referred to as "Poor Man's Segment Routing" (PMSR), and it can efficiently support the large majority of the use cases of traditional SR [95, 96]. As for the previously reported algorithms, also in this case PMSR has been verifies considering a TE use case, starting from a traditional *flow assignment* optimization problem which allocates *hop-by-hop* paths to flows.

The main purpose of PMSR algorithm is to avoid the distribution of SIDs by the SR nodes, as it implies significant extensions to the routing protocols and to the routing daemons implementing these protocols. For this reason, the algorithm only uses global segments types whose SIDs can be automatically generated by each node in a distributed fashion, with no need of explicit advertising (and no extensions to routing protocols). The automatic generation avoids the need of node management procedures for SID assignment, and a significant coverage of the SR use cases is advocated to be achieved by only using global segments that can be automatically generated.

In order to avoid the use of local Adj-SIDs, the adoption of *DL-SID*s is proposed, which identifies the ways in which a target destination node can be reached: if a node has a direct link toward the destination node, the *DL-SID* forces the node to use the direct link rather than the shortest path dictated by the routing protocol; conversely, if a node does not have a direct link toward the target node, it will process the segment in the same way it processes a Node segment toward the same destination node.

A *DL-SID* needs to identify the target node, like the Node-SID, and to carry further information that identifies it as *DL-SID*. When using MPLS as Data Plane, the *DL-SID* can be obtained by the Node-SID adding a bit to distinguish between *DL-SID* and Node-SID. When using IPv6 as Data Plane, *DL-SID*s are IPv6 addresses globally valid in the network domain. They need to be derived in a deterministic way from the loopback interfaces addresses used as Node-SIDs. As an example, Node-SIDs can be restricted to have an odd numbered Device address part of the IPv6 address, so that the *DL-SID*s will be even numbered, obtained by adding one to the Device address part of the IPv6 address of the localhost interface.

A limitation of the approach based on *DL-SID* is that it does not allow the handling of multiple parallel links between two routers at L3 (i.e., with different IP ad-

dresses). If present, such *multi-links* must be handled at L2 and seen at IP level as a single link. Having multiple parallel links bonded at L2 is anyway a typical solution for operators, so in IoT-oriented scenarios this may not be a critical limitation.

There are advantages in using the automatically generated global *DL-SID*s rather than the local Adj-SIDs or the global Adj-SID . Consider the *strict source routing* case, that is enforcing a path through a set of links. Using local Adj-SIDs, the segment list will have a length equal to the double of the number of links. In fact, for each link to be crossed, first the source node needs to be addressed, then the local segment will indicate the outgoing link. Using global Adj-SIDs, the list will be equal to the number of links, but the global Adj-SIDs needs to be advertised and one entry for each advertised Adj-SIDs needs to be added in the routing state of all nodes. In PMSR, with automatically generated global *DL-SID*s the length of a segment list to enforce a path through a set of links also equals the number of links (like with the global Adj-SID), but there is no need to advertise SIDs, and in each node it is only needed to add an additional entry for each node instead that for each link. Let η be the number of nodes and k be the number of unidirectional links; in the worst case, $k = O(\eta^2)$.

2.2.9 Segment Routing Assignment - Second Algorithm

Problem Definition and Heuristic Resolution

Let F be a set of unidirectional flows $f_i(s_i, d_i, r_i)$, in which s is the source node, d the destination node and r the nominal bandwidth requirement (b/s); let $T(N, E)$ be a directed graph representing the topology, N is the set of nodes and E is the set of directed edges. An edge e_j can be represented as $e_j(u_j, v_j, c_j)$, where u_j is the source node, v_j the destination node and c_j the edge capacity (b/s). An edge can also be denoted simply as $e(u, v)$, where u is the source and v the destination. Each flow f_i needs to be mapped into an *hop-by-hop* path P_i that can be represented as the set of intermediate nodes from source s to destination d (denoted as Pn_i), or equivalently by the set of links (Pe_i):

$$Pn_i = \{n_{i0}{=}s, n_{i1}, n_{i2}, \ldots . , n_{iN-1}, n_{iN}{=}d\}$$

$$Pe_i = \{e_{i1}, e_{i2}, ... , e_{iN-1}, e_{iN}\}$$

where:

$$e_{i1}{=}e(s, n_{i1}), e_{i2}{=}e(n_{i1}, n_{i2}), ... , e_{iN}{=}e(n_{iN-1}, d)$$

The traditional *hop-by-hop* path assignment consists in finding an "optimal" set of paths $\{P_i\}$ (i.e., a set chosen according to an optimality criterion). Let the flow mapping variables be defined as a_{ij}, which tells if flow f_i is mapped over link e_j:

$$a_{ij} = 1 \text{ if } e_j \in Pe_i, a_{ij} = 0 \text{ if } e_j \notin Pe_i$$

A feasibility check is also included in this formulation: the sum of the nominal flow rates of the flows crossing a link needs to be smaller than the link capacity. In symbols:

$$\forall \; link \; j : \sum_i a_{ij} \cdot r_j < c_j$$

For the experiments there will be reused (with few changes) the definition of the *flow assignment* problem and the heuristic for its resolution originally previously detailed [97, 98]). The problem formulation is very effective in equalizing the load of the links in the network and avoiding critical bottleneck. In addition, the heuristic provides a good trade-off between computation time and optimality of results. Anyway, the interest is in the adoption of a set of *hop-by-hop* path allocated by the TE algorithm, as input, and considering their mapping into SR paths.

Mapping Hop-by-Hop Paths into Segment Routing Paths

A SR path will be denoted as S_i and represented as a sequence of SIDs Sn_i:

$$Sn_i = \{n_{i0}{=}s, n_{i1}, n_{i2}, ... , n_{iN-1}, n_{iN}{=}d\}$$

In PMSR, as previously detailed, each SID can be a Node-SID or a *DL-SID* (in the traditional SR architecture, a SID can also be a local or global Adj-SID, corresponding to an outgoing adjacency). A Node-SID is simply represented by the node name n_x, while the corresponding *DL-SID* is represented as $n_x{}^*$. In both cases, the SID corresponds to a node that needs to be crossed before reaching the destination node.

Two consecutive nodes in a SR path Sn_i do not need to be adjacent as it is for Pn_i. When two consecutive nodes are not adjacent, the links that will be crossed depend on the underlying IP routing. If all the shortest paths from a given node toward the next node in the SR path insist on the same output link, then the output link is univocally determined. If there are multiple shortest paths and they insist on different output links, then the output link is not univocally determined. In this case, two options are possible, depending on the configuration of the router. If Equal Cost Multi Path (ECMP) is enabled, all the "candidate" output links that are part of a shortest path towards the next node in the SR path are considered (typically they are selected based on a hash function over the port numbers of the transport protocol, in order to balance the traffic). If ECMP is not enabled, one of the candidate output links is arbitrarily selected by the node. In both cases, such type of segment is not applicable to the classical TE approach, in which the network operator wants to deterministically route a flow over a given path.

A SR path is congruent to a *hop-by-hop* path if the route enforced by the SR path is deterministically equivalent to the one enforced by the *hop-by-hop* path. To provide examples of *hop-by-hop* paths, of congruent SR paths, and of the use of *DL-SID*s, consider the network topology depicted in Fig. 2.18 and the two *hop-by-hop* paths P_1 and P_2 that are represented using P_n notation as:

$$Pn_1 = \{n_1, n_3, n_5, n_7\}$$
$$Pn_2 = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7\}$$

The only SR path congruent to the *hop-by-hop* path P_1 is:

$$Sn_1 = \{n_1, n_3, {n_5}^*, {n_7}^*\}$$

in which three segments are needed, and the *DL-SID*s ${n_5}^*$ and ${n_7}^*$ are respectively used to select the links 3→5 and 5→7.

There are multiple SR paths that are congruent to the *hop-by-hop* path P_2; a subset of them is listed hereafter (they only contain Node-SIDs):

$$Sn_{2-a} = \{n_1, n_2, n_4, n_7\}$$
$$Sn_{2-b} = \{n_1, n_2, n_3, n_4, n_7\}$$

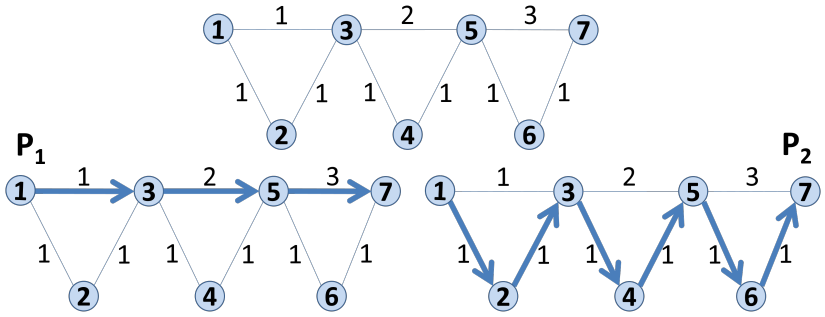


Figure 2.18: A network topology and two *hop-by-hop* paths.

$$Sn_{2-c} = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7\}$$

Among them, Sn_{2-a} is the optimal SR path, in the sense that it has the minimum number of segments.

Optimal Segment Routing Assignment Procedure

In the *SR assignment* problem, given a *hop-by-hop* path P, the purpose is to find a congruent SR path S composed of the minimum number of segments. Hereafter it is proposed an efficient algorithm for the *SR assignment*, both for traditional SR and for the proposed PMSR, thus proving that the algorithm finds the optimal solution, i.e., the shortest list of SIDs that allows the packets to follow the assigned *hop-by-hop* path, according to the default IP routing tables of the nodes. In order to detail the *SR assignment* algorithm, the following notation (similar to those previously detailed) is defined.

- f: a single traffic flow from node s to node d, characterized by its *hop-by-hop* path P_n:

$$P_n = \{n_0{=}s, n_1, n_2, \ldots, n_{N-1}, n_N{=}d\}$$

- $tep(x, y)$: portion of the *hop-by-hop* path starting from node x and ending with node y. As particular case, $tep(s, d)$ is the complete *hop-by-hop* path from s to d;

- $SPN(x, y)$: the number of equal-cost shortest paths from x to y, based on the current routing tables that are considered to be already set-up by a link-state routing protocol (e.g. OSPF), using Shortest Path First (SPF) algorithm;
- $sp(x, y)$: the set of the shortest paths from x to y; if $SPN(x, y) \equiv 1$, it is the shortest path from x to y;
- $prec(p, x)$: the preceding node of x along a path p;
- $succ(p, x)$: the succeeding node of x along a path p;
- srp: the SR path containing the list of assigned SIDs;
- $sp^*(x, y^*)$: the set of *direct-links biased* shortest paths from x to y^*; a *direct-links biased* shortest path is built heading from x to y on a shortest path, unless there is a direct link from an intermediate node to y, which is always followed;
- $SPN^*(x, y^*)$: number of *direct-links biased* shortest paths $sp^*(x, y^*)$.

A pseudo-code representation of the *SR assignment* algorithm for the traditional SR architecture is detailed in Algorithm 2, in which *T_SRP* stands for Traditional SR Path. The algorithm takes as input the topology and the assigned *hop-by-hop* path, and returns as output a congruent "optimal" SR path. At each step, a *hop-by-hop* sub-path between two nodes x and y is compared with the shortest path between the same pair of nodes. At the beginning x=s and y=d. If there is only one shortest path and it matches the *hop-by-hop* sub-path, y is added to the SR path. Otherwise (i.e., if there is more than one shortest path or the shortest path does not match the *hop-by-hop* sub-path), if the sub-path $tep(x, y)$ between x and y is just one link, then it means that there is a direct link between x and y different from the shortest path; in this case the Adj-SID corresponding to the link $e(x, y)$ is added to the SR path. If $tep(x, y)$ is more than one link, the procedure repeats with y set to the node that precedes the old y. If a segment has been added, it is checked if $y \equiv d$, in which case the procedure ends and the SR path is returned; otherwise, if $y \neq d$, the algorithm considers the remaining part of the path, from y to d. For each direct link different from the shortest path, this

algorithm will add two segments in the SR path: the preceding node and the Adj-SID representing the outgoing link.

Algorithm 2 Pseudo-code of SR path assignment for traditional SR.

```
function T_SRP(tep(s,d))
    x = s; y = d; srp = { };
    START:
    p = tep(x, y);
    //check if the sub-path p is the only shortest path
    if ((SPN(x, y) == 1) AND (sp(x, y) == p)) then
        ADD y to srp; GOTO ADDED;
    else
        //check if the sub-path p is just one link
        if (prec(p, y) == x) then
            ADD Adj-SID of e(x, y) to srp; GOTO ADDED;
        else
            //no segment added, try with a shorter path
            //(from x to the node that precedes y)
            y = prec(p, y); GOTO START;
        end if
    end if
    ADDED:
    if (y != d) then
        //consider the remaining part of the path
        x = y; y = d; GOTO START;
    end if
    return srp;
end function
```

The *DL_SRP* algorithm detailed in Algorithm 3 takes as input the SR path (that includes Adj-SIDs) computed by *T_SRP* and returns, as output, a SR path that includes only Node-SIDs and *DL-SID*s. When possible, it replaces a couple of Node-

SID + Adj-SID with a single *DL-SID*. When a single *DL-SID* is not enough to enforce the required *hop-by-hop* path, the algorithm will leave a couple Node-SID + *DL-SID*. The algorithm inspects step-by-step the SR path and replaces any Adj-SID with the corresponding *DL-SID*. The Node-SID that precedes the Adj-SID is kept only when required, that is when there is more than one *direct-links biased* shortest path from the node that precedes the current Node-SID and the successive *DL-SID*, or if such a *direct-links biased* shortest path differs from the *hop-by-hop* path.

Algorithm 3 Replacement of Adj-SIDs with *DL-SID*s.

```
function DL_SRP(srp)
    dlsrp = { };
    for (i = 0; i < srp.length; i++) do
        if srp[i] is an Adj-SID then
            d = destination of srp[i]; ADD d* to dlsrp;
        else
            if (srp[i+1] is NOT an Adj-SID) then
                ADD srp[i] to dlsrp;
            else
                if ((SPN*(srp[i-1], srp[i+1]) > 1 OR
                  sp*(srp[i-1], srp[i+1]*) != tep(srp[i-1], srp[i+1])) then
                    ADD srp[i] to dlsrp;
                end if
            end if
        end if
    end for
    return dlsrp;
end function
```

Optimality of the Segment Routing Assignment Algorithm

In order to demonstrate the optimality of the SR path assignment, it is needed to define the following Lemmas.

***Lemma 1*:** if there is a unique shortest path from s to d, then there is a unique shortest path from s towards all intermediate links in the path from s to d (it can be easily proven by contradiction).

***Lemma 2*:** if it does not exist a unique shortest path from y to d, then it does not exist a unique shortest path from a node x to d that passes through y (it can be easily proven by contradiction).

Focusing on the *T_SRP* algorithm, consider the *hop-by-hop* path P_n.

$$P_n = \{n_0{=}s, n_1, n_2, ... , n_{N-1}, n_N{=}d\}$$

Assume that the directed edge from n_{k-1} to n_k is not the shortest path from n_{k-1} to n_k (or it is one of a set of equal-cost shortest paths), then an Adj-SID is needed to enforce the use of the link $e(n_{k-1}, n_k)$. Under this hypothesis, starting from s the *T_SRP* algorithm can find one or more segments up to n_{k-1} (the last segment being n_{k-1} itself), but then it will identify the link that requires the Adj-SID (the first check "if the sub-path p is the only shortest path" fails and the second check "if the sub-path p is just one link" is verified) and adds it. This happens for all the links that are not the shortest path between their source and destination. In the end, the SR path will be composed at least by all the Adj-SIDs, needed in order to route the packets on links that are, by definition, off the shortest path dictated by the routing protocol. Each Adj-SID will be preceded in the SR path by the Node-SID of the node that originates the link that requires the Adj-SID. Now it is needed to demonstrate that the number of segments, selected by the algorithms in any portion of the *hop-by-hop* path that does not need to include Adj-SIDs, is the minimum possible. Assume from now on to be in a portion of the *hop-by-hop* that does not need to include Adj-SIDs (i.e., all links correspond to the only shortest path between source and destination of the link). The *T_SRP* algorithm starts from the source s and tries to find the longest portion of the *hop-by-hop* path $P{=}tep(s, d)$ that corresponds to a shortest path. If it arrives to the destination d, then the solution is optimal. If it stops at an intermediate node x, this means that $tep(s, x)$ is a unique shortest path, while $tep(s, succ(P, x))$ is not a unique shortest path. The algorithm tries to find segments from x to d. If there is a unique shortest path from x to d, then the algorithm has found a SR path with two segments:

$\{s, x, d\}$. This is optimal, as a solution with one segment does not exist (it is known that $tep(s, succ(P, x))$ is not a unique shortest path and, by *Lemma 1*, there cannot be a unique shortest path from s to d). If the algorithm finds that an intermediate node y is needed from x to d, then a three segments solution is needed: $\{s, x, y, d\}$, and it is proved that it is not possible to find a two segments solution $\{s, z, d\}$ for any z in P. In fact, the segment z can not be after x by construction. It cannot be before x because, by *Lemma 2*, there cannot be a unique shortest path from z to d passing through x. This reasoning can be extended to any number of segments: each time that the algorithm introduces a segment, it is not possible to find a solution with a smaller number of segments.

It is easy to prove that the *DL_SRP* algorithm is optimal as well. In fact, it includes one *DL-SID* for each Adj-SID (they correspond to the minimum number of segments). In each portion of the path without Adj-SID, the algorithm verifies if it is possible to reduce the segments eliminating the last Node-SID and using only the *DL-SID*.

2.2.10 Evaluation of the Second Segment Routing Assignment Algorithm

The PMSR solution and TE algorithms have been implemented and evaluated considering the same relatively large scale topology of the previously detailed evaluation, the "Colt Telecom" topology which is included in the Topology Zoo dataset, assuming that all links have the same capacity.

Fig. 2.19 reports the time spent for the computation of TE paths (*flow assignment* heuristic) and of SR paths (*SR assignment* algorithm) using a PC with an Intel Core i7 @ 2Ghz and 6GB RAM. Note that processing time of the *flow assignment* heuristic has a step-wise dependence on the number of iterations of the heuristic optimization cycle, which tends to increase with the number of flows.

Therefore a set of seemingly parallel lines can be appreciated in the figure (each one corresponds to a given number of cycles). As it is possible to see from the figure, the processing time of the *SR assignment* algorithm is negligible with respect to the *flow assignment* heuristic. In the considered range (up to 900 admitted flows) it was

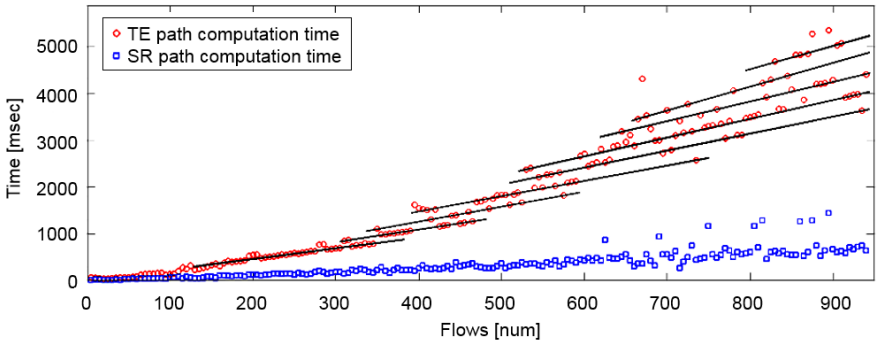


Figure 2.19: Execution time of the algorithms.

possible to run both algorithms and allocate the flows in less than 8 seconds. This performance seems adequate for periodic (e.g., nightly) reallocation procedures that aim to evenly redistribute the load on the network links, and confirm the goodness of the adoption of TE and SR paradigms in conjunction for SDN-based approaches, as for IoT-oriented ones.

Chapter 3

Application Layer for IoT

3.1 CoSIP Protocol for IoT

It is a common opinion that in the near future, IP (and in particular IPv6) will be the common network protocol for the IoT. At the same time, significant reasons for proper protocol optimizations and adaptations for resource-constrained objects are required. In particular: (i) the use of compressed protocols can significantly reduce the fragmentation and postponed transmissions; (ii) processing larger packets is associated with higher energy consumption, which can be a critical issue for battery-powered devices; and (iii) minimized versions of protocols (at all layers) can reduce the number of exchanged messages.

The Session Initiation Protocol (SIP) is the standard application protocol for establishing application-level sessions. It allows the endpoints to create, modify, and terminate any kind of (multi)media sessions, such as Voice over IP (VoIP) calls, multimedia conferences, or data communications. It also already includes mechanisms for subscribe/notify communication paradigms [99] and for resource discovery, particularly useful in IoT scenarios, for which proper CoAP extensions are currently being specified [100, 101]. The main drawback of using standard SIP protocol in constrained environments is the large size of text-based SIP messages (compared to other binary protocols such CoAP), and the processing load required for parsing

such messages. For these reasons, a constrained version of SIP, denoted as CoSIP, whose intent is to allow constrained devices to instantiate communication sessions in a lightweight and standard fashion and can be adopted in IoT application scenarios such as service discovery, distributed networks, and publish/subscribe applications.

In order to reduce the message size of text-based signaling protocols, especially when using low-bandwidth wireless links, the IETF defined the Signaling Compression (SigComp) protocol [102]. SigComp works as an intermediate layer between the application (signaling) and the underlying transport layer (e.g., UDP, TCP, or Stream Control Transmission Protocol (SCTP)). SigComp takes application messages from the signaling protocol, compresses them, and passes the compressed version to the transport protocol. At the receiver side, the compressed messages are passed to a SigComp component, the Universal Decompressor Virtual Machine (UDVM), that operates decompression according to the algorithm used by the sender. Depending on the signaling protocol in use, chosen compression algorithm, memory capacity, and the availability of a pre-loaded dictionary, different compression performance can be obtained. The main drawback of such an approach is that it has been mainly designed for being used by wireless handsets in wireless 2.5G and 3G cellular networks, where the main constraints come from narrow-band communication resources. When dealing with more constrained devices (e.g., SOs), which are limited also in memory and processing capabilities, other aspects should be considered, such as memory footprint, processing load, and energy consumption.

During last decades, Peer-to-Peer (P2P) networks have been deeply analyzed and several overlays and different architectures have been designed for different purposes and application scenarios [103]. P2P networks typically provide some desirable and unique features for large-scale systems, comprising a huge number of nodes, such as scalability, fault-tolerance, and self-configuration. One of the main features that makes P2P networks appealing is the fact that as the number of participating nodes increases, the overall capacity of the system increases as well. These features are very appealing for IoT-like scenarios, where they are required to support large-scale and robust applications.

IoT application scenarios represent a new challenging opportunity of redemption

and renaissance for P2P. In order to design and deploy new P2P overlays involving SOs, it is crucial to properly identify the right protocol that has to be used at the same time by constrained and traditional peers. It is reasonable that the same kind of optimization that has characterized the IoT development should be applied also to P2P IoT overlays. For these reasons, it is interesting to perform an experimental evaluation of a Distributed Hash Table (DHT)-based P2P overlay implemented using both SIP and CoSIP protocols in order to understand the benefits of a constrained solution in terms of bandwidth consumption and transmitted/received data.

3.1.1 CoSIP and Distributed Location Service

As previously introduced, CoSIP is a binary protocol which maps to SIP, similarly to what CoAP does to HTTP [104]. In order to avoid a complete re-design of a session initiation protocol for constrained environments, the layered architecture of SIP has been reused, by re-defining the messaging layer with a constrained-oriented binary encoding by reusing the same CoAP message syntax [105]. A CoSIP message can be virtually seen as a standard SIP message, consisting of one request-line or one status-line (depending if the message is a request or a response), followed by a sequence of SIP header fields, followed by a message body, if present. In particular, SIP header fields are logically the same as those of the standard SIP protocol, properly encoded in corresponding CoSIP "Options" fields. For each SIP header field, a different option number has been set, and a proper encoding mechanism has been defined.

Table 3.1 shows a comparison between uncompressed SIP, CoSIP, and compressed SIP through the SigComp protocol.

Table 3.1: Comparison between SigComp and CoSIP compression rates.

SIP	SigComp (with Deflate algorithm)				CoSIP
	ND	SD	SD-DC	SD-SDC	
100%	61.63%	40.72%	17.46%	12.00%	55.10%

The signaling cost, in terms of overall compressed signaling data, is reported for a classical SIP session setup and tear-down (INVITE, 100 TRYING, 180 RINGING,

200 OK, ACK, BYE 200 OK), compared with non-compressed SIP. For SigComp, results have been based on the studies in [106, 107, 108], in which SigComp/SIP performance studies have been separately presented. As can be noted, the Deflate algorithm is the one that achieves best performance. In Table 3.1 different combinations of the Deflate algorithm with Static Dictionary (SD), Dynamic Compression (DC), and Shared Compression (SDC) have been considered. It is important to note that each of these variants, even if increasing the compression performance, may introduce more processing complexity and resource requirements (at least in terms of memory footprint).

The results show that CoSIP has similar performances as SigComp/SIP when No Dictionary (ND) or only SD is used. Instead, when SigComp/SIP is used with DC and with SDC, it achieves better performance. However, using CoSIP has still the advantage of having a smaller memory footprint (only one protocol must be implemented), processing load, and energy consumption (no compression/decompression functions must be executed) than SigComp/SIP, which are particularly important when dealing with constrained devices. Starting from the implementation of SIP [109] and CoAP [110], the implementation of the CoSIP protocol, as an open source library, has been released [111].

Using these available libraries, it has been decided to implement and evaluate the P2P overlay, introduced in [112], denoted as Distributed Location Service (DLS). The DLS is a DHT-based architecture which provides a name resolution service based on storage and retrieval of bindings between a URI, identifying some resources (e.g., a Web Service), and the information that indicates how such a resource can be accessed. Basically, the DLS implements a location service used to store data for discovering how to access resources. Together with each contact URI, some other information can be stored, like the expiration time, an access priority value, and, optionally, a displayable text (for example a description of the contact or a readable name). The service provided by DLS can be considered similar to that of the Domain Name System (DNS) systems, since it can be used to resolve a name to the information needed to access the content related to that name. However, the use of the DLS is preferable compared to the DNS for its robustness: if, for some reasons, a DNS server

is unreachable, the resolution cannot be performed. On the contrary, P2P overlays do not have single-point of failures or bottlenecks that might cause service disruption, resulting in a more robust, dynamic, and scalable solution. A DLS can provides two basic Remote Procedure Calls (RPCs) methods: (i) *put*(*key*, *value*) and (ii) *get*(*key*), where *key* is a resource URI (actually its hash) and *value* is a structured information that may include location information (e.g., a contact URI) together with a display name, expiration time, priority value, etc. The *get*(*key*) method returns the set of the corresponding values (actually the contact information) associated with the targeted resource. These methods allow to achieve support for (i) mobility: it is sufficient to replace an old resource with an updated one which considers the new position of the resource; and (ii) replication: it is sufficient to execute several *put* operations for the same resource in order to have copies diffused in the DHT.

Table 3.2 shows the available DHT messages, with fields characterization. Each RPC is mapped into a SIP REGISTER message, and each resource added to the DLS is defined by a JavaScript Object Notation Web Service Protocol (JSON-WSP) schema (a WS protocol that uses JSON for service description, requests, and responses, and that has been designed to cope with the lack of service description specification with documentation in JavaScript Object Notation Remote Procedure Call (JSON-RPC), a remote procedure call protocol in JavaScript Object Notation (JSON) format. The Table 3.2 also reports the cost in terms of bytes of each RPC messages both using SIP and CoSIP.

3.1.2 Experimental Analysis and Performance Evaluation

As previously enunciated, the aim of the presented experimental evaluation is to compare the influence on peers load, in term of exchanged bytes using SIP or CoSIP protocols. The experimentation has been conducted on an evaluation platform composed of four cluster nodes, in which each computing node is an 8-CPU Intel Xeon E5504 @ 2GHz, 16GB RAM, running Ubuntu 12.04 OS. The evaluation has monitored and measured the following metrics: (i) the total amount of received/sent Mbytes in the overlay as a function of the number of active nodes; (ii) amount of received Mbytes for a single node that is active in the overlay from the beginning of the experiment;

Table 3.2: Comparison between SIP and CoSIP DLS primitives in terms of the cost in bytes and field types required for their implementation.

DLS primitive		CoSIP (Bytes)	SIP (Bytes)	From	To	Contact
JOIN	Peer Registration Request	514	767	Peer URI	Peer URI	Peer URI
	Peer Registration Response	477	712	Peer URI	Peer URI	Peer URI
	Peer Query Request	435	649	Peer URI	Peer URI	—
	Peer Query Response	418	624	Peer URI	Peer URI	—
GET	Resource Query Request	411	614	Peer URI	Res. URI	—
	Resource Query Response	1559	2327	Peer URI	Res. URI	Contact Details
PUT	Resource Registration Request	1562	2332	Res. URI	Res. URI	Contact Details
	Resource Registration Response	1536	2292	Res. URI	Res. URI	Contact Details

and (iii) sent Mbytes for the same node active from the beginning to the end of the life of the entire overlay.

The evolution of experimental evaluation could be summarized in the following steps used to build the target overlay with 100 active nodes:

1. JOIN of new 10 peers, emulated using 10 independent Java processes (equally distributed on available cluster nodes);

2. overlay stabilization and execution of 10 PUT operations, executed by distinct random peers, storing different resource descriptors, encoded in JSON-WSP format;

3. execution of 10 GET operations, looking for existing resources already stored into the DLS overlay;

4. wait for overlay stabilization until the message propagation is done and each simulated peer is able to keep the resources of its competence;

5. ask each running peer to store data related to exchanged bytes and general execution logs;

6. run a collector process in order to retrieve stored logs of each peer for subsequent analysis;

7. return to phase 1 and start a new iteration to increase the number of active peers in the overlay.

Through the defined execution steps an overlay of different sizes (ranging from 10 to 100 nodes) has been created, in order to collect required data at each step and to understand the benefits of using a constrained protocol such as CoSIP instead of SIP for IoT P2P overlays. Presented values have been obtained through multiple executions, in order to have a statistical confidence interval of 95%. Collected and analyzed data show how, using CoSIP, it is possible to obtain an average gain of approximately 33% of exchanged Mbytes in the overall overlay compared with the use of traditional SIP protocol. Graphs in Fig. 3.1 illustrate the total amount of received Mbytes by active peers in the DLS overlay during the entire overlay lifetime, in both SIP and CoSIP configurations as a function of the overlay size.

With the aim to analyze the behavior and the impact of a constrained protocol on a single node, it has been evaluated the logs of a peer active in the overlay from the beginning to the end of the overlay lifetime. Fig. 3.2 reports the total amount of received Mbytes during peer's lifetime both using SIP and CoSIP protocol.

The trend confirms, as expected, how the use of a constrained protocol allows to reduce the overall traffic and in this specific case to process a lower amount of incoming bytes to the node. The same trend is displayed and confirmed in Fig. 3.3, in which transmitted Mbytes of a single peer is reported as a function of the experimentation lifetime.

Starting from the collected data and with the aim to evaluate the load in terms of exchanged packets, the amount of exchanged packets associated to SIP and CoSIP

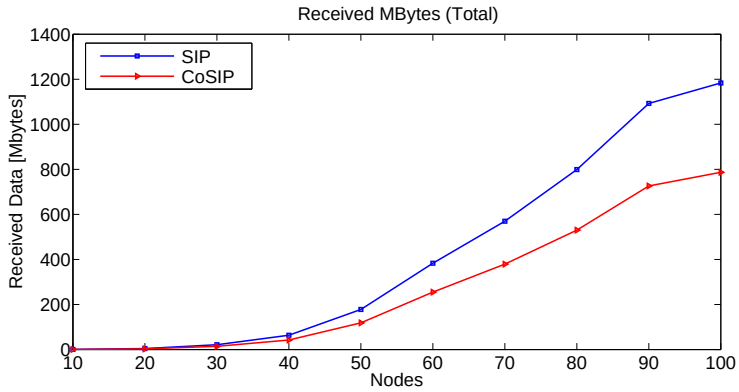


Figure 3.1: Total amount of received Mbytes by peers active in the the DLS as a function of the overlay size.

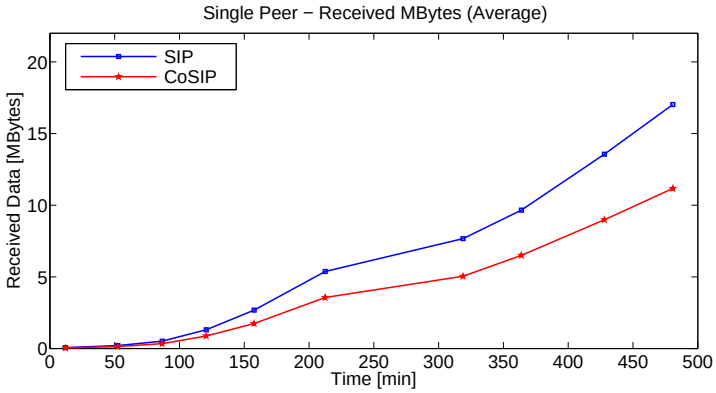


Figure 3.2: Received Mbytes by a single node active in the overlay during its life-cycle.

signaling at the application layer and the consequent number of packets after the typical fragmentation occurring when considering IEEE 802.15.4 at data link layer have been estimated.

Results in Table 3.3 show how the use of CoSIP allows not only to reduce the generated traffic in terms of Mbytes, but also to decrease the average number of received packets at lower layers, in particular for constrained networks in which the

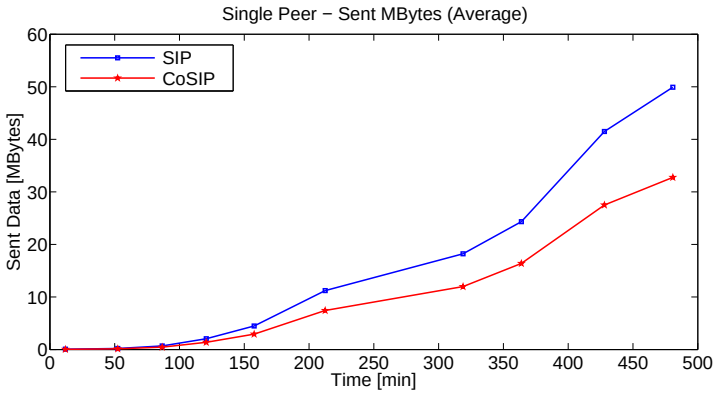


Figure 3.3: Transmitted Mbytes by a single node active in the overlay during its life-cycle.

Table 3.3: Average number of received application-layer messages and data link-layer frames (IEEE 802.15.4) for a single peer in an overlay of 100 nodes.

	CoSIP Signaling	SIP Signaling
Application-layer Messages	6561	6561
IEEE 802.15.4 Frames	70536	101702
CoSIP/SIP Ratio	0.69355	

effect of fragmentation is massive (e.g., by using Ethernet, the advantage in terms of number of packets is minimized since the impact of fragmentation is lower for the considered RPC messages and their sizes).

In the end, this significant gain, in the context of P2P overlays for IoT application scenarios, is important not only to reduce bandwidth usage and amount of transmitted data, but also in terms of energy consumption and processing capabilities, especially important for constrained devices. In particular, constrained nodes can save CPU cycles, memory, and transmission periods during their lifetime.

3.2 The Web of Things Testbed

Due to the already highlighted heterogeneity of IoT-oriented SOs, in recent years the global reach of the IoT presents major application development challenges. In order to foster IoT development and diffusion, applications are increasingly built around the well-known Web model, bringing about the so-called Web of Things (WoT). Whereas simulation tools typically focus on evaluating lower-layer communication protocols, in recent years several IoT testbeds have been deployed to evaluate IoT solutions in realistic smart environments under real-world conditions. Starting from these needs, a real IoT/WoT-oriented testbed has been deployed, denoted as Web of Things Testbed (WoTT), composed by heterogeneous and self-configurable SOs (thanks to service discovery mechanisms previously presented).

The WoTT is a heterogeneous and innovative WoT-based testbed that enables developers to easily design and evaluate new services and applications in a real IoT environment and to effectively test human-object interaction mechanisms [76, 113], which will play a fundamental role in broadening IoT use. WoTT is particularly suited for this purpose because its architecture is completely based on standard protocols and network interfaces, without custom or proprietary solutions that would jeopardize interoperability among nodes. WoTT's main goals are to:

- hide low-level implementation details;
- enhance network self-configuration by minimizing human intervention;
- transparently and simultaneously manage multiple protocols and platforms;
- provide a platform for the design and testing of human-object interaction patterns.

To effectively test new WoT-related applications, WoTT consists of several types of nodes that differ in terms of both computational capabilities and radio-access interfaces. Nonetheless, these nodes can be grouped into two main classes: (i) Constrained IoT (CIoT) nodes and (ii) Single-Board Computer (SBC) nodes. CIoT nodes are mainly based on the open source Contiki OS and correspond to Class 1 devices

— those that cannot easily talk to other Internet nodes that have a full protocol stack, but that can use a protocol stack designed for constrained nodes — as defined in the IETF memo "Terminology for Constrained Node Networks" [114]. SBC nodes are more powerful, typically running a Linux OS and having multiple network interfaces; these correspond to Class 2 devices — those that use the same protocols as notebooks or servers [114]. Regardless of what the actual nodes are, the standard communication protocols and mechanisms used in WoTT enable the testbed to manage node diversity seamlessly, making it possible to treat each node simply as an IP-addressable host.

3.2.1 An IP-based Infrastructure for Smart Objects

CIoT nodes are connected at the physical layer by IEEE 802.15.4 wireless links, whereas IPv6 is used at the network layer in combination with 6LoWPAN [115] and IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) (the routing protocol for low-power and lossy networks, or LLNs). Sensor-equipped CIoT nodes can act as CoAP servers or clients running Erbium, a lightweight CoAP implementation. In Table 3.4 are detailed the CIoT nodes involved in the deployment of the WoTT.

As with CIoT nodes, SBC nodes can act as CoAP clients or servers, with fewer implementation constraints. For example, on Arduino Yun nodes, a JavaScript application initializes a CoAP server through an instance of Node.js [116]. Other SBC nodes, such as the Intel Galileo boards and Raspberry Pi, can support different types of languages ranging from Python to Java. In Table 3.5 are detailed the SBC nodes involved in the deployment of the WoTT.

As shown in Fig. 3.4, the WoTT is heterogeneous by design, to enable seamless communication among SOs and between the WoTT and external Internet elements, such as the Cloud, ISPs, and consumers. As previously highlighted, IP protocol adoption — in particular, IPv6 or IPv6+6LoWPAN — is universally considered a key communication enabler for the Future IoT. Thus, WoTT adopts IP as a common network substrate, thereby allowing simple integration into the existing Internet. Note that all WoTT nodes use standard protocols at all layers of the protocol stack; this includes several PHY- and MAC-layer standards — e.g., IEEE 802.11, IEEE 802.15.4, and IEEE 802.3 — as well as application-layer protocols. The architecture also con-

Table 3.4: Constrained Internet of Things (CIoT) nodes in the WoTT.

Constrained IoT nodes				
No.	**Node**	**Hardware**	**OS**	**Network interface(s)**
6	TelosB	MCU: TI MSP430F1611	Contiki	IEEE 802.15.4
		RAM: 10 KB		
		ROM: 48 KB		
20	Zolertia Z1	MCU: TI MSP430F2617	Contiki	IEEE 802.15.4
		RAM: 8 KB		
		ROM: 92 KB		
10	OpenMote	MCU: ARM Cortex-M3	Contiki	IEEE 802.15.4
		RAM: 32 KB		
		ROM: 512 KB		
35	SimpleLink SensorTag	MCU: ARM Cortex-M3	TI-RTOS	IEEE 802.15.4 / BLE
		RAM: 20 KB		
		ROM: 128 KB		

MCU: memory control unit

tains an innovative network element, the IoT Hub [117], which operates at different layers of the protocol stack to further enhance interoperability among communicating devices by integrating several networks into a single IP-based substrate and implementing important functions at the application layer. Due to greater capabilities in computational power and networking, SBC nodes can effectively implement all IoT Hub functions.

WoTT's Wi-Fi networking infrastructure is based on Cisco Connected Mobile Experiences (CMX) [118] Access Points (APs) and can be used to track devices for indoor localization purposes. In particular, the CMX platform provides Mobility Services Engine RESTful APIs, which allow developers to integrate service customization with location information into mobile applications, such as location-aware equipment tracking, guest access, and device-based services. This feature is currently used to build user location–aware IoT applications that can continuously monitor users, enabling specific and augmented interaction with the surrounding environment.

Focusing on the application layer, WoTT currently supports: (i) CoAP, (ii) MQTT,

Table 3.5: Single-Board Computer (SBC) nodes in the WoTT.

SBC nodes				
No.	**Node**	**Hardware**	**OS**	**Network interface(s)**
20	Intel Galileo	CPU: SoC X Intel Quark X1000	[Linux] Debian	IEEE 802.3
		RAM: 256 MB		
		Memory (SD): 8 GB		
5	Raspberry Pi B	CPU: Broadcom BCM2835 ARM11	[Linux] Raspbian	IEEE 802.3/802.11
		RAM: 512 MB		
		Memory (SD): 8 GB		
5	Arduino Yún	**Linux environment**	[Linux] OpenWRT	IEEE 802.3/802.11
		CPU: Atheros AR9331		
		RAM: 64 MB		
		ROM: 16 MB		
		Arduino environment	Arduino	
		MCU: ATmega32u4		
		RAM: 2.5 KB		
		ROM: 32 KB		
4	UDOO	**Linux environment**	[Linux] UDOObuntu	IEEE 802.3/802.11
		CPU: Freescale i.MX 6 ARM Cortex-A9		
		RAM: 1 GB		
		Memory (SD): 8 GB		
		Arduino environment	Arduino	
		MCU: Atmel SAM3X8E ARM Cortex-M3		
		RAM: 100 KB		
		ROM: 512 KB		

MCU: memory control unit

and (iii) HTTP, which is mainly used for communication between WoTT and external Internet actors or consumers, such as Cloud storage services or IoT-unaware clients. Among WoTT components, the IoT Hub is the key IoT enabler because it manages

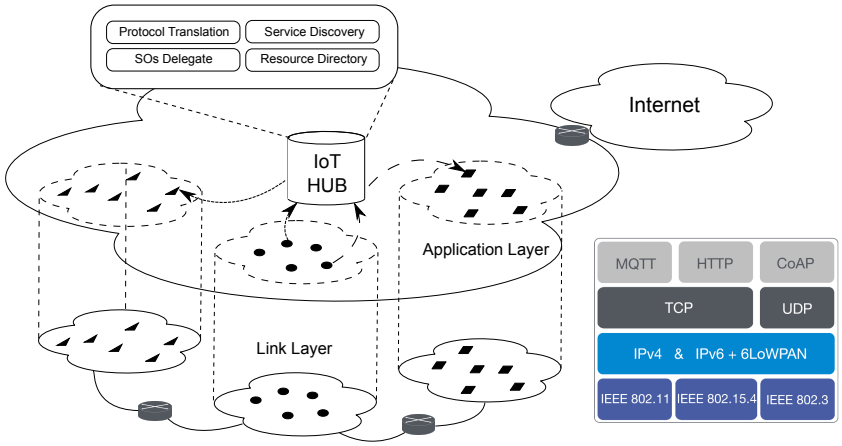


Figure 3.4: WoTT architecture and protocol stack. The central component is the IoT Hub, which interacts with the various layers and manages the testbed's heterogeneous network.

the different access technologies and supports full IP connectivity among all objects. The implementation of several functionalities at the application layer enables all the protocols listed in Fig. 3.4 to coexist in the same environment.

The software running on different SOs has been developed with different programming languages, reinforcing the idea that — because of various features provided at the application layer, together with strict compliance to IoT standards — developers can create new IoT applications easily and without additional constraints.

3.2.2 IoT Hub-enabled Smart Object Interactions

WoTT does not simply enable communications between IoT actors; it constitutes a "uniform" super-entity able to provide enhanced functionalities that go beyond the mere union of its components' features.

To achieve this super-entity status, WoTT uses the various communication technologies in the IoT Hub to bridge and merge together several networks into a single IP network. The IoT Hub also implements several functions at the application layer: it manages the services and resources available in the overall infrastructure, thereby

playing a key role at the application layer.

IoT Hub use is expedient for several reasons. The extreme heterogeneity of IoT devices requires mechanisms to support the management of and seamless interactions among SOs as well as humans. Moreover, because SOs' limited data-collection capabilities could preclude them from handling large numbers of concurrent requests, limiting direct access to SOs is preferable. In other cases, extremely limited devices could act as clients to implement data-collection behavior.

From a networking standpoint, the IoT Hub is a Fog node placed at the edge of multiple physical networks with the goal of creating an IP-based IoT network. The IoT Hub plays a fundamental role by implementing the following functions at the link and application layers of the protocol stack.

- *Border Router (BR)*: the IoT Hub bridges one or more networks (such as several IEEE 802.15.4 networks).
- *Service and Resource Discovery*: the IoT Hub is able to discover which SOs are available in the network and their hosted resources.
- *Resource directory*: the IoT Hub complies with the specifications provided in the IETF technical report "CoRE Resource Directory" [101] and maintains a list of all resources available in the bridged networks, thereby creating a centralized entry point for applications that need to perform resource lookup.
- *Origin-Server*: the IoT Hub provides a CoAP server that hosts the resources of different SOs.
- *CoAP-to-CoAP (C2C) proxy*: the IoT Hub provides proxying capabilities for CoAP requests coming from external clients targeting constrained nodes.
- *HTTP-to-CoAP (H2C) proxy*: in order to allow HTTP clients to access CoAP resources, the IoT Hub provides HTTP-to-CoAP cross-proxying (protocol translation).
- *Cache*: the IoT Hub keeps a cache with the representation of most recently

accessed resources to act as a SO delegate, thus minimizing latencies and unloading constrained devices.

Because the IoT Hub relies on standard interaction mechanisms and communication protocols, SOs do not depend on it for their operation. Instead, the IoT Hub mitigates the presence of nonstandard components that are interoperable with standard-compliant devices. The IoT Hub is not required for interaction and interoperability, but its presence extends the IoT network and increases its capabilities by simplifying and hiding complex and important tasks such as service discovery and routing.

3.2.3 Integration Challenges

The main challenges encountered in WoTT deployment have been design related: how to define different elements and their functionalities, represent different resources and their relationships through suitable hypermedia, and maintain compatibility with standards.

The efforts devoted to WoTT design, the IoT Hub, and the use of standards have simplified the deployment process, making the integration of all different elements straightforward, notwithstanding their heterogeneity. In particular, WoTT hides critical implementation issues encountered at lower layers. For example, in a constrained network, such as IEEE 802.15.4, the resource advertisement feature is a critical point, requiring strict assumptions about energy and memory consumption on each constrained device. It is possible to tackle this issue by adopting an efficient multicast-based solution that can reduce the energy consumption. WoTT's application-oriented end users are not concerned with these implementation details.

Another implementation challenge hidden from end users is configuring network elements such as routers and APs to create a unique IP-addressable network. Original firmware provided with common network elements typically does not allow such unusual configurations. To overcome this limitation, WoTT network components have been "hacked" with other more customizable and advanced firmwares.

3.2.4 Security, Authentication and Authorization

With the goal of having the WoTT eventually become a publicly available platform, it has been designed to comply with security policies and mechanisms and to adopt strong defensive measures to counteract security threats coming from external environments as well as internal malicious IoT entities. Web-derived security technology and mechanisms are useful — they can be taken as a reference and applied to enhance the protection and reliability of IoT environments.

The WoTT can manage and authenticate external request issuers, while simultaneously defining processing policies and rules. To comply with these specifications, the testbed allows external connections through VPN and Secure Shell (SSH) tunnels using credentials issued by WoTT administrators.

Several security mechanisms for the interaction among SOs are implemented within the testbed. Transport Layer Security (TLS) (in conjunction with HTTP) and Datagram Transport Layer Security (DTLS) (in conjunction with CoAP) are implemented on a set of devices hosting resources that need to be accessed through secured application protocols. The IoT Hub implements both protocols to provide secure resource access through proxying. Of course, end-to-end security through proxying cannot be ensured as no TLS-to-DTLS mapping is defined.

Although DTLS and TLS are implemented to enforce confidentiality and authenticity of communication between endpoints, real-world IoT systems must be able to manage several users accessing resources deployed in a smart environment. This is when authorization comes into play: mechanisms must be defined to ensure that only authorized entities can interact with objects. For example, building offices in an IoT environment should be secure and only able to be unlocked by individuals who have been granted access. Although authorization is obviously a critical issue, research has yet to focus on defining mechanisms to manage IoT access policies and grant authorizations.

3.2.5 Buiding Web of Things-oriented Applications

To validate WoTT's benefits and demonstrate the ease of integrating a newly deployed application within the testbed, an application for wearable and mobile-oriented applications has been developed.

Thanks to their portability, wearable and mobile devices are obvious solutions for tracking people's activities. To accomplish this, it has been implemented indoor localization features into WoTT using CMX APs to triangulate the locations of people and objects. The availability of localization APIs in the CMX system enables to create applications that can follow users throughout an IoT environment and possibly even anticipate their movements. Together with access-point-based localization, the use of on-board inertial measurement units can further improve the tracking experience.

Mobile devices play an important role in WoTT's architecture. Aside from interacting with SOs, they can also act as SOs, providing data generated by their on-board sensors. Mobile devices can thus be considered as WoTT nodes, making WoTT highly dynamic.

As a uniform, application-oriented platform, WoTT can be used by developers to easily create and test real-world IoT applications in a short period of time, thus making it more attractive than other currently available platforms. This is due to ready-to-use capabilities and the direct/active interactions that a deployed application can have with the resources available in WoTT. From an operational point of view, developers simply run their applications on testbed resources without needing to add virtualized environments or services; thus the application becomes part of a WoT scenario in which consumers are not only "readers" but active participants.

Based on the WoTT's capabilities, the application shown in Fig. 3.5 has been developed and tested on the Android Wear platform using LG G Watches and Android 5.0.1 smartphones [119]. In the near future, as more SOs are deployed, vendor-provided apps are less likely to be the usual means by which humans interact with things, and a more standard approach will be required to do so effectively.

The application performs the following steps. First, the mobile device discovers nearby SOs proactively and reactively, by means of standard service and resource discovery mechanisms. Then, it forwards the collected information to its connected

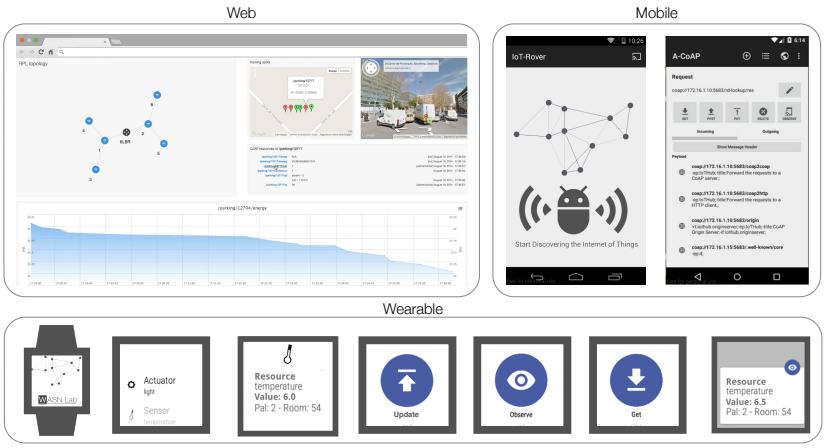


Figure 3.5: A WoTT-based application for mobile and wearable devices. Resources and interactions are revealed gradually, according to the REST paradigm, so that the application can adapt itself dynamically.

wearable device interface. Through wearable interfaces, a user can see and browse a list of all the resources that have been discovered and select one to interact with. Interactions are thereby WoTT resources can be deployed on different platforms, such as different SOs, and by using heterogeneous protocols. However, this is completely transparent to a developer who is able to access all these resources; the only constraint is to use standard protocols. This is possible through the IoT Hub's abstraction ability — it is not provided by other existing testbeds, such as SmartSantander.

The SOs' own sensors and actuators, users' devices, and other SOs can each act as clients and interact with one another through the following approaches.

- *Polling* allows clients to retrieve the value associated with the queried resource by performing a CoAP GET request.

- *Observing* can be used by clients to receive asynchronous updates when the value of the specified resource changes, which is a more efficient mechanism because it avoids periodic data polling.

- *Acting* is used by clients to set up the value of a specified resource, such as activating an actuator, depending on the function set provided by the resource.

The observing approach, which has not been defined in HTTP, is a lightweight CoAP-oriented interaction mechanism [100]. SO-related resource observation can be achieved by performing a particular CoAP GET request, which contains an *Observe* option. This option instructs the target SO to add a new subscriber that will receive subsequent resource updates in push mode. Subscribers can also stop observing a resource at any time and unsubscribe from updates.

The use of standard communication protocols and network interfaces, well-known Web-based design approaches, and widely varied hardware platforms are changing the IoT and presenting new opportunities to developers, businesses, and users. In this dynamic and evolving scenario, the availability of real and accessible resource-oriented testbeds that allow active and direct interaction among users and devices with low-level nodes and services are a key enabler for widespread future IoT adoption — and, indeed, are driving the transition from the IoT to the WoT.

WoTT exemplifies a novel architectural and networking approach to important IoT challenges. Designing and implementing the testbed has confirmed the need for open evaluation platforms to explore and integrate innovative IoT applications and to bridge the gap between users and things. WoTT's hardware/software heterogeneity confirms that proper use of standard protocols such as HTTP, CoAP, and MQTT and of interaction paradigms such as REST and service/resource discovery are fundamental to enabling transparent and dynamic interactions among multiple SOs and personal mobile and wearable devices. Moreover, this heterogeneity can be extended by improving the IoT Hub features (e.g., by adding support to Bluetooth devices). This would open WoTT to an emerging category of IoT-enabled devices, namely those using Bluetooth Low Energy (BLE) [120], and BLE service discovery mechanisms such as Eddystone [121].

Nevertheless, important issues must be addressed to make the IoT part of daily life. Testbeds like WoTT are the perfect experimental playgrounds to boost and support IoT application development by creating a common space that designers, hardware manufacturers, and companies can exploit to make the IoT accessible and easy

to use for everyone, just as the Internet is right now.

3.3 Integration of Wi-Fi Mobile Nodes in the WoTT

Current trends show that the main goal of recently deployed IoT-oriented testbeds (e.g., the WoTT) is to foster a rapid deployment of Web-enabled everyday objects, allowing end users to manage and control SOs in a simple way by using Web browsers. In this way, the ubiquity of Wi-Fi technology in modern scenarios and its interoperability with widely used application protocols (e.g., HTTP) makes it a promising technology for the future, since Wi-Fi guarantees large-area coverage, high bandwidth, robustness, and cost-effectiveness. Whereas Wi-Fi technology adoption seems to be a good solution in some IoT scenarios (e.g., video monitoring), it can be interesting to analyze and describe an experimental integration of Wi-Fi nodes [122] in the WoTT testbed, composed of heterogeneous nodes hosting resources that can be accessed through the CoAP protocol. The integration of new Wi-Fi nodes hosting HTTP resources, using only standard mechanisms, dynamic nodes and resource discovery paradigms, allows end-users to easily interact with all SOs, without worrying about specific protocol translation needs.

3.3.1 Architecture

The goal of the proposed integration, whose main component modules are shown in Fig. 3.6, is to exploit the basic concept of the IoT, integrating Wi-Fi-based devices in the WoTT testbed, mainly composed of CoAP-enabled IEEE 802.15.4 nodes.

Resource Directory Module

The resource directory module [123] acts as an information repository for resources hosted by the endpoints in an IoT network. An endpoint is an IP-enabled entity associated with an address and a port: therefore, a physical node can host one or more endpoints, each one owning one or more resources. The resource directory is also able, in conjunction with a proxy module, to handle requests through different appli-

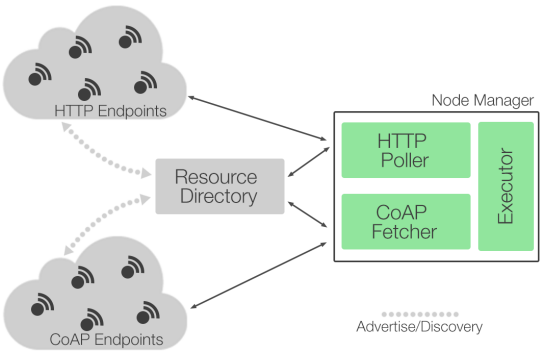


Figure 3.6: Overview of the proposed architecture.

cation protocols. In the WoTT testbed, all endpoints advertise themselves through the JmDNS library [124], a Java-based implementation of the Multicast DNS (mDNS) discovery protocol, regardless of the supported application protocol, so that resources hosted by endpoints are automatically added to the resource directory.

Node Manager Module

The Node Manager module is a Java-based entity managing, in absence of the IoT Hub module, the communication with all nodes integrated in the WoTT. It is composed of several sub-modules.

- The **HTTP Poller**, implemented using the Jetty library [125], retrieves the list of available HTTP resources from the resource directory, looking for the desired ones accordingly with CoRE interface (*if*) and resource type (*rt*) attributes. Once the resource address is retrieved, the HTTP Poller starts the interaction with them. Because the CoAP-defined *Observe* option is not natively supported by HTTP, the module must send periodic HTTP GET requests (*polling*) to the available HTTP resources.

- The **CoAP Fetcher**, implemented with the Californium (*Cf*) library, is responsible for interacting with CoAP-based endpoints.

- The **Executor** module processes the responses received by other modules (encoded in JSON format to allow a light data exchange) and decides the actions to be performed, if the responses satisfy some criteria (e.g., forwarding the obtained data to the Cloud [126] or to remote processing infrastructures [127, 128], or making new requests to an actuator in the WoTT).

Wi-Fi Nodes

Among many options, the Wi-Fi node integrated in the WoTT is the TI SimpleLink Wi-Fi CC3200 LaunchPad [129]. This board is equipped with two processors, which control the integrated Web server and the TCP/IP stack for networking operations, and a radio interface through the IEEE 802.11b/g/n protocol. It is also equipped with temperature and three-axis accelerometer sensors: sensed values can be extracted by means of internal prioritized tasks. The LaunchPad board has been chosen because it is one of the most complete devices available for IoT application development and can be battery powered. These features make it mobile, thus increasing the heterogeneity of the WoTT.

To smartly manage the new Wi-Fi node, the proposed architecture abides by the following principles: the internal representation of the resource list and the representation of the values of each sensor must be maintained on the LaunchPad board. The former aspect has been solved by "re-engineering" the board and enabling to create, at start-up time, a new HTTP resource reachable at */.well-known/core* URI: this represents the entry-point used for resource discovery and returns the list of all available HTTP resources in CoRE Link-Format fashion [130], in which each resource is described by its URI, *if* and *rt* CoRE attributes. Furthermore, the criticality related to the representation of the resource values is because, by default, the LaunchPad node responds to an HTTP GET request encapsulating the last retrieved value into an HTML page, which is difficult to read and parse. Considering this observation, in the proposed approach the measured value is written in a text file by using the JSON format. This solution is more readable, lightweight, and easily interpreted than that provided by TI.

3.3.2 Use Case

The proposed architecture has been tested by implementing the Wi-Fi node-based IoT-based surveillance system shown in Fig. 3.7: a security camera takes a snapshot of a monitored environment if a sensor detects an unexpected presence.

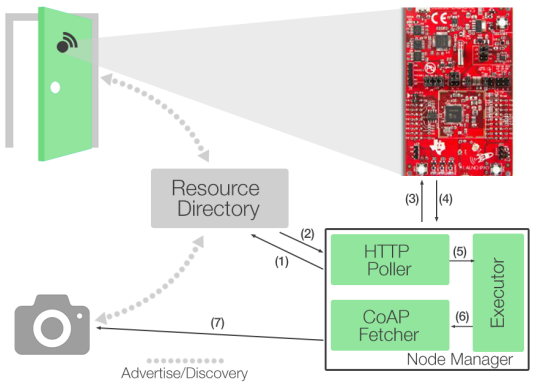


Figure 3.7: IoT-based surveillance infrastructure.

The system is managed by the Node Manager module running on a Raspberry Pi 1 Model B, which starts querying the resource directory to discover the movement sensor and the surveillance camera. The former is handled by the accelerometer sensor on the TI LaunchPad Wi-Fi board, which is installed on the door in the supervised environment. The board starts advertising itself, to be discovered by the resource directory module. The camera entity is represented by another Raspberry Pi 1 Model B, already integrated in the WoTT testbed and equipped with a PiNoIR camera module, running a built-in HTTP server (written in Python language) and a CoAP server (written with *Cf* library). Both of these servers allow the capture of pictures upon a request to specific URIs. First, the Node Manager discovers the movement resource on the Wi-Fi node, identified by the CoRE attributes *if=core.s* and *rt=accl*, and the surveillance camera, with *if=core.s* and *rt=camera*. The HTTP Poller then starts to periodically poll the LaunchPad board with HTTP GET requests, whose responses are finally handled by the Executor module. The current value received from the Wi-Fi node is then compared with two thresholds and, if a movement is detected (namely,

the acceleration intensity is above a pre-set threshold), the Executor asks the CoAP Fetcher module to send a synchronous CoAP GET request to the Raspberry Pi board handling the surveillance camera, which captures a new snapshot (at a resolution of 320×240 px) and forwards it back to the Executor module.

In Table 3.6, the experimentally measured times (in terms of average values and standard deviations), with reference to the different steps highlighted in Fig. 3.7, are shown.

Table 3.6: Measured times.

Operation	Average Duration [s]	Standard Deviation [s]
Intrusion sensor polling (steps 3–5)	0.91	0.24
Snapshot request (steps 6–7)	1.7	0.47

The choice to send a CoAP request, instead of an HTTP one, is to highlight the heterogeneity of the analyzed use case and of the proposed architecture, in which there is no physical connection between the Wi-Fi board and the camera, because they interact only owing to the presence of the resource directory.

Thanks to the adoption of only standard solutions, the proposed integration makes the WoTT architecture scalable and lightweight, because HTTP and CoAP resources on heterogeneous devices are managed in a similar way.

Chapter 4

Services for IoT

One of the distinctive features of IoT systems is the deployment of a huge amount of heterogeneous data sources collecting data from the environment and sending information through the Internet to collectors. The work of all data sources generate, as a whole, streams with a very high frequency. Moreover, several relevant IoT scenarios (such as industrial automation, transportation, networks of sensors and actuators) require real-time performance or, at least, a predictable (and consistent) latency. The presence of a large number of IoT data sources, on one hand, that globally generate data at a high rate (even though a single IoT service generates a limited amount of data), and low-latency constraints, on the other hand, call for innovative Cloud architectures, able to efficiently handle such massive amount of information. As already highlighted, Big Data approaches typically have an intrinsic inertia because they are based on batch processing. By this, they are not suitable to the dynamicity of IoT scenarios with real-time requirements. To better fit these requirements, the Big Data paradigm is shifted to a new paradigm, which is denoted as "Big Stream". Big Stream-oriented systems should react effectively to changes and provide smart behavior for allocating resources, thus implementing scalable and cost-effective Cloud services. The Big Stream paradigm is specifically designed to perform real-time and ad-hoc processing in order to link incoming streams of data to consumers. This new paradigm should have a high degree of scalability and fine-grained/dynamic config-

urability, and efficiently manage heterogeneous data formats which are not a-priori known.

4.1 The Big Stream Paradigm

The Big Stream paradigm allows to perform real-time and ad-hoc processing in order to link incoming streams of data to consumers, with a high degree of scalability, fine-grained and dynamic configuration, and management of heterogeneous data formats. In brief, while both Big Data and Big Stream deal with massive amounts of data, the former focuses on the analysis of data, as shown in Fig. 4.1, while the latter focuses on the management of flows of data, as shown in Fig. 4.2.

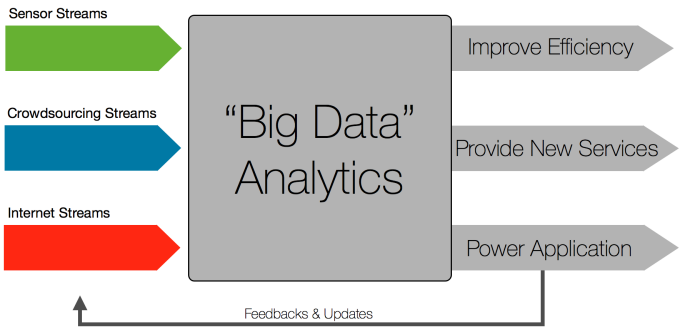


Figure 4.1: The volume of data analysis in Big Data systems.

The main differences between Big Data and Big Stream paradigms can be summarized as follows.

- **The meaning of the term "Big"**: in Big Data it refers to "volume of data", while in Big Stream it refers to "data generation rate".
- **Real-time or low-latency requirements of different consumers**: they are typically not taken into account by Big Data.
- **Nature of data sources**: Big Data deals with heterogeneous data sources in a wide range of different areas (e.g., health, economy, natural phenomena),

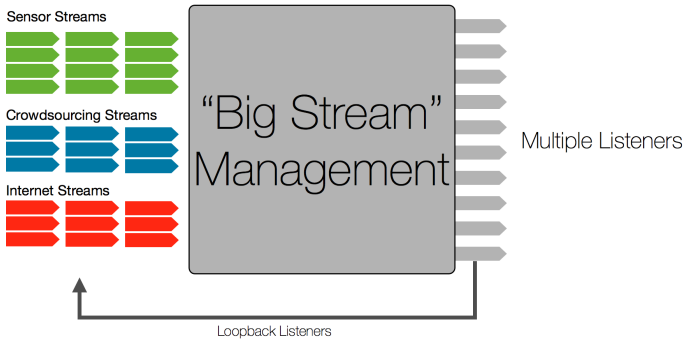


Figure 4.2: The multiple data sources and listeners management in Big Stream system.

not necessarily related to IoT. Instead, Big Stream data sources are strictly related to the IoT, where heterogeneous devices send small amounts of data generating, as a whole, a continuous and massive information stream.

- **Objective**: Big Data focuses on information management, storage and analysis, following the Data-Information-Knowledge (DIK) model [131]; instead, Big Stream focuses on the management of data flows, being specifically designed to perform real-time and ad-hoc processing, in order to forward incoming data streams to consumers.

Big Stream-oriented systems should react efficiently to changes, providing smart resource allocation and, thus, implementing scalable and cost-effective Cloud services. This also affects the data selected as relevant, in different processing steps, for the final consumer. For instance, while for Big Data applications it is important to store all data in order to be able to perform any successive required computation, Big Stream applications might decide to perform data aggregation, filtering or pruning, in order to minimize the latency in conveying the final computation output to consumers, with no need for persistence. Eventually, as a generalization, a Big Data application might be a consumer of Big Stream data flows and perform storage operations.

Based on these observations, it is interesting to propose and detail an architecture

targeting Cloud-based applications with real-time constraints, i.e., Big Stream applications, for IoT scenarios. The proposed architecture relies on the concepts of "data listener" and "data-oriented processing graph", in order to implement a scalable, highly configurable, and dynamic chain of computations on incoming Big Streams, and to dispatch data with a push-based approach, thus providing the lowest delay between the generation of information and its consumption.

4.2 The Big Stream-oriented Architecture

As previously stated, a major difference between Big Data and Big Stream resides in the real-time and low-latency requirements of consumers. The gigantic amount of data sources in IoT applications has mistakenly made Cloud services implementers believe that re-using Big Data-driven architectures would be the right solution for all applications, rather than designing specific paradigms for those scenarios. IoT application scenarios are characterized by a huge number of data sources, sending small amounts of information to a collector service, typically at a limited data rate. Many services can be built on top of these data, such as environmental monitoring, building automation, and SCs applications. These applications are typically characterized by low-latency or real-time requirements, in order to provide efficient reactive/proactive behaviors.

Applying a traditional Big Data approach for IoT application scenarios might lead to high — even unpredictable — latencies between data generation and its availability to a consumer, since this was not among the main objectives behind the design of Big Data systems. The main delay contributions introduced when data, generated by SOs in IoT networks, need to be processed, stored, and then polled by consumers, are illustrated in Fig. 4.3. Clients interested in processed data are extremely heterogeneous, spanning from mobile or desktop applications to Data Warehouse (DW) applications and till other IoT networks composed by different SOs.

The total delay required by any data to be delivered to a consumer can be expressed as:

$$t_{tot} = t_0 + t_1 + t_2$$

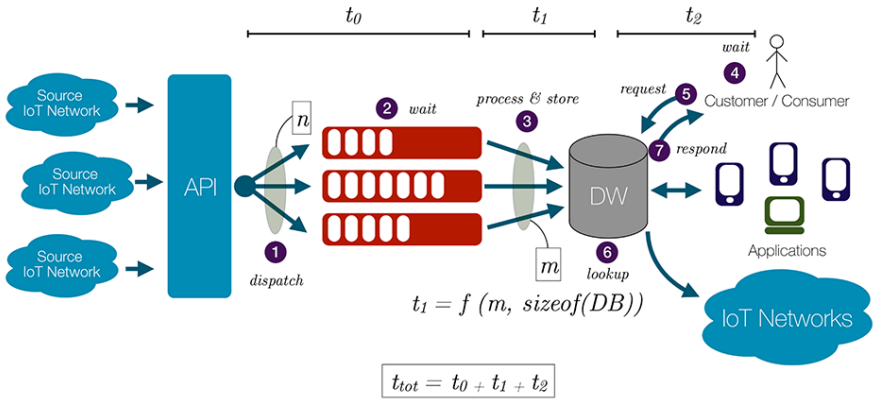


Figure 4.3: Delay contributions in a traditional Big Data architecture for IoT, from data generation to applications information delivery.

where:

- t_0 is the time elapsed from the moment a data source sends information, through an available API, to the Cloud service (1), which dispatches the data to an appropriate queue, where it can wait for an unpredictable time (2), in order to decouple data acquisition from processing;
- t_1 is the time needed for data, extracted by the queue, to be pre-processed and stored into a DW (3): this time contribution depends on the number of concurrent processes that need to be executed and get access the common DW and the current size of the DW;
- t_2 is the data consumption time, which depends on: (i) the remaining time that a polling consumer needs to wait before performing the next fetch (4); (ii) the time for a request to be sent to the Cloud service (5); (iii) the time required for lookup in the DW and post-process the fetched data (6); and (iv) the time for the response to be delivered back to the consumer (7).

It can be observed that the described architecture is not optimized to minimize the latency and, therefore, to feed (possibly a large number of) real-time applications

but, rather, to perform data collection and batch processing. Moreover, it is important to underline and understand that significant data for Big Stream applications might be short-lived, since they are to be consumed immediately, while Big Data applications tend to collect and store massive amounts of data for an unpredictable time.

The main design criteria of the Big Stream architecture proposed in [126] are: (i) the minimization of the latency in data dispatching to consumers, and (ii) the optimization of resource allocation. The main novelty lies in the concepts of "consumer-oriented" data flows and "listeners". The former denotes a different approach in retrieving incoming data, rather than being based on the knowledge of collection points (repositories) to which request data. The latter relies on final consumers: data generated by a deployed SOs, might be of interest for some consumer application, denoted as listener, which can register itself in order to receive updates (either in the form of raw or processed data) coming from a particular streaming endpoint (i.e., Cloud service). On the basis of application-specific needs, each listener defines a set of rules, which specify what type of data should be selected and the associated filtering operations. For instance, referring to a smart parking scenario, a mobile application might be interested in receiving contents related only to specific events that occur within a given geographical area, in order to accomplish relevant tasks. Specifically, the application can listen for parking sensor status updates, the positions of other cars, or weather conditions, in order to find available parking spots.

The proposed Big Stream architecture guarantees that, as soon as they are available, data will be dispatched to the listener, which is thus no longer responsible to poll data, thus minimizing latencies and possibly avoiding network traffic. The information flow in a listener-based Cloud architecture is shown in Fig. 4.4. With the Big Stream paradigm, the total time required by any data to be delivered to a consumer can be expressed as:

$$t_{tot} = t_0 + t_1$$

where:

- t_0 is the time elapsed from the moment a data source sends information, through an available API, to the Cloud service (1), which dispatches the data to an

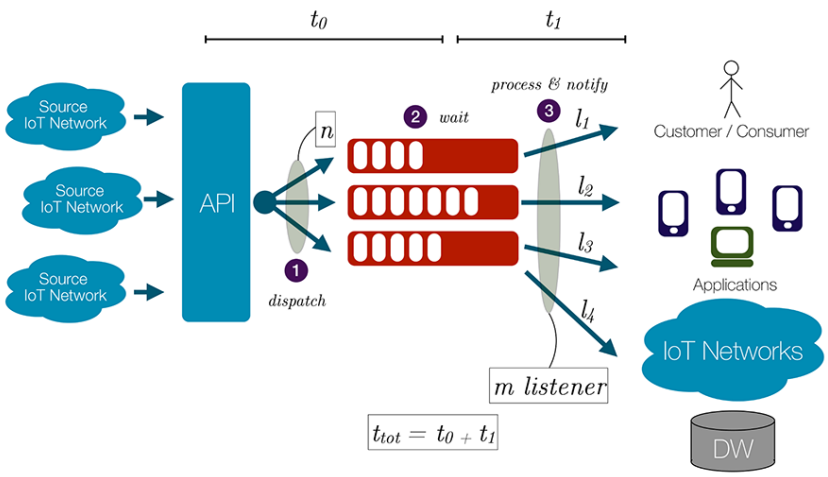


Figure 4.4: The delay contributions from data generation to consumers information delivery following the listener-based Big Stream approach.

appropriate queue, where it can wait for an unpredictable time (2), in order to decouple data acquisition from processing;

- t_1 is the time needed to process data extracted from the queue and be processed (according to the needs of the listener, e.g., to perform format translation) and then deliver it to registered listeners.

It is clear that the perspective inversion introduced by a listener-oriented communication is optimal in terms of minimization of the time that a listener must wait before it receives data of interest. In order to highlight the benefits brought by the Big Stream approach, with respect to Big Data, an alerting application (where an event should be notified to one or more consumers in the shortest possible time) can be considered. The traditional Big Data approach would require an unnecessary pre-processing/storage/post-processing cycle to be executed before the event can be made available to consumers, which would be responsible to retrieve data by polling. The listener-oriented approach, instead, guarantees that only the needed processing will be performed before data are being delivered directly to the listener, thus providing an

effective real-time solution. This general discussion proves that a consumer-oriented paradigm may be better suited to real-time Big Stream applications, rather than simply reusing existing Big Data architectures, which better fit applications that do not have critical real-time requirements.

4.2.1 Acquisition Module

The Acquisition module represents the entry point for external IoT networks of SOs to the Cloud architecture. Its purpose is to collect raw data from different and heterogeneous data sources and make them available to the other functional blocks. It is important to underline that several application-layer protocols can be implemented by SOs. For this reason, the Acquisition module includes a set of different connectors in order to properly handle each protocol-specific incoming data stream.

4.2.2 Normalization Module

Raw data are generally application-dependent, thus a Normalization module has been designed in order to normalize all the collected information and generate a representation suitable for processing. The normalization procedure is made by fundamental and atomic operation on data such as: (i) the suppression of useless information (e.g., unnecessary headers or metadata); (ii) the annotation with additional information; and (iii) the translation of the payload to a suitable format. In order to handle the huge amount of incoming data efficiently, the normalization step is organized with protocol-specific queues and *Exchanges*. An Exchange works as a router in the system and dispatches incoming data to one or more output queues depending on dynamic routing rules. As shown in the normalization section of Fig. 4.7, the information flow originating from the Acquisition module is handled as follows:

- all data streams relative to a specific protocol are routed to a dedicated protocol-specific exchange, which forwards them to a protocol-dedicated queue;
- a normalization process handles the input data currently available on the queue and performs all necessary normalization operations in order to obtain a stream of information units that can be processed by next modules;

- the normalized stream is forwarded to an output exchange; the output of the Normalization block represents the entry-point of the first Exchange inner to the Graph module, that pass it to all the interested listeners of the next levels.

The main advantage of using Exchanges is that queues and normalization processes can be dynamically adapted to the current workload; for instance, normalization queues and processes could be easily replicated to avoid system congestion.

4.2.3 The Graph Framework

In order to overcome the limitations of the "process-oriented" approach previously described and to fit with the Big Stream paradigm, the proposed Cloud architecture is based on a Graph framework. More precisely, is is consider a graph composed by basic building blocks that are self-consistent and perform "atomic" processing on data, but that are not directly linked to a specific task. In such a system, the data flows are based on dynamic graph-routing rules determined only by the nature of the data itself and not by a centralized coordination unit. This new approach allows the platform to be "consumer-oriented" and to implement optimal resource allocation. Without the need of a coordination process, the data streams can be dynamically routed in the network by following the edges of the graph and allowing the possibility to automatically switch off nodes (if some processing units are not required at a certain point) and transparently replicate nodes (if some processing entities are consumed by a significant amount of concurrent consumers).

The proposed directed graph-based processing architecture, adhering to the publish/subscribe model, and the concept of listener, are illustrated in Fig. 4.5. A listener is an entity (e.g., a processing unit in the graph or an external consumer) interested in the raw data stream or in the output provided by a different node in the graph. Each listener represents a node in the topology and the presence and combination of multiple listeners, across all processing units, defines the routing of data streams from producers to consumers. More in detail, the proposed architectural approach is composed by the following entities.

- **Nodes**: processing units (processes) performing some kind of computation on

incoming data, not directly linked to a specific task. A node of the Graph framework can be a listener of one or more streams and a publisher of a new stream for other nodes.

- **Edges**: flows of information linking together various processing units, which are thus able to implement some complex behavior as a whole.

Therefore, nodes in the Graph framework are logically organized in layers, characterized by an increasing degree of complexity. This means that data streams, generated from nodes in a layer of the graph, can be used by nodes in higher layers, generating new streams which can be used in higher layers, and so on.

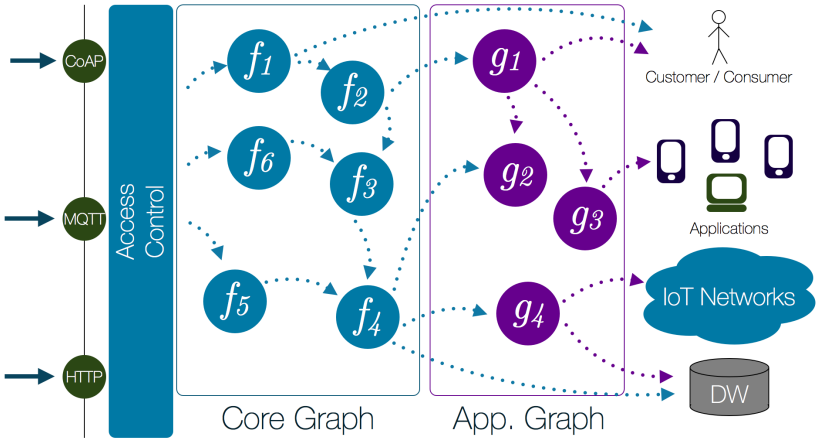


Figure 4.5: The proposed listener-based Graph architecture: the nodes of the graph are listeners; the edges refer to the dynamic flow of information data streams.

The designed graph-based approach allows to optimize resource allocation in terms of efficiency, by switching off processing units that have no listeners registered to them (enabling cost-effectiveness and scalability) and by replicating those processing units which have a large number of registered listeners. The combination of these two functionalities and the concept of listener allow the platform and the overall system to adapt itself to dynamic and heterogeneous scenarios, by properly routing

data streams to the consumers, and to add new processing units and functionalities on demand.

As highlighted, in order to provide a set of commonly available functionalities, while allowing to dynamically extend the capabilities of the system, the graph is composed by concentric layers. Each layer contains two types of nodes, as shown in Fig. 4.6 (a):

- **Core Graph Nodes**: listeners which perform basic processing operations provided by the architecture (e.g., format translation, normalization, aggregation, data correlation, and other transformations).
- **Application Graph Nodes**: listeners that require data coming from an inner graph layer, in order to perform custom processing on already processed data.

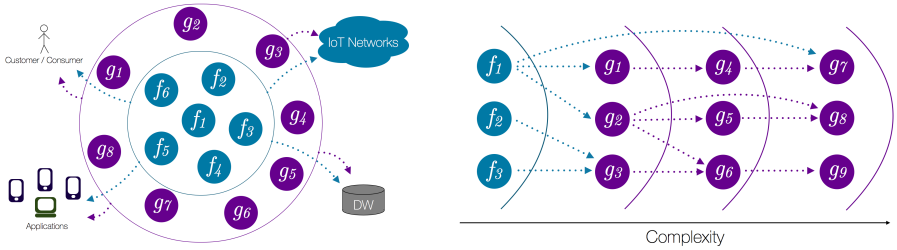
The complexity of processing is directly proportional to the number of layers crossed by the data. This also means that data at an outer graph layer must not be processed again at an inner layer, which also guarantees that processing loops, due to misconfigurations, are avoided by design.

From an architectural viewpoint, as shown in Fig. 4.6, nodes at inner graph layers cannot be listeners of nodes of outer graph layers. In other words, there can be no link from an outer graph node to an inner graph node, but only vice-versa. Same layer graph nodes may be linked together if there is a need to do so.

In particular, a processing unit of the Core Graph layer can be a listener only for other nodes of the same layer (e.g., for x incoming streams) and a source for other Core and Application Graph nodes (e.g., for y outgoing streams). A node of an Application Graph layer can be, at the same time:

- a listener of x incoming flows from Core and/or Application Graph layers;
- a data source only for other y nodes of the Application Graph layers or heterogeneous external consumers.

In the Graph framework, each level is accessible from a level-dedicated Exchange that forwards all data streams to nodes in its level. Each graph node n_i in a specific



(a) (b)

Figure 4.6: (a) The concentric linked Core and Application layers. (b) Basic processing nodes build the Core Graph layer, the outer nodes have increasing complexity.

layer L_k can listen for incoming data stream on a dedicated queue managed by the Exchange of level L_k. If the node n_i, as well as being a consumer, it acts also as a publisher, then its computation results are delivered to the Exchange of level L_k, which is bounded with the Exchange of layer L_{k+1}. Therefore the Exchange in level L_{k+1} can forward streams coming from level L_k to all nodes of level L_{k+1} interested in this kind of data.

In the end, the overall behavior of a task is generated by following a complete path in the graph from a data source to a final consumer. Processing units perform operations that can be reused, thus data produced by a node can belong to several different paths and can be forwarded to all interested listeners. For this reason, in order to optimize the workload, nodes with a large number of listeners can be replicated and nodes with no listeners can be shut down.

4.2.4 Application Register Module

The Application Register module has the fundamental responsibility to maintain all the information about the current state of all graph nodes in the system, and to route data across the graph. In more detail, the Application Register module performs the following operations: (i) attach new nodes or consumer applications interested in some of the streams provided by the system; (ii) detach nodes of the graph that are no more interested in streaming flows and eventually re-attach them; (iii) handle nodes

that are publishers of new streams; and (iv) maintain information regarding topics of data, in order to correctly generate the routing-keys and to compose data flows between nodes in different graph levels. In order to accomplish all these functionalities, the Application Register module is composed by two main components, as shown in Fig. 4.7.

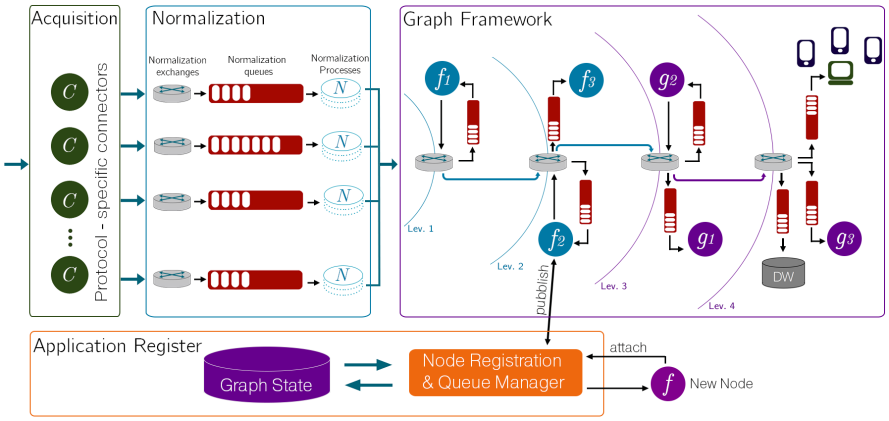


Figure 4.7: The complete Graph Cloud Architecture with reference to the data stream flow between all building blocks, from IoT data sources to final consumers.

The first component is the "Graph State Database" (GSDB), which is dedicated to store all the information about active graph nodes, such as their state, level, and whether they are publishers. The second one is the "Node Registration and Queue Manager" (NRQM), which handles requests from graph nodes or external process, and handles the management of queues and the routing in the system. When a new process joins the Graph framework as a listener, it sends an attach request to the Application Register module, specifying the kind of data to which it is interested. The NRQM module stores the information about a new process in the GSDB and creates a new dedicated input queue for the process, according to its preferences. Finally, the NRQM sends a reference of the queue to the process, which becomes a new listener of the Graph and can read the incoming stream from the input queue. After this registration phase, the node can perform new requests (e.g., publish, detach, get status, etc).

The designed graph-based architecture allows to optimize resource allocation in terms of *efficiency*, by switching off processing units that have no listeners registered to them (enabling cost-effectiveness), and *scalability*, by replicating those processing units which have a large number of registered listeners. The combination of these two functionalities and the concept of listener allow the platform and the overall system to adapt itself to dynamic and heterogeneous scenarios by properly routing data streams to the consumers and add new processing unit and functionalities on demand. In Fig. 4.7, all the architecture modules with the complete flow of information through all steps described above are presented in detail.

4.3 Implementation

Although the Big Stream architecture outlined above can be deployed on a single server, exploiting it on the Cloud allows to improve the system's scalability and to manage the workload with a huge number of data sources and processing nodes. Another important benefit, brought by the use of Cloud, is that it provides a common platform in which data streams can be shared among several actors (e.g., IoT data sources, developers or consumers), in order to build useful services for different consumers. Therefore, this architecture is mainly intended for developers interested in building applications based on IoT networks data, with real-time constraints and low overhead. Developers authenticated on the platform can thus customize paths in the graph through the definition and deployment of new processing nodes. Analyzing the Cloud components of the platform, the service model is well suited to the SaaS paradigm, providing the following services for developers: (i) node upload/deletion functions, based on the insertion or deletion of processing nodes; (ii) stream status function, to get the list of all streams available in the graph; (iii) data source upload/deletion functions, to load or remove an external data source (i.e., a new IoT provider). It is important to observe that, by accessing the Big Stream architecture, each developer can operate on data streams coming from IoT networks which he/she does not own.

In [126], a first implementation of the Big Stream architecture with open source

technologies has been presented, in which three main modules concur in forming the entire Big Stream system: (i) Acquisition and Normalization modules for the management of the incoming raw data; (ii) Graph framework; (iii) Application Register entity. The implementation of a novel Cloud architecture for Big Stream applications based on standard protocols and open source components, which provides a scalable and efficient processing platform for IoT applications, has been carried out by deploying a VM equipped with 2 CPUs, 2GB RAM and Linux Ubuntu 12.04 64-bit. Since the architecture is based on a queue-communication paradigm, an instance of RabbitMQ [132], an open-source queue server implementing the standard Advanced Message Queuing Protocol (AMQP) [133], has been used. RabbitMQ provides a multi-language (Java, PHP, Python, C, etc.) and platform-independent API.

4.3.1 Acquisition and Normalization Modules

The system needs an input block capable to handle external incoming raw data, through different application-layer protocols. Data must then be processed and structured, in order to be managed by the graph processes.

Acquisition Module

The Acquisition module represents the entry point, for external IoT networks of SOs, to the Cloud architecture. Its purpose is to receive incoming raw data from heterogeneous sources, making them available to all subsequent functional blocks. As mentioned before, about IoT models, several application-layer protocols can be implemented by SOs; adhering to this idea, the Acquisition module has been modeled to include a set of different connectors, in order to properly handle each protocol-specific incoming data stream. Considering the main and most widespread IoT application-layer protocols, the current implementation of the Acquisition interfaces supports HTTP, CoAP and MQTT.

For the sake of scalability and efficiency, in the module implementation an instance of NGINX [134] server has been adopted as an HTTP acquisition server node. The server is reachable via the default HTTP port (80), working with a dedicated

PHP page, as processing module, which has been configured to forward incoming data to the inner queue server. NGINX server has been chosen, instead of the prevailing and well-known open source Apache HTTPD [135] server, because it uses an event-driven asynchronous architecture to improve scalability and, specifically, aims to guarantee a high performance even in the presence of a critical number of requests.

The CoAP acquisition interface has been implemented using a Java process, based on a mjCoAP server instance, waiting for incoming raw messages, and connected to the RabbitMQ queue server, passing it injected elements. Indeed, since the proposed architecture is Big Stream-oriented, a well-fitting messaging paradigm is given by queue communication; therefore, in the developed platform an instance of RabbitMQ queue broker was adopted.

The MQTT acquisition node is built by implementing an ActiveMQ [136] server through a Java process which listens for incoming data over a specific input topic (*mqtt.input*). This solution has been preferred over other existing solutions (e.g., the C-based server Mosquitto [137]) because it provides a dedicated API that allows a custom development of the component. The MQTT acquisition node is also connected to the architecture's queue server. In order to avoid potential bottlenecks and collision points, each acquisition protocol module has a dedicated Exchange module and queue (managed by RabbitMQ), linked together with a protocol-related routing key, ensuring the efficient management of incoming streams and their availability to the subsequent nodes. In the described implementation, an Exchange is a RabbitMQ component which acts as a router in the system and dispatches incoming messages to one or more output queues, following dynamic routing rules.

Normalization Module

Since incoming raw data are generally application- and theme-dependent, a Normalization module has been designed in order to normalize all the collected information and generate a representation suitable for processing.

As shown in the normalization section of Fig. 4.8, the information flow originated by the Acquisition module is handled as follows.

- All protocol-specific data streams are routed to a dedicated protocol-dependent Exchange, which forwards them to a specific queue.

- A normalization process handles the input data currently available on that queue and performs all necessary normalization operations in order to obtain a stream of information units that can be processed by subsequent modules.

- The normalized stream is forwarded to an output Exchange.

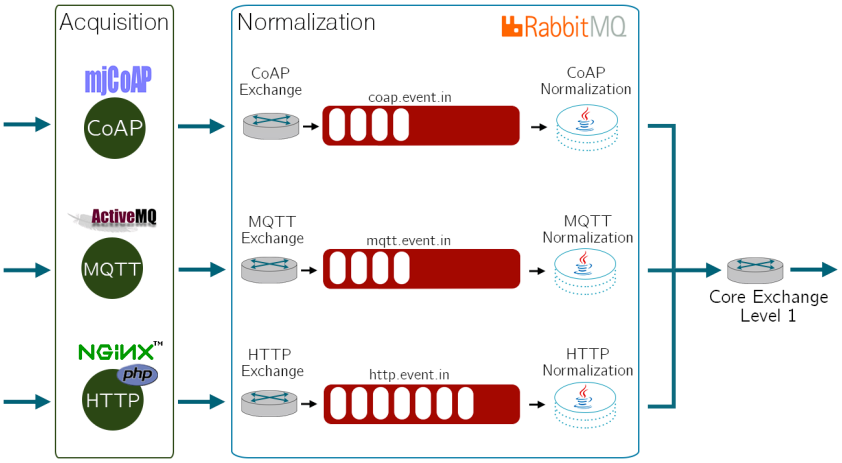


Figure 4.8: Detailed representation of Acquisition and Normalization blocks.

Each normalization node has been implemented as a Java process, which analyzes incoming raw data extracted from a queue identified through a protocol-like routing key (e.g., *<protocol>.event.in*), leaving unaltered the associated routing key, which identifies the originator SO's protocol. The received data are fragmented and encapsulated into a JSON-formatted document, which provides an easy-to-manage format.

At the end of the normalization chain, each processor node forwards its new output chunk to its next Exchange that represents the entry-point of the Graph framework, promoting data flows to next layers of the proposed architecture.

4.3.2 Graph Framework

The Graph framework is composed by an amount of different computational processes representing a single node in the topology; layers are linked together with frontier Exchanges, forwarding data streams to their internal nodes.

Incoming messages are stored into active queues, connected to each Exchange. Queues can be placed into the Core Graph, for basic computation, or into Application Graph, for enhanced data treatment. Layers are connected with one-way links with their own successor Exchange by using the binding rules allowed by queue manager, ensuring proper propagation of data flows and avoiding loops. Each graph layer is composed by Java-based Graph nodes, that can be Core or Application nodes, dedicated to process data provided by the Graph layer's Exchange. Messages, identified by a routing key, are first retrieved from the layer's Exchange, then processed, and finally sent to the target Exchange, with a new work-related routing key, as depicted in Fig. 4.9.

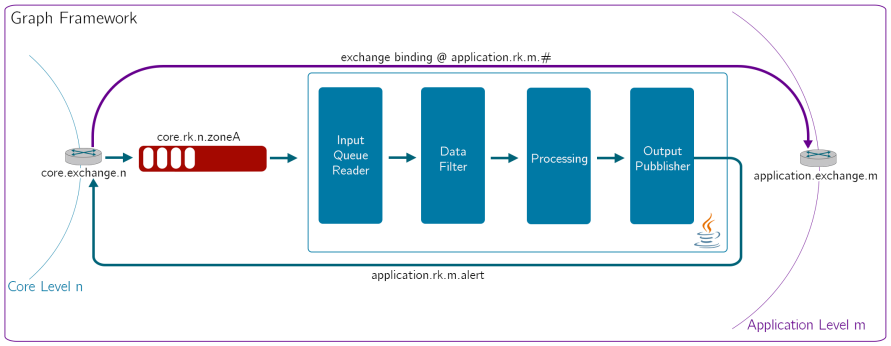


Figure 4.9: Interaction between Core and Application layers with binding rule.

If the outgoing routing key belongs to the same incoming graph layer, data object or stream stays within the same Exchange and becomes available for other local processes. If the outgoing routing key belongs to an outer graph layer, then data are forwarded to the corresponding Exchange, and finally forwarded by following a binding rule and assuring data flow. Each graph node, upon becoming part of the system, can specify if it acts as a data publisher, capable of handling and forwarding data to

its layer's Exchange, or if it acts as data consumer only. A data flow continues until it reaches the last layer's Exchange, responsible to manage the notification to the external entities that are interested in final processed data (e.g., DWs, browsers, smart entities, other Cloud graph processes, etc).

4.3.3 Application Register Module

The overall architecture is managed by the Application Register module, that has been implemented as a Java process that has the role to coordinates the interactions between graph nodes and external services, like the RabbitMQ queue server and the MySQL [138] database. It maintains and updates all information and parameters related to processing unit queues.

Firstly, the Application Register starts up all the external connections, then it activates each layer's Exchange, binding them with their successors. Finally, it proceeds with the activation of a Jetty HTTP server, responsible for listening and handling all Core and Application nodes requests, coming in a RESTful way (as depicted in Fig. 4.10).

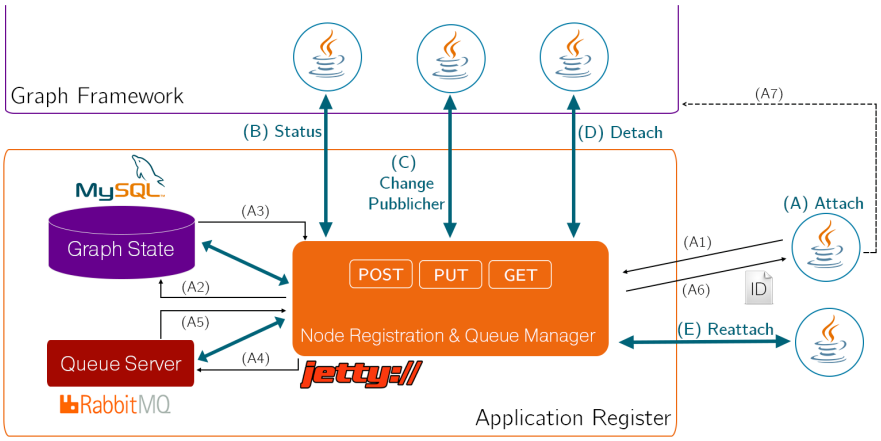


Figure 4.10: Detail of Application Register module, with possibles actions required by graph nodes, deepening steps for ATTACH request.

- Attach new nodes or consumers, interested in some of the streams provided by the platform (A).
- Manage status requests from nodes (B).
- Detach (D) nodes from the Graph framework, when they are no longer interested in receiving flows, and, possibly, re-attach them (E).
- Handle change publishing policies and nodes that are publishers of new streams (C).
- Maintain information regarding topics of data, in order to correctly generate the routing keys, and to compose data flows between nodes in different Graph layers.

4.4 Performance Evaluation

In order to assess the feasibility, scalability, and the performance of the proposed architecture, the implementation has been evaluated in [126] through an experiment in a real-world smart parking scenario. The data traces used for the evaluation of the proposed architecture have been provided by Worldsensing [139] from one of the company's deployments in a real-life scenario, used to control parking spots on streets. The traces are a subset of an entire deployment (more than 10,000 sensors) with information from 400 sensors over a 3-month period, forming a dataset of more than 604k parking events.

Each dataset item is represented by: (i) sensor ID; (ii) event sequence number, relative to the specific sensor; (iii) event timestamp; and (iv) parking spot status (*free*/*busy*). No additional informations about parking zone are provided. Therefore, in order to create a realistic scenario, parking spot sensors have been separated into 7 groups, representing different parking zones of a city. This parking spot-city zone association is stored into an external database.

4.4.1 Experimental Setup

The parking dataset has been used in the Cloud infrastructure using a Java-based data generator, which simulates the IoT sensors network. The generator randomly selects an available protocol (HTTP, CoAP, or MQTT) and periodically sends streams to the corresponding acquisition node interface. Once the data has been received by the acquisition layer, they are forwarded to the dedicated normalization Exchange, where corresponding nodes enrich incoming data with platform-specific details. With reference to the selected scenario, the normalization stage adds parking zone details to input data, retrieving the association from an external database. Once the normalization module has completed its processing, it sends the structured data to the Graph framework, allowing to further process the enriched data stream.

The Graph framework considered in this experimental setup is composed by 8 Core layers and 7 Application layers, within which different node topologies are built and evaluated. Processed data follow a path based on routing keys, until the final external listener is reached. Each Application node is interested in detecting changes of parking spot data, related to specific parking zones. Upon a change of the status, the Graph node generates a new aggregated descriptor, which is forwarded to the responsible layer's Exchange, which has the role to notify the change event to external entities interested in the update ($free \rightarrow busy$, $busy \rightarrow free$). The rate of these events, coming from a real deployment in a European city, respects some rules imposed by the company, and might seems low for a performance evaluation. Thus, in order to stress enough the proposed Big Stream Cloud system, the performance is evaluated by varying the data generation rate in a proper range. In other words, a specific rate for incoming events has been forced, without taking into account real parking spots timestamps gathered from the dataset.

4.4.2 Experimental Results

The proposed architecture has been evaluated, using the testbed previously described, by varying the data rate from 1 msg/s to 100 msg/s. The evaluation consists in assessing the performance of the acquisition stage and the computation stage.

First, performance is evaluated by measuring the time difference (dimension: [ms]) between the instant at which data are sent from a data generator to the corresponding acquisition interface and the instant at which the data are enriched by normalization nodes, thus becoming available for the first processing Core node. The results are shown in Fig. 4.11. It can be observed that the acquisition time is slightly increasing, but it is around 15 ms at all considered rates.

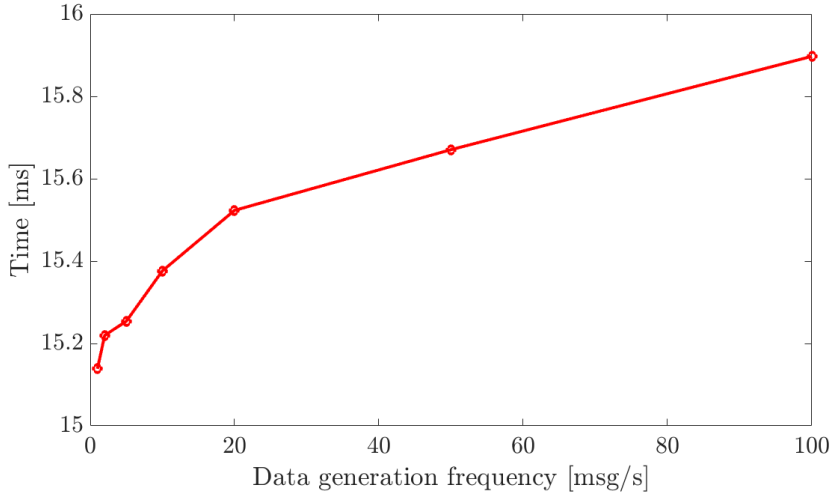


Figure 4.11: Average times (dimension: [ms]) related to the Acquisition block.

The second performance evaluation has been carried out by measuring the time (dimension: [ms]) between the instant in which enriched data become ready for processing activities, and the time when the message ends its Graph framework routes, becoming available for external consumers/customers. In order to consider only the effective overhead introduced by the architecture, and without considering implementation-specific contributions, performance results were obtained by subtracting the processing time of all Core and Application nodes. Finally, these times have been normalized over the number of computational nodes, in order to obtain the per-node overhead introduced by the architecture, in a way that is independent of the specific routing and topology that were implemented.

The results, shown Fig. 4.12, have been calculated using the following expres-

sion:

$$T_{processing_{freq}} = \frac{T_{out} - T_{in} - \sum_{k=1}^{N} GP_k}{N-1}$$

in which: T_{out} is the instant at which parking data reach the last Application layer; T_{in} indicates the instant in which normalized data comes to first Core layer; and GP_k is the processing time of a Graph process $k \in 1, ..., N$.

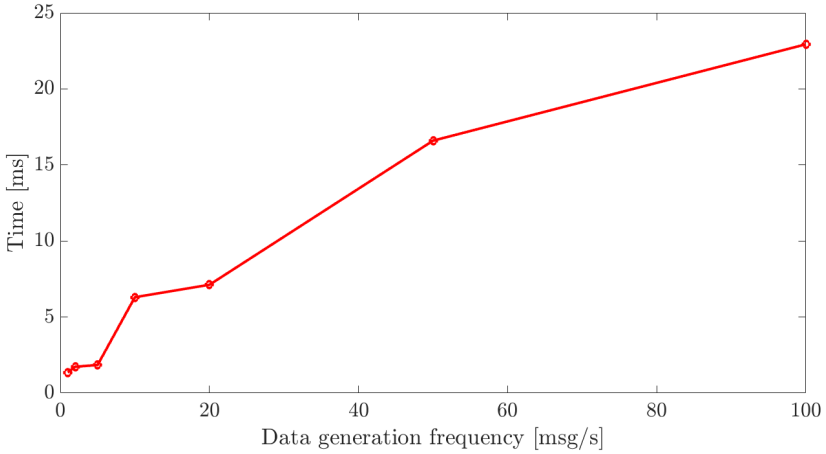


Figure 4.12: Average times (dimension: [ms]) related to Graph framework processing block.

Fig. 4.13 shows how $T_{processing}$ values grow increasing the data generation frequency (from 10 msg/s to 100 msg/s). Each curve is related to a different Graph topology.

Fig. 4.14 shows how $T_{processing}$ values grow increasing the number of nodes composing the Graph topology (from 20 to 50 nodes). Each curve in Fig. 4.14 is related to a different value of frequency rate.

4.5 Solutions and Practical Consideration

The described Big Stream architecture is designed with reference to a specific IoT scenario with strict latency and real-time requirements, namely a Smart City-related

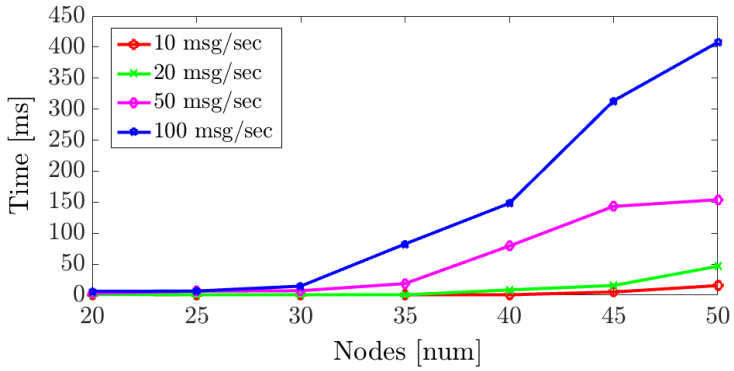


Figure 4.13: Average times (dimension: [ms]) related to Graph framework processing block, showing per-node time, varying data generation rate, for each subset of nodes deployed into the Graph topology.

smart parking scenario. There are several possible use-cases and applications fitting this scenario, alerting or real time monitoring applications.

The work of [140] shows how SCs are having difficulties in real deployment, even though obvious factors justify the necessity and the usefulness of making cities smarter. The authors of [140] analyze in detail the causes and factors which act as barriers in the process of institutionalization of SCs, and propose an approach to make SCs become a reality. More in detail, three different stages are advocated, in order to deploy SCs technologies and services.

- **The bootstrap phase**: this phase is dedicated to offer services and technologies that are not only of great use and really improve urban living, but also offer a return on investments. The important objective of this first step is, thus, to set technological basis of the infrastructure and guarantee the system long life by generating cash flows for future investments.

- **The growth phase**: in this phase, the finances generated in the previous phase are used to ramp up technologies and services which require large investments and not necessarily produce financial gains but are only of great use for consumers.

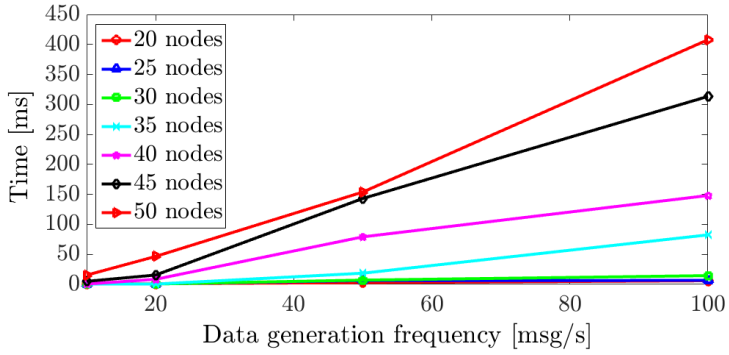


Figure 4.14: Average times (dimension: [ms]) related to Graph framework processing block, showing per-node time, varying the subset of nodes deployed into the Graph topology, for each evaluated data generation frequency.

- **The wide adoption phase**: in this third phase, collected data are made available through standardized APIs and offered by all different stakeholders to third party developers in order to create new services. At the end of this step, the system becomes self-sustainable and might produce a new tertiary sector specifically related to services and applications generated using the underlying infrastructure.

With reference to the third phase, in [140] three main different business models to handle the delivery of informations to third parties are proposed.

- **The Apple App Store-like model**: developers can build their Apps using a set of verified APIs after a subscription procedure which might involve some subscription fee. IoT operators can hold a small percentage of gains of Apps published in Apple App Store and/or Google Play Store.

- **The Google Maps-like model**: the percentage fee on Apps sales price is scaled according to the number and granularity of the queries to deployed APIs.

- **The Open Data model**: this model grants access to APIs in a classical Open Data vision, without charging any fee to developers.

The Big Stream architecture previously described is compatible with the steps highlighted in [140] and, more specifically, it can adopt the Google Maps-like model, in which infrastructure APIs make available different information streams with different complexity layers. The graph architecture, moreover, gives another opportunity to extend the business model, as developers can use available streams to generate a new node of the graph, and publish a new stream for the system.

Finally, it is possible to observe that the interactions between IoT developers and the proposed Cloud architecture are similar to those provided by Node-RED [141], a Web-based application, running on Node.js engine, which allows developers to create IoT graphs, wiring together hardware devices, APIs, and online services.

4.6 Analysis of Security Challenges

To fulfill all the scenarios in which the Big Stream paradigm can be applied and, in general, the IoT vision, some issues (maybe with a relevant impact on the business model) must be addressed, such as security. Securing the IoT and the systems that adhere to this paradigm requires further research and entails both authentication and authorization mechanisms to address security and privacy problem, and this represents a really critical aspect in several IoT applications.

Security is a central aspect to be taken into account, in order to enhance the overall reliability of the Big Stream architecture and the processing control. Security features, including authorization, authentications and confidentiality, should be integrated into the architecture, in order to make the implementation complete and usable.

To provide guarantees at input stages, an optimal solution could be represented by the introduction of an authorization module, which tokenizes incoming data adopting an asymmetric security paradigm, to sure that raw data providers are authorized to provide information. This entails both processing module and interaction with external entities. It is possible to adopt different policies related to authentication and/or authorization on data sources, e.g., based on well-known and standard solutions such as OAuth, avoiding data stream malicious alterations and following negative conse-

quences, that could affect both processing results and platform reliability. Looking for a reliable behavior at the output stage, a good solution could be reached by introducing an Accounting/Authentication/Authorization (AAA) module, which manages and controls the acceptance of consumers, providing some cryptographic functionalities, to check security level of each entity.

Addressing the security problem in the Big Stream Graph-based Cloud system entails a wide approach, owing to different needs of each specific component involved. In the proposed Graph architecture is assumed that streams generated by nodes are "open" and potentially accessible by any interested subscriber. However, this assumption can not meet the security requirements of all consumers or developers accessing the architecture. Moreover, security management may become necessary in application scenarios that require to control or filter subscribers' access to one or more streams. For this reason, the Big Stream architecture preliminary proposed has to be extended in order to take into account also security aspects. In particular, the extension introduces additional modules needed to make the proposed Graph architecture able to handle both "secure" and "open" data streams, focusing, in particular, on solutions able to decouple security roles and management purposes (e.g., OAuth paradigm).

In Fig. 4.15, the main security building blocks introduced in the Big Stream architecture are shown. The main components, in correspondence to which security mechanisms are required, are explicitly indicated.

The enhanced Graph architecture provides security by means of the following two modules.

- **Outdoor Front-end Security (OFS)** module, which carries out security operations which could be applied a-priori, before receiving data from a generic external source, as well as before a final consumer can start interacting with the Graph-based Cloud platform.

- **In-Graph Security (IGS)** module, which adopts security filters that could be applied inside the heart of the Big Stream platform, in order to make processing nodes able to control accesses to the streams generated by internal computa-

tional modules.

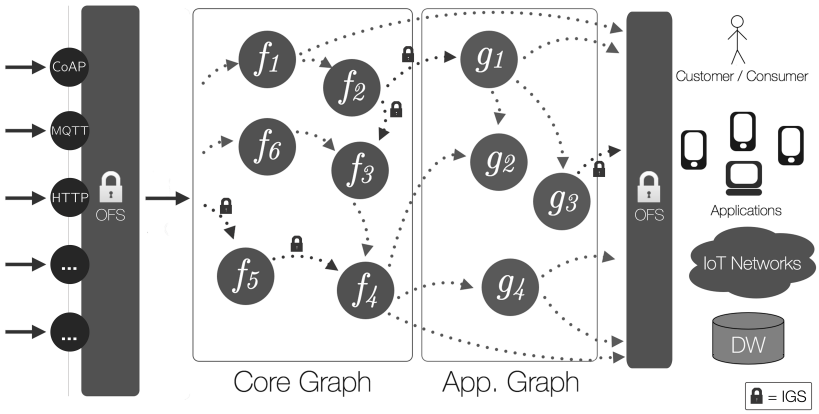


Figure 4.15: Main building blocks of the proposed Big Stream-based architecture. Nodes in the Graph are listeners, edges (that can be "open" or "secure") between nodes represent dynamic flows followed by information streams.

The OFS module is crucial for infrastructure safety: its role includes monitoring the access to the platform and authorizing the information flows coming from or directed to external entities. On one side, OFS must verify and authorize only desired external input data sources, allowing them to publish raw streams inside the IoT system. On the other hand, OFS is required to secure outgoing streams, generated by different layers of the Graph platform itself, authorizing external consumers to use, if needed, the processed streams.

Consider, as example, the case of the company *C* that owns a set of particular sensors, and wishes to become an IoT stream source for the Graph-based Cloud platform. However, this company requires: (i) to sell sensed data only to a specific subset of customers, in order to protect its commercial interests, and (ii) to reach a profit from these sales. Therefore, the OFS module is strictly related to sensors and devices, at the input side, and to customers SOs, at the output stage, so that it becomes protocol-dependent and can be adapted to the specific technologies supported by the target devices.

The IGS module is not related to the OFS module, as it acts exclusively at the heart of the Big Stream architecture coordinating and managing inner inter-node interactions. The IGS module must be implemented inside single processing nodes enabling them to define a set of rules which describe what entities may become listeners of a generated stream. Referring to the Graph architecture, shown in Fig. 4.15, edges in the Graph can be classified as follows:

- "open" edges: data streams, generated by both Core or Application nodes in the Graph platform, that can be forwarded to all interested listeners without the need of isolation or access control;
- "secure" edges: data streams that should comply with some specified rules or restrictions, which describe all possible consumers of generated data.

As an example, consider company *C*, which provides its sensors as data sources and has notified the architecture that the streams produced by its sensors should be secured. The integration of security modules in the IoT architecture entails modifications in the structure and into the modules of the first (not secured) Big Stream architecture previously detailed.

An analysis of each module composing the Graph architecture will be presented hereafter, in order to explain how security mechanisms can be embedded and managed. In particular, the OFS module, which supports the Acquisition and Normalization modules on authorization of external entities, is introduced. Moreover, an enhanced version of the Application Register module is described, in order to underline the management of secure interaction with processing nodes. Finally, an overview inside Graph nodes, analyzing how security has been applied in processing stages, is presented.

4.6.1 Securing Acquisition and Normalization with OFS Module

The Acquisition and Normalization modules, shown in Fig. 4.16, represent the entry point, for external sources (e.g., SOs deployed in different IoT networks), to the proposed Big Stream architecture.

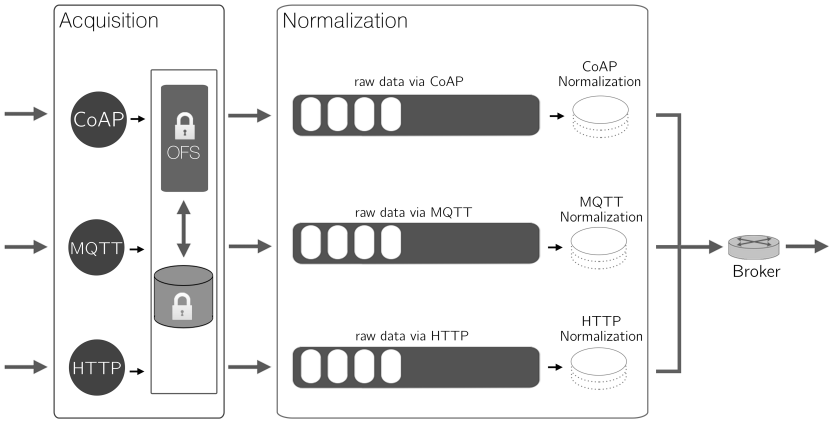


Figure 4.16: The OFS module manages security in the Acquisition and Normalization blocks, interacting with an authentication storage entity containing data sources identities.

As stated before, SOs can communicate using different protocols. For this reason, the Acquisition block has to include a set of different connectors, one for each supported protocol, in order to properly handle each protocol-specific incoming data stream. As shown in Fig. 4.16, these modules must cooperate with the OFS module, which has to be activated before an external source becomes able to operate with the Graph platform. At the Acquisition stage, in order for the proposed IoT platform to support both "open" and "secure" communications, protocol-specific security mechanisms have to be implemented at proper layers (e.g., at the network layer through IPSec, at the transport layer through TLS/DTLS, at the application layer through S/MIME or OAuth). As stated before, the current implementation of the IoT-oriented Big Stream architecture supports different application protocols at the Acquisition stage, namely: MQTT, HTTP, and CoAP. In order to secure all communications with these protocols, different protocol-specific policies need to be introduces. In [66], an OAuth-based secure version of the MQTT protocol is proposed, showing that MQTT complies also with n-legged OAuth protocol.

This means that the proposed IoT platform can provide a good way to authenticate external data providers, adopting open-source and well-known solutions. Therefore

the OFS module can be secured by OAuth, being employed according to specific communication protocols supported by heterogeneous IoT SOs.

A suitable solution to provide authorization in IoT scenarios is IoT-OAS [142], which represents an authorization framework to secure HTTP/CoAP services. The IoT-OAS approach can be applied by invoking an external OAuth-based Authorization Service (OAS). This approach is meant to be flexible, highly configurable, and easy to integrate with existing services, guaranteeing: (i) lower processing load with respect to solutions with access control implemented in the SO; (ii) fine-grained (remote) customization of access policies; and (iii) scalability, without the need to operate directly on the device.

Following the previous example, company *C*, to became a secured IoT data source, selects one of the supported protocols (HTTP, CoAP or MQTT) to send raw data stream in the secured version.

4.6.2 Enhancing Application Register with IGS Module

One of the main motivations to internally secure a system is the need to secure some of its operations, as well as to isolate some processing steps of the entire stream management. The security features should be coordinated by the Application Register module, which maintains and manages interactions between inner Graph nodes of the IoT platform using different communication protocols, as requested by the architecture itself.

In order to accomplish the operational functionalities listed previously, the Application Register module has two main components, previously introduced and shown in Fig. 4.17: (i) the GSDB module, responsible to maintain all information about the current Graph status, and (ii) the NRQM module, which is responsible to manage communications with existing Graph nodes, as well as with external entities that ask to join the Big Stream architecture.

To add security features to these modules, the Application Register defines entities and modules specifically related to security management and coordination. As shown in Fig. 4.17, the Application Register is composed by the following additional modules:

- the **Policy Manager and Storage Validator (PMSV)** module, responsible for managing and verifying authorization rules, interacts with a storage element, which persistently keeps updated authorization policies;
- the **Graph Rule Authorization Notifier (GRAN)** module, which interacts with publisher Graph nodes and verifies if listener nodes are authorized to receive streams associated with specific topics;
- the **Persistent Security Storage Container (PSSC)** module, which maintains authorization rules specified by publisher Graph nodes (e.g., a non-relational database).

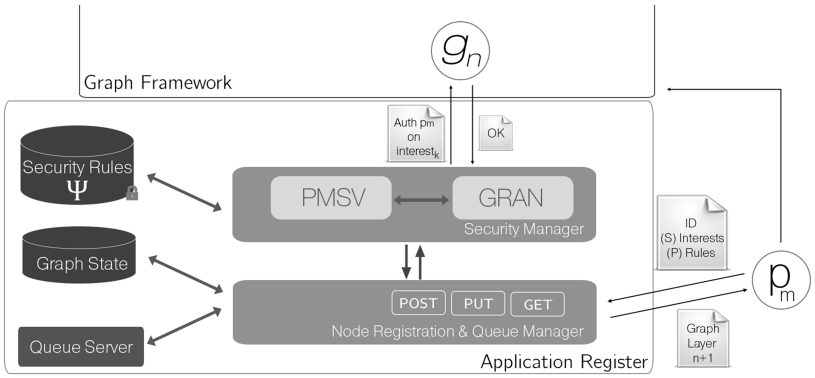


Figure 4.17: The Application Register module structure with security elements. PMSV and GRAN modules interact with a storage entity to manage authorization in the Graph.

While a processing node p_m asks to register to the IoT platform, requiring to join the Graph, after authentication (e.g., using a username/password pair, cryptographic certificates, ACLs, OAuth), there are two cases that require the use of security mechanisms and involve the defined modules:

- a registration request coming from a node that is willing to become a publisher node for a secured stream (e.g, an Application node created by developers of company *C*);

- a registration request sent by a node which asks to be attached as a listener for some streams.

In the first case, when an external process p_m requests to register to the Graph architecture, in order to secure one or more of its own streams, it updates the PMSV module. After indicating its published topics, it specifies some policies and rules, to be stored, together with the assigned operative Graph layer, into the PSSC module, by the PMSV module itself. These rules will be checked in case of future subscription requests for the node.

In the second case, an external process p_m, upon issuing a request to attach to the Graph platform and to become a node, provides information related to its identity and also specifications about its interests, on which p_m asks to subscribe.

The Application Register module, after having identified the Graph layer into which the new node could be placed, takes charge of these interests specifications, passing then to PMSV, that acts as follows.

- For each provided $interest_k$, the PMSV module interacts with the PSSC module, making a lookup for a matching between interest and stored publishing policies, and refining this lookup with layer matching:

$$Match = \{layer = x \text{ OR } layer = (x+1)\} \text{ AND } \{interest_k \in \Psi\}$$

  where: x stands for identified listener Graph layer; $interest_k$ indicates each single specified interest, extracted from attaching request; Ψ represents the PSSC; and *Match* contains a list of publisher nodes that have to authorize subscriptions.

- If *Match* contains some results (e.g., g_m node), these are forwarded to the GRAN module, which interacts with discovered publisher nodes, sending them the identity of the requesting listener node and asking them to allow or deny the subscription to the requested topics. This response is forwarded back to the GRAN, that analyzes it and, in compliance with the Application Register module, authorizes or rejects the listener Graph node subscription.

In order to better explain the behavior of the Application Register module, in relation to the join operation of an external entity that asks to become a Graph listener, in the Algorithm 4 the interactions between this external entity and the Application Register are detailed through a pseudo-code representation.

The processing nodes in the Graph architecture can be, at same time, listeners as well as publishers, so that the previously detailed mechanisms could be applied together, without any constraint on the execution order. The flows shown in Fig. 4.17 represent the interactions related to this mixed case. The rule on node authority, restricted to the same layer and to next one, decreases lookup times in rules matching execution.

Moreover, external SOs producers could also request a totally "secure" path, from source to final consumer. These constraints have a higher priority than policies defined by publisher Graph nodes, being forced by the stream generators. In this way, these external priority rules are stored into the PSSC as well, and when a new Graph node registers to the proposed IoT platform, its Graph-related policies are left out, forcing these new nodes to comply with these external rules.

4.6.3 Securing Stream inside Graph Nodes

As already highlighted, the Graph framework is composed by several processing entities, which perform some kind of computation on incoming data, representing a single node in the Graph topology.

The connection of multiple listeners across all processing units defines the routing path of data streams, from producers to consumers. All nodes in the Graph can be input listeners for incoming data and output producers for other successor Graph nodes.

Since each Graph node is owner of the stream generated by its processing activity, it is reasonable to assume that it could decide to maintain its generated stream "open" and accessible to all its interested listeners, instead of applying securing policies, isolating its own information and defining a set of rules that restrict the amount of authorized listeners nodes. In this latter case, a "secure" stream should be created and encrypted using the algorithms selected by the owner. Each listener is thus

required to decrypt incoming data before performing any processing. These encryption/decryption operations could be avoided if listeners adopt homomorphic encryption [143], that allows to carry out computations on ciphertext, instead of on plaintext, generating an encrypted result that matches with one performed on the plaintext, without exposing the data to each of different steps chained together in the workflow. Homomorphic encryption allows to execute computation in the encrypted domain, providing end-to-end security, avoiding encryption/decryption hop-by-hop needs.

The modules inside a Graph node are shown in Fig. 4.18: the broker of the Core layer *n* forwards streams to interested Graph nodes, forwarding these data into the input queue of the single node. The output stream, generated by the processing of this node, will be sent to the same broker of the Core layer *n*, which is linked to the broker of the next Graph layer, that "spreads" generated streams to all interested nodes. Some of these modules will be activated only in specific situations. In particular, the illustrated node acts as listener of a "secure" data stream, so it has to decrypt an incoming message, activating the decryption module. Moreover, this Graph node acts also as a producer of a "secure" stream and, then, it has to encrypt its processed streams with the encryption module before forwarding it, thus hiding the stream from other unauthorized listener Graph nodes. Referring to the previously described example, this is the case in which a Graph node, that is already a listener of the secured stream owned by company *C*, wants to secure the stream generated by its processing.

It is important to point out that each Graph node controls its generated flow with a visibility of only one step. This means that a listener of a "secure" flow can publish an "open" stream, and vice-versa, thus producing "hybrid" path combinations, that across a flow from IoT source to final consumer produce a combination of "secure" and "open" steps.

Referring to the example of company *C*, which generates a "secure" stream with data coming from its sensors, an IoT developer can decide to create a new Graph node listening from both the "secure" stream of company *C* and the stream of another company *D*. The processing unit of the new Graph node, the developer can aggregate and transform input streams, generating new and different output streams, which can be

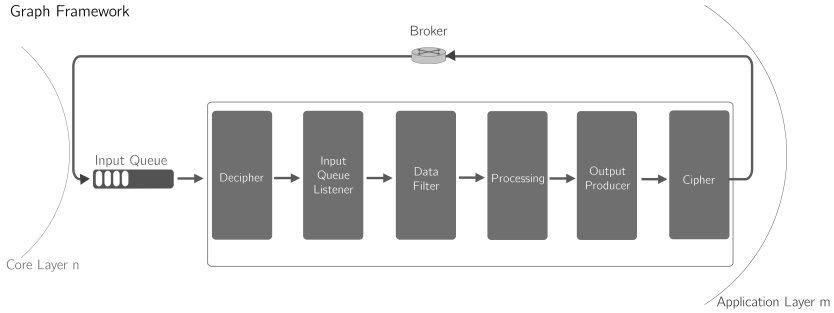


Figure 4.18: Detail of the structure of a single Graph node: the decryption module is activated when the node is a listener of a secured stream, while the encryption module is activated if the node generates a secured stream.

published in an "open" mode, since the developer is the owner of this new produced stream.

According to the inner organization of the IoT architecture, there could exist a parallelism between actors enrolled in Graph Framework and OAuth roles, as illustrated in Table 4.1. More precisely, OAuth roles could be detailed as follows.

Table 4.1: Comparison between Graph framework actors and OAuth roles.

Graph framework actor	OAuth role
Publisher Graph node, owner of the outgoing stream	Resource Owner
Listener Graph node, willing to subscribe to interested topics	Consumer
Infrastructure routing element (Broker in a pub/sub paradigm)	Provider

- **Resource Owner**: the entity which owns the required resource and has to authorize an application to access it, according to authorization granted (e.g., read/write permission).
- **Consumer**: the entity that wants to access and use the required resource, operating in compliance with granted policies related to this resource.
- **Provider**: the entity that hosts the protected resource and verifies the identity

of the Consumer that issues an access request to this resource.

As previously stated, each Graph node can apply security to its streams, using encryption and decryption modules. The security mechanisms leave a few degrees of freedom to developers, who can adopt self-made solutions (e.g., using OAuth tokens) to secure streams, as well as rely on already secured protocols, thus adopting well-known and verified solutions. An overall view of the envisioned IoT architecture is shown in Fig. 4.19, highlighting all component modules and their interactions.

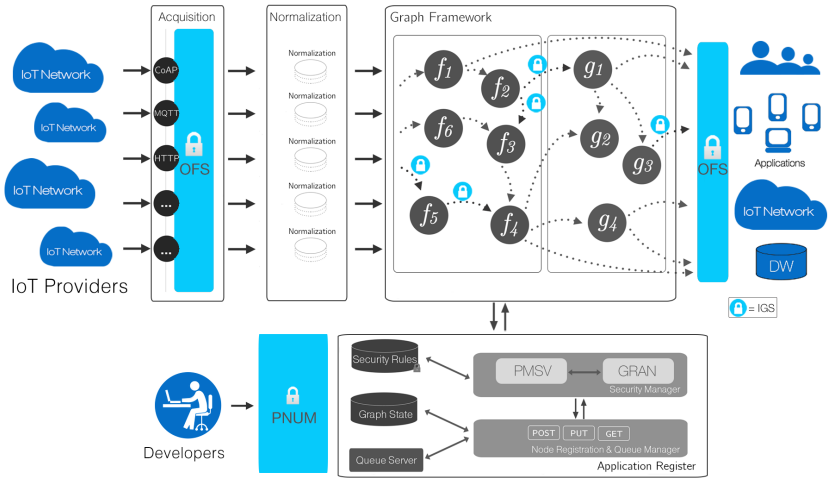


Figure 4.19: Complete IoT-oriented Big Stream architecture, including proposed security modules and showing different interactions, from incoming stage to final consumer notification.

4.7 Evaluation of the Secured Architecture

The secured Big Stream architecture has been implemented and evaluated considering the same smart parking use-case of the previously detailed evaluation, in which the Graph considered is composed by 8 Core layers and 7 Application layers, within which different graph topologies (from 20 to 50 nodes) are built and evaluated.

The proposed architecture has been evaluated by varying the incoming data stream generation rate between 10 msg/s and 100 msg/s. The first evaluation, which represents a benchmark for the performance analysis of the IoT-oriented architecture, has been made using the platform without security mechanisms. Then, security mechanisms have been introduced in the Graph framework module, in order to assess the impact of a security stage on the overall architecture.

The first performance evaluation has been conducted by measuring the delay (dimension: [ms]) between the time instant at which normalized data are injected into Graph framework and the time instant at which the message reaches the end of its routes, becoming available for external consumers/customers. In order to consider only the effective overhead introduced by the architecture, and without taking into account implementation-specific contributions, performance results were obtained by subtracting the processing time of all Core and Application nodes. Finally, these times have been normalized over the number of computational nodes, in order to obtain the per-node overhead introduced by the architecture, in a way that is independent of the implemented routing and topology configurations.

The second performance evaluation has been conducted adopting the same structure of the unsecured implementation, but introducing security mechanisms inside Graph nodes, through the adoption of symmetric encryption scheme, in order to encrypt/decrypt operations previously mentioned. In order to guarantee a trade-off between security level and reliability, the choice fells on Advanced Encryption Standard (AES) algorithm [144], in its 256-bit key version. AES is a symmetric block cipher cryptosystem based on a "substitution and permutation" combination, working on 128-bit blocks. The chosen key size specifies the number of repetitions of transformation rounds that convert the input, applying several processing stages and depending on the encryption key. The strength of AES256 is derived by its key space, with 10^{77} possible 256-bit keys, which affects the time needed to made a successful brute-force attack to a system implementing this cipher.

Moreover, in order to perform the second evaluation, a new version of the Core and Application processing nodes has been implemented, applying security at both input and output stages of the single Graph node, and defining the following behavior:

- if the processing node has received an AES256 decryption key from the Application Register module, it have to uses this key to decrypt incoming messages, returning a plaintext useful for processing operations;

- if an AES256 encryption key has been provided by the Application Register to the Graph node, then it encrypt the processed stream before forwarding it to its proper Exchange, using this symmetric key as encryption secret.

This security model is applicable also to the previously considered example, in which the company *C* would like to secure its paths into the Graph framework. The second evaluation has been made providing encryption and decryption keys to all Graph nodes, in order to secure all the intermediate routes followed by streams owned by that company. The results of the performance evaluations outlined above, carried out using different topologies obtained by varying the subset of nodes deployed, from 20 to 50, and the data generation rate R_{gen}, from 10 msg/s to 100 msg/s, are shown in Fig. 4.20.

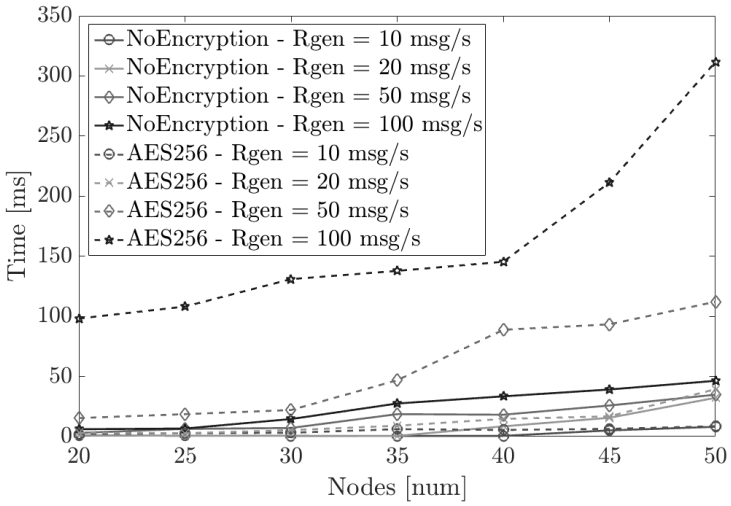


Figure 4.20: Average stream delay (dimension: [ms]) related to Graph framework processing block, showing per-node time, in the case of unsecured communication as well as the case of adoption of symmetric encryption.

Table 4.2: Average stream delay related to the adoption of asymmetric encryption solution (RSA) into the Graph Framework processing block.

Number of nodes	Stream delay (Rgen = 50 msg/s)	Stream delay (Rgen = 100 msg/s)
20	128.7453 ms	10890.7263 ms
25	156.909 ms	12962.2934 ms
30	2783.8599 ms	13841.4744 ms
35	10104.7272 ms	14048.8625 ms
40	11283.9916 ms	14515.0021 ms

The stream delay can be given using the following expression:

$$T_{processing_{freq}} = \frac{T_{out} - T_{in} - \sum_{k=1}^{N} GP_k}{N-1}$$

where: T_{out} is the instant (dimension: [ms]) at which parking data reach the last Application processing node; T_{in} indicates the instant (dimension: [ms]) in which normalized data comes to the first Core layer; and GP_k is the processing time (dimension: [ms]) of a Graph node $k \in \{1, ..., N\}$.

Moreover, in order to investigate the benefits and drawbacks of the adoption of other security solutions, different from symmetric encryption, an asymmetric cryptography version of the Graph processing nodes has been implemented, adopting RSA [145] algorithm with 512-bit key, which represent a public/private key cryptosystem. In Table 4.2, the performance results (in terms of stream delay) retrieved from a third evaluation scenario and obtained by replacing symmetric cryptosystem adoption with public/private RSA certificates provided to the Graph nodes by the Application Register module, are shown.

The obtained results shown in Table 4.2 highlight how the adoption of an asymmetric cryptosystem represents a bad choice for the Graph inter-node security. Asymmetric solutions might be adopted outside of the Graph nodes, when an external node is willing to become an operating entity of the Graph framework, challenging an authentication transaction with its signed certificate, that allows to verify its identity by the Application Register. Therefore, in the joining phase the asymmetric solutions could also be motivated by the evidence that time consumption is not the main

constraint of this step.

In order to better highlight this final analysis of the evaluation results, in Fig. 4.21 a logarithmic-scaled version of previously carried results is shown, evaluating the logarithm of the stream delay as a function of the number of nodes in the Graph: (i) evaluation without encryption, (ii) evaluation with symmetric encryption (AES256), and (iii) evaluation with asymmetric encryption (RSA512), considering only, for comparison purpose, two values of data generation rate R_{gen}: 50 msg/s and 100 msg/s.

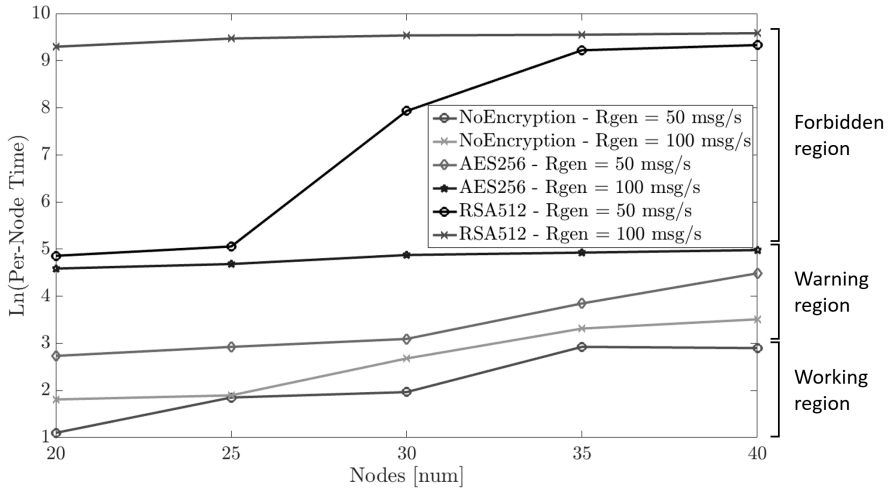


Figure 4.21: Logarithmic representation of the stream delay as a function of the number of nodes of the Graph, evaluated both without security mechanisms, as well as with cryptographic features.

The obtained results highlight that it seems possible to identify 3 main performance regions, in which the proposed Big Stream architecture could work:

- **"Working" region**, tentatively identified around the "NoEncryption" curves, on which the system has the benchmark results, and where processing do not introduce heavy delays;
- **"Warning" region**, limited around the "AES256" curves, in which delays in-

troduced by security adoptions degrade performances in a little way, while maintaining good Quality of Service (QoS);

- **"Forbidden" region**, around the "RSA512" curves, in which the system incurs high delays, that may cause crashes and drop services, invalidating QoS and any Service Level Agreements (SLAs) signed with data stream producers and consumers.

Algorithm 4 Pseudo-code of the operations done by the Application Register when an external process p_m asks to become a Graph listener.

```
NodeIdentityCertificate = {NodeInfo, INTERESTS = {interest_1, .... , interest_N}};
procedure MAIN()
    p_m = receive_join_request(NodeIdentityCertificate);
    authenticated = authenticate_node(p_m);
    if (authenticated is TRUE) then
        x = identify_graph_layer_for_node(p_m);
        for all (interest_k in INTERESTS) do
            Match = send_request_to_PMSV(layer = (x OR x+1), interest_k);
            if (Match is EMPTY) then
                REPLY_DENY(interest_k, p_m);
            else
                for all (g_m in Match) do
                    //topic = interest_k, owner = g_m, listener = p_m
                    allow = req_grant_to_owner_via_GRAN(interest_k, g_m, p_m);
                    if (allow is FALSE) then
                        REPLY_DENY(interest_k, p_m);
                    else
                        REPLY_SUCCESS(interest_k, p_m);
                    end if
                end for
            end if
        end for
    end if
end procedure
procedure REPLY_DENY(TOPIC, DESTINATION)
    send_DENY_response_to_dest(req_topic = topic);
end procedure
procedure REPLY_SUCCESS(TOPIC, DESTINATION)
    send_SUCCESS_response_to_dest(req_topic = topic, sec_params = {....});
end procedure
```

Chapter 5

Conclusions

This thesis has been focused on technologies and paradigms that allow to improve the powerful nature of the Internet of Things (IoT) paradigm at all communication layers, allowing a better and well organized integration of Smart Objects (SOs) interconnected in an Internet-like structure, all cooperating to collect the gigantic amount of data generated by SOs and provide services to end users.

At lower layers, in addition to the well-known wireless access technologies (like Wi-Fi, IEEE 802.15.4, and 3G/4G), Power Line Communication (PLC) has been considered as possible wired technology. In this context an experimentation study has been carried out, highlighting possible advantages and limitations. A possible architecture for IoT backhaul has been also considered, based on the Software-Defined Networking (SDN) paradigm. In this context, a solution for Traffic Engineering (TE) with Segment Routing (SR) has been presented, implemented and tested, and an extensive performance evaluation campaign has been done. Moreover, a new type of SR-related segments, denoted as *Direct-Link Segment Identifier (DL-SID)*, has been introduced, thus verifying its optimality compared with classical TE heuristics needed to allocate *hop-by-hop* paths.

Thereafter, related to the protocols that can be adopted in IoT scenarios, a performance evaluation of the low-power CoSIP protocol has been presented, showing that, on average, its usage allows to save approximately 33% of exchanged data in

the overall network. This significant gain, in the context of IoT application scenarios composed of (mostly) constrained devices, is important not only to reduce bandwidth usage and the amount of transmitted data, but also in terms of energy consumption and processing capabilities. In particular, constrained nodes can save CPU cycles, memory, and transmission periods during their lifetime.

The heterogeneity and diffusion of IoT-oriented SOs, together with the need to allow final users to interact, in an easier way, with these SOs, has facilitated the development of resource-oriented testbeds, moving the focus from the IoT to the concept of Web of Things (WoT). Following this Web model, a Web-based testbed, denoted as WoTT, has been detailed, exemplifying its novel architecture and its ability to face with several IoT challenges. Afterward, the integration of Wi-Fi devices into the WoTT, in conjunction with a novel approach to combine short-range IoT devices, denoted as Micro IoT, and the more recent paradigm of LPWANs, denoted as Macro IoT, have been introduced and motivated with experimental evaluations.

Later, the novel concept of Big Stream has been defined and introduced, motivating its adoption for the management of information coming from IoT scenarios, and highlighting its differences with respect to the Big Data paradigm. More in detail, the introduced listener-oriented approach can lead to several benefits, such as: (i) decreased latency: the push-based approach guarantees that no delays due to polling and batch processing are introduced; (ii) fine-grained self-configuration: listeners can dynamically "plug" to those that output data of interest; and (iii) optimal resource allocation: processing units that have no listeners can be switched off, while those with many listeners can be replicated, thus leading to cost-effectiveness from the Cloud service perspective. Then, a novel Cloud architecture for the management of Big Stream applications in IoT scenarios has been presented, describing: (i) the main requirements, in terms of reduced latency between the data creation instant and the instant at which processed data can be delivered to a consumer, and (ii) its main components: the Acquisition module, the Normalization module, the Graph framework, and the Application Register module. Then, the implementation of the overall system and its evaluation on a real-world smart parking dataset has been presented.

Finally, security issues and threats for an IoT-oriented Big Stream architecture

have been analyzed and addressed, implementing a secured version of each module of the platform and taking into account security enhancements and access control constraints. Then, by means of federated access policies and OAuth paradigm that, adapted to the specific needs of each module, complies with the proposed publish/subscribe platform and is expedient to carry out all security tasks, experimental evaluations have been performed, thus obtaining interesting results that further motivate the feasibility of this secured approach.

Bibliography

[1] J. Postel. Internet protocol. RFC 791, Sep 1981. Updated by RFCs 1349, 2474, 6864. URL: http://www.ietf.org/rfc/rfc791.txt.

[2] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. RFC 2460 (Draft Standard), December 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. URL: http://www.ietf.org/rfc/rfc2460.txt.

[3] IETF. The Internet Engineering Task Force. URL http://www.ietf.org.

[4] IEEE. The Institute of Electrical and Electronics Engineers. URL https://www.ieee.org.

[5] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, Jun 2014.

[6] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California, 2000. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, February 2013. URL: http://tools.ietf.org/html/rfc2616.

[8] Laura Belli, Simone Cirani, Andrea Gorrieri, and Marco Picone. A Novel Smart Object-Driven UI Generation Approach for Mobile Devices in the Internet of Things. In *Proceedings of the 1st International Workshop on Experiences with the Design and Implementation of Smart Objects*, SmartObjects '15, pages 1–6. ACM, 2015. doi:10.1145/2797044.2797046.

[9] European Community's 7th Framework Programme. (2012-2015). Internet of Things - Architecture (IoT - A). URL: http://www.iot-a.eu.

[10] European Community's 7th Framework Programme. (2008-2010). SENSEI Project. URL: http://www.ict-sensei.org.

[11] European Community's 7th Framework Programme. (2011-2014). CAL*IP*SO - Connect All IP-based Smart Objects. URL: http://www.ict-calipso.eu.

[12] P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 3921, RFC Editor, October 2004.

[13] MQTT. MQ Telemetry Transport. URL http://mqtt.org/.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. Sip: Session initiation protocol. RFC 3261, RFC Editor, June 2002. URL: http://www.rfc-editor.org/rfc/rfc3261.

[15] Simone Cirani, Marco Picone, and Luca Veltri. *CoSIP: A Constrained Session Initiation Protocol for the Internet of Things*, pages 13–24. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-45364-9_2.

[16] Simone Cirani, Marco Picone, and Luca Veltri. A session initiation protocol for the Internet of Things. *Scalable Computing: Practice and Experience*, 14(4):249–263, 2013. doi:10.12694/scpe.v14i4.931.

[17] G. Z. Papadopoulos, J. Beaudaux, A. Gallais, T. Noël, and G. Schreiner. Adding value to WSN simulation using the IoT-LAB experimental platform.

In *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 485–490, Oct 2013. doi:10.1109/WiMOB.2013.6673403.

[18] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R. Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, and Dennis Pfisterer. SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014. Special issue on Future Internet Testbeds – Part I. doi:10.1016/j.bjp.2013.12.020.

[19] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, RFC Editor, September 2007. URL: http://www.rfc-editor.org/rfc/rfc4944.txt.

[20] Bluetooth Low Energy (BLE). URL: https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy.

[21] L. Kriara, M. K. Marina, and A. Farshad. Characterization of 802.11n wireless LAN performance via testbed measurements and statistical analysis. In *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pages 158–166, June 2013. doi:10.1109/SAHCN.2013.6644974.

[22] Wireless Mesh Networking: ZigBee vs. DigiMesh. URL: https://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf.

[23] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O.Hersent. LoRaWAN Specification. URL: https://www.lora-alliance.org/portals/0/specs/LoRaWANSpecification1R0.pdf.

[24] Sigfox. URL: http://makers.sigfox.com.

[25] L. Li, H. Xiaoguang, C. Ke, and H. Ketai. The applications of WiFi-based Wireless Sensor Network in Internet of Things and Smart Grid. In *2011 6th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 789–793, June 2011. doi:10.1109/ICIEA.2011.5975693.

[26] Sinem Coleri Ergen. ZigBee/IEEE 802.15.4 Summary. http://pages.cs.wisc.edu/~suman/courses/707/papers/zigbee.pdf. Accessed: 2016-04-14.

[27] S. Tozlu. Feasibility of Wi-Fi enabled sensors for Internet of Things. In *2011 7th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 291–296, July 2011. doi:10.1109/IWCMC.2011.5982548.

[28] Benedikt Ostermaier, Matthias Kovatsch, and Silvia Santini. Connecting Things to the Web Using Programmable Low-power WiFi Modules. In *2nd International Workshop on Web of Things*, WoT'11, pages 2:1–2:6, New York, NY, USA, 2011. ACM. doi:10.1145/1993966.1993970.

[29] S. Tozlu, M. Senel, W. Mao, and A. Keshavarzian. Wi-Fi enabled sensors for internet of things: A practical approach. *IEEE Communications Magazine*, 50(6):134–143, June 2012. doi:10.1109/MCOM.2012.6211498.

[30] B. Ostermaier, F. Schlup, and K. Römer. WebPlug: A framework for the Web of Things. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 690–695, March 2010. doi:10.1109/PERCOMW.2010.5470522.

[31] IETF CoRE Working Group. https://tools.ietf.org/wg/core. Accessed: 2016-04-15.

[32] Industrial Internet of Things (IIoT). URL: http://internetofthingsagenda.techtarget.com/definition/Industrial-Internet-of-Things-IIoT.

[33] E. Ancillotti and R. Bruno and M. Conti. The role of communication systems in smart grids: Architectures, technical solutions and research challenges. *Computer Communications*, 36(17–18):1665–1697, 2013. doi:10.1016/j.comcom.2013.09.004.

[34] N. Bui and A. P. Castellani and P. Casari and M. Zorzi. The Internet of Energy: a Web-enabled Smart Grid System. *IEEE Network*, 26(4):39–45, Jul 2012. doi:10.1109/MNET.2012.6246751.

[35] ETSI. The European Telecommunications Standards Institute. URL http://www.etsi.org.

[36] W3C. The World Wide Web Consortium. URL https://www.w3.org.

[37] S. Galli, A. Scaglione, and Z. Wang. For the Grid and Through the Grid: The Role of Power Line Communications in the Smart Grid. *Proceedings of the IEEE*, 99(6):998–1027, Jun 2011. doi:10.1109/JPROC.2011.2109670.

[38] M. Hoch. Comparison of PLC G3 and PRIME. In *2011 IEEE International Symposium on Power Line Communications and Its Applications (ISPLC)*, pages 165–169, Apr 2011. doi:10.1109/ISPLC.2011.5764384.

[39] T. A. Papadopoulos and C. G. Kaloudas and A. I. Chrysochos and G. K. Papagiannis. Application of Narrowband Power-Line Communication in Medium-Voltage Smart Distribution Grids. *IEEE Transactions on Power Delivery*, 28(2):981–988, Apr 2013. doi:10.1109/TPWRD.2012.2230344.

[40] K. Razazian and J. Yazdani. CENELEC and Powerline Communication Specification in realization of Smart Grid Technology. In *2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe)*, pages 1–7, Dec 2011. doi:10.1109/ISGTEurope.2011.6162685.

[41] H. Kellerbauer and H. Hirsch. Simulation of Powerline Communication with OMNeT++ and INET-Framework. In *2011 IEEE International Symposium on Power Line Communications and Its Applications (ISPLC)*, pages 213–217, Apr 2011. doi:10.1109/ISPLC.2011.5764395.

[42] OMNeT++ - Discrete Event Simulator. URL: https://omnetpp.org.

[43] D. P. F. Möller and H. Vakilzadian. Ubiquitous Networks: Power Line Communication and Internet of Things in Smart Home Environments. In *IEEE International Conference on Electro/Information Technology*, pages 596–601, Jun 2014. doi:10.1109/EIT.2014.6871832.

[44] Q. Ou, Y. Zhen, X. Li, Y. Zhang, and L. Zeng. Application of Internet of Things in Smart Grid Power Transmission. In *2012 Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing (MUSIC)*, pages 96–100, Jun 2012. doi:10.1109/MUSIC.2012.24.

[45] Software-Defined Networking: The New Norm for Networks. URL: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[46] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar 2008. doi:10.1145/1355734.1355746.

[47] Clarence Filsfils, Stefano Previdi, Ahmed Bashandy, Bruno Decraene, Stephane Litkowski, Martin Horneffer, Rob Shakir, Jeff Tantsura, and Edward Crabbe. Segment Routing Architecture. Internet-Draft draft-ietf-spring-segment-routing, IETF Secretariat, July 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-spring-segment-routing.

[48] Katarina Stanoevska-Slabeva and Thomas Wozniak. *Grid and Cloud Computing: A Business Perspective on Technology and Applications*, pages 3–11. 2010. doi:10.1007/978-3-642-05193-7_1.

[49] Peter M. Mell and Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, 2011. doi:10.6028/NIST.SP.800-145.

[50] European Union's Seventh Framework Programme . OpenIoT - Open Source cloud solution for the Internet of Things. URL http://openiot.eu/, 2007. URL: http://openiot.eu/.

[51] European Community's 7th Framework . FIWARE Project. URL http://www.fi-ware.org/, 2011. URL: http://www.fi-ware.org/.

[52] OGF - Open Grid Forum. OCCI - Open Cloud Computing Interface. URL http://occi-wg.org/. URL: http://occi-wg.org/.

[53] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42, October 2012. URL: https://www.openstack.org.

[54] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures. *Computer*, 45(12):65–72, Dec 2012. doi:10.1109/MC.2012.76.

[55] Nir Kshetri. Privacy and security issues in cloud computing: The role of institutions and institutional evolution. *Telecommunications Policy*, 37(4):372–386, May 2013. doi:10.1016/j.telpol.2012.04.011.

[56] Mazhar Ali, Samee U. Khan, and Athanasios V. Vasilakos. Security in cloud computing: Opportunities and challenges. *Information Sciences*, 305:357–383, June 2015. doi:10.1016/j.ins.2015.01.025.

[57] Marcos A. P. Leandro, Tiago J. Nascimento, Daniel R. dos Santos, Carla M. Westphall, and Carlos B. Westphall. Multi-Tenancy Authorization System with Federated Identity for Cloud-Based Environments Using Shibboleth. pages 88–93, 2012.

[58] David Lake, Rodolfo Milito, Monique Morrow, and Rajesh Vargheese. Internet of Things: Architectural Framework for eHealth Security. *Journal of ICT Standardization*, 1(3):301–328, January 2014. doi:10.13052/jicts2245-800X.133.

[59] G. Suciu, A. Vulpe, S. Halunga, O. Fratu, G. Todoran, and V. Suciu. Smart Cities Built on Resilient Cloud Computing and Secure Internet of Things. In *2013 19th International Conference on Control Systems and Computer Science*, pages 513–518, May 2013. doi:10.1109/CSCS.2013.58.

[60] Parves Kamal, Saad Mustafiz, Faisal Md Abdur Rahman, and Raihan Taher. Evaluating the Efficiency and Effectiveness of a Federated SSO Environment Using Shibboleth. *Journal of Information Security*, 6(3):166–178, July 2015.

[61] M. Collina, G. E. Corazza, and A. Vanelli-Coralli. Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, pages 36–41, Sept 2012. doi:10.1109/PIMRC.2012.6362813.

[62] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, October 2012. URL: http://www.rfc-editor.org/rfc/rfc6749.txt.

[63] D. Lagutin, K. Visala, A. Zahemszky, T. Burbridge, and G. F. Marias. Roles and security in a publish/subscribe network architecture. In *2010 IEEE Symposium on Computers and Communications (ISCC)*, pages 68–74, June 2010. doi:10.1109/ISCC.2010.5546746.

[64] Chenxi Wang, A. Carzaniga, D. Evans, and A. L. Wolf. Security issues and requirements for Internet-scale publish-subscribe systems. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002. HICSS*, pages 3940–3947, Jan 2002. doi:10.1109/HICSS.2002.994531.

[65] C. Raiciu and D. S. Rosenblum. Enabling Confidentiality in Content-Based Publish/Subscribe Infrastructures. In *Securecomm and Workshops, 2006*, pages 1–11, Aug 2006. doi:10.1109/SECCOMW.2006.359552.

[66] P. Fremantle, B. Aziz, J. Kopecký, and P. Scott. Federated Identity and Access Management for the Internet of Things. In *2014 International Workshop on Secure Internet of Things (SIoT)*, pages 10–17, Sept 2014. doi:10.1109/SIoT.2014.8.

[67] Jean Bacon, David Evans, David M. Eyers, Matteo Migliavacca, Peter Pietzuch, and Brian Shand. *Enforcing End-to-End Application Security in the Cloud*, pages 293–312. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-16955-7_15.

[68] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, 2012. doi:10.1145/2342509.2342513.

[69] Andrew McAfee and Erik Brynjolfsson. Big Data: The Management Revolution. URL: https://hbr.org/2012/10/big-data-the-management-revolution.

[70] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[71] Cory Isaacson. *Software Pipelines and SOA: Releasing the Power of Multi-Core Processing*. Addison-Wesley Professional, 1 edition, 2009.

[72] Marcos D. Assunção, Rodrigo N. Calheiros, Silvia Bianchi, Marco A.S. Netto, and Rajkumar Buyya. Big Data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79–80:3–15, 2015. Special Issue on Scalable Systems for Big Data Management and Analytics. doi:10.1016/j.jpdc.2014.08.003.

[73] Apache. Storm. URL https://storm.incubator.apache.org/.

[74] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed Stream Computing Platform. In *2010 IEEE International Conference on Data Mining Workshops*, pages 170–177, Dec 2010. doi:10.1109/ICDMW.2010.172.

[75] S. Reiff-Marganiec, M. Tilly, and H. Janicke. Low-Latency Service Data Aggregation Using Policy Obligations. In *2014 IEEE International Conference on Web Services (ICWS)*, pages 526–533, June 2014. doi:10.1109/ICWS.2014.80.

[76] Laura Belli, Simone Cirani, Luca Davoli, Andrea Gorrieri, Mirko Mancin, Marco Picone, and Gianluigi Ferrari. Design and Deployment of an IoT Application-Oriented Testbed. *Computer*, 48(9):32–40, Sep 2015. doi:10.1109/MC.2015.253.

[77] European Committee for Electrotechnical Standardization (CENELEC). URL: http://www.cenelec.eu.

[78] Association of Radio Industries and Businesses (ARIB). URL: http://www.arib.or.jp.

[79] Federal Communications Commission (FCC). URL: https://www.fcc.gov.

[80] PRIME Alliance. URL: http://www.prime-alliance.org.

[81] HomePlug Alliance. URL: http://www.homeplug.org/tech-resources/resources.

[82] G3-PLC Alliance. URL: http://www.g3-plc.com.

[83] V. Oksman and J. Zhang. G.HNEM: the new ITU-T standard on narrowband PLC technology. *IEEE Communications Magazine*, 49(12):36–44, December 2011. doi:10.1109/MCOM.2011.6094004.

[84] IEEE P1901.2 Standard. URL: http://grouper.ieee.org/groups/1901/2.

[85] TMDSPLCKIT-V3 - C2000 Power Line Modem Developer's Kit. URL: http://www.ti.com/tool/TMDSPLCKIT-V3".

[86] Java Simple Serial Connector (JSSC). URL: https://code.google.com/archive/p/java-simple-serial-connector.

[87] TMDSDC3359 - Data Concentrator Evaluation Module. URL: http://www.ti.com/tool/tmdsdc3359.

[88] M-K. Shin, Y. Hong, and C.Y. Ahn. A Software-Defined Approach for End-to-end IoT Networking. URL: https://www.ietf.org/proceedings/91/slides/slides-91-sdnrg-3.pdf.

[89] SDN based Traffic Engineering and Segment Routing. URL: https://github.com/netgroup/SDN-TE-SR.

[90] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, October 2011. doi:10.1109/JSAC.2011.111002.

[91] Stefano Salsano, Pier Luigi Ventre, Francesco Lombardo, Giuseppe Siracusano, Matteo Gerola, Elio Salvadori, Michele Santuari, Mauro Campanella, and Luca Prete. OSHI - Open Source Hybrid IP/SDN networking and Mantoo - a set of management tools for controlling SDN/NFV experiments. *CoRR*, abs/1505.03579, 2015. URL: http://arxiv.org/abs/1505.03579.

[92] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, 2010. doi:10.1145/1868447.1868466.

[93] Peter Psenak, Stefano Previdi, Clarence Filsfils, Hannes Gredler, Rob Shakir, Wim Henderickx, and Jeff Tantsura. OSPF Extensions for Segment Routing. Internet-Draft draft-ietf-ospf-segment-routing-extensions, IETF Secretariat, July 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-ospf-segment-routing-extensions.

[94] Stefano Previdi, Clarence Filsfils, Ahmed Bashandy, Hannes Gredler, Stephane Litkowski, and Bruno Decraene. IS-IS Extensions for Segment Routing. Internet-Draft draft-ietf-isis-segment-routing-extensions, IETF Secretariat, October 2016. URL: http://www.ietf.org/internet-drafts/draft-ietf-isis-segment-routing-extensions.

[95] Stefano Salsano, Luca Veltri, Luca Davoli, Pier Luigi Ventre, and Giuseppe Siracusano. PMSR – Poor Man's Segment Routing, a minimalistic approach to Segment Routing and a Traffic Engineering use case. *CoRR*, abs/1512.05281, Dec 2015. URL: http://arxiv.org/abs/1512.05281.

[96] Stefano Salsano, Luca Veltri, Luca Davoli, Pier Luigi Ventre, and Giuseppe Siracusano. PMSR – Poor Man's Segment Routing, a minimalistic approach to Segment Routing and a Traffic Engineering use case. In *2016 IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 598–604, Apr 2016. doi:10.1109/NOMS.2016.7502864.

[97] Luca Davoli, Luca Veltri, Pier Luigi Ventre, Giuseppe Siracusano, and Stefano Salsano. Traffic Engineering with Segment Routing: SDN-based Architectural Design and Open Source Implementation. *CoRR*, abs/1506.05941, Dec 2015. URL: http://arxiv.org/abs/1506.05941.

[98] Luca Davoli, Luca Veltri, Pier Luigi Ventre, Giuseppe Siracusano, and Stefano Salsano. Traffic Engineering with Segment Routing: SDN-based Architectural Design and Open Source Implementation. In *2015 Fourth European Workshop on Software Defined Networks (EWSDN)*, pages 111–112. IEEE, Sep 2015. doi:10.1109/EWSDN.2015.73.

[99] A. B. Roach. Session initiation protocol (sip)-specific event notification. RFC 3265, RFC Editor, June 2002.

[100] Klaus Hartke. Observing resources in coap. Internet-Draft draft-ietf-core-observe, IETF Secretariat, December 2014. URL: http://tools.ietf.org/id/draft-ietf-core-observe.

[101] Zach Shelby, Klaus Hartke, and Carsten Bormann. Core resource directory. Internet-Draft draft-ietf-core-resource-directory, IETF Secretariat, July 2016. URL: https://tools.ietf.org/html/draft-ietf-core-resource-directory.

[102] R. Price, C. Bormann, J. Christoffersson, H. Hannu, Z. Liu, and J. Rosenberg. Signaling compression (sigcomp). RFC 3320, RFC Editor, January 2003.

[103] Michele Amoretti. A Survey of Peer- to -Peer Overlay Schemes: Effectiveness, Efficiency and Security. *Recent Patents on Computer Science*, 2(3):195–213, 2009. doi:10.2174/1874479610902030195.

[104] Simone Cirani, Luca Davoli, Maarco Picone, and Luca Veltri. Performance Evaluation of a SIP-based Constrained Peer-to-Peer Overlay. In *2014 International Conference on High Performance Computing Simulation (HPCS)*, pages 432–435. IEEE, Jul 2014. doi:10.1109/HPCSim.2014.6903717.

[105] Zach Shelby, Klaus Hartke, and Carsten Bormann. Constrained application protocol (coap). Internet-Draft draft-ietf-core-coap-18, IETF Secretariat, June 2013. URL: http://www.ietf.org/internet-drafts/draft-ietf-core-coap-18.txt.

[106] Haipeng Jin and A. C. Mahendran. Using SigComp to compress SIP/SDP messages. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, volume 5, pages 3107–3111, May 2005. doi:10.1109/ICC.2005.1494974.

[107] Guo fang Dong, Zhiquan Bai, Zi fu Fan, and Ping Yang. Performance enhancement of signaling compression in IP Multimedia Subsystem. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, pages 483–487, Sept 2009. doi:10.1109/ISCIT.2009.5341200.

[108] Hongzhou Li Derong Du, Jianming Liu and Zhiyong Peng. Research of SIP Compression Based on SigComp. In *Research Journal of Applied Sciences, Engineering and Technology*, pages 5320–5324, May 2013.

[109] mjSIP Project. URL: http://mjsip.org.

[110] mjCoAP. URL http://www.mjcoap.org/.

[111] CoSIP Project. URL: http://cosip.org.

[112] S. Cirani and L. Veltri. Implementation of a framework for a DHT-based Distributed Location Service. In *Software, Telecommunications and Computer Networks, 2008. SoftCOM 2008. 16th International Conference on*, pages 279–283, Sept 2008. doi:10.1109/SOFTCOM.2008.4669495.

[113] Web of Things Testbed (WoTT). URL: http://iotlab.unipr.it/wott.

[114] C. Bormann, M. Ersue, and A. Keranen. Terminology for Constrained-Node Networks. RFC 7228, RFC Editor, May 2014. URL: http://www.rfc-editor.org/rfc/rfc7228.txt.

[115] Geoff Mulligan. The 6LoWPAN Architecture. In *Proceedings of the 4th Workshop on Embedded Networked Sensors*, EmNets '07, pages 78–82, 2007. doi:10.1145/1278972.1278992.

[116] Node.js. URL: https://nodejs.org.

[117] Simone Cirani, Gianluigi Ferrari, Nicola Iotti, and Marco Picone. The IoT Hub: A Fog Node for Seamless Management of Heterogeneous Connected Smart Objects. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*, pages 1–6, June 2015. doi:10.1109/SECONW.2015.7328145.

[118] Cisco Connected Mobile Experiences (CMX). URL: http://www.cisco.com/c/en/us/solutions/enterprise-networks/connected-mobile-experiences.

[119] Simone Cirani and Marco Picone. Wearable Computing for the Internet of Things. *IT Professional*, 17(5):35–41, Sept 2015. doi:10.1109/MITP.2015.89.

[120] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*, 12(9):11734, 2012.

[121] Eddystone. URL: https://github.com/google/eddystone.

[122] Luca Davoli, Laura Belli, Antonio Cilfone, and Gianluigi Ferrari. Integration of Wi-Fi mobile nodes in a Web of Things Testbed. *ICT Express*, 2(3):95–99, 2016. Special Issue on ICT Convergence in the Internet of Things (IoT). doi:10.1016/j.icte.2016.07.001.

[123] Simone Cirani, Luca Davoli, Gianluigi Ferrari, Remy Léone, Paolo Medagliani, Marco Picone, and Luca Veltri. A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things. *IEEE Internet of Things Journal*, 1(5):508–521, Oct 2014. doi:10.1109/JIOT.2014.2358296.

[124] JmDNS. http://jmdns.sourceforge.net. Accessed: 2016-04-16.

[125] Jetty - Servlet Engine and HTTP Server. URL: http://www.eclipse.org/jetty.

[126] Laura Belli, Simone Cirani, Luca Davoli, Lorenzo Melegari, Màrius Mónton, and Marco Picone. An Open-Source Cloud Architecture for Big Stream IoT Applications. In Ivana Podnar Žarko, Krešimir Pripužić, and Martin Serrano, editors, *Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia, September 18, 2014, Invited Papers*, pages 73–88. Springer International Publishing, Sep 2015. doi:10.1007/978-3-319-16546-2_7.

[127] Laura Belli, Simone Cirani, Luca Davoli, Gianluigi Ferrari, Lorenzo Melegari, Màrius Montón, and Marco Picone. A Scalable Big Stream Cloud Architecture for the Internet of Things. *International Journal of Systems and Service-Oriented Engineering (IJSSOE)*, 5(4):26–53, 2015. doi:10.4018/IJSSOE.2015100102.

[128] Laura Belli, Simone Cirani, Luca Davoli, Gianluigi Ferrari, Lorenzo Melegari, and Marco Picone. Applying Security to a Big Stream Cloud Architecture for the Internet of Things. *International Journal of Distributed Systems and Technologies (IJDST)*, 7(1):37–58, 2016. doi:10.4018/IJDST.2016010103.

[129] SimpleLink Wi-Fi CC3200 LaunchPad. http://www.ti.com/tool/cc3200-launchxl. Accessed: 2016-04-15.

[130] Z. Shelby. Constrained RESTful Environments (CoRE) Link Format. RFC 6690, Aug 2012.

[131] Agnar Aamodt and Mads Nygård. Different Roles and Mutual Dependencies of Data, Information, and Knowledge&Mdash;an AI Perspective on Their Integration. *Data Knowl. Eng.*, 16(3):191–222, Oct 1995. doi:10.1016/0169-023X(95)00017-M.

[132] Rabbitmq. URL http://www.rabbitmq.com/.

[133] Steve Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6):87–89, November 2006. URL: http://dx.doi.org/10.1109/MIC.2006.116, doi:10.1109/MIC.2006.116.

[134] NGINX. The High-performance Web Server and Reverse Proxy. URL http://wiki.nginx.org/Main.

[135] R. T. Fielding and G. Kaiser. The apache http server project. *IEEE Internet Computing*, 1(4):88–90, Jul 1997. doi:10.1109/4236.612229.

[136] Apache ActiveMQ. URL http://activemq.apache.org/.

[137] Mosquitto. An Open Source MQTT Broker. URL http://mosquitto.org/.

[138] MySQL. URL http://www.mysql.com/.

[139] Worldsensing. URL http://www.worldsensing.com/.

[140] I. Vilajosana, J. Llosa, B. Martinez, M. Domingo-Prieto, A. Angles, and X. Vilajosana. Bootstrapping smart cities through a self-sustainable model based on big data flows. *IEEE Communications Magazine*, 51(6):128–134, June 2013. doi:10.1109/MCOM.2013.6525605.

[141] IBM Emerging Technology. Node-RED. URL: http://nodered.org.

[142] Simone Cirani, Marco Picone, Pietro Gonizzi, Luca Veltri, and Gianluigi Ferrari. IoT-OAS: An OAuth-Based Authorization Service Architecture for Secure Services in IoT Scenarios. *IEEE Sensors Journal*, 15(2):1224–1234, Feb 2015. doi:10.1109/JSEN.2014.2361406.

[143] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009.

[144] Joan Daemen and Vincent Vincent Rijmen. *The Advanced Encryption Standard Process*, pages 1–8. Springer Berlin Heidelberg, 2002. doi:10.1007/978-3-662-04722-4_1.

[145] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM*, 21(2):120–126, Feb 1978. doi:10.1145/359340.359342.

Acknowledgments

Ci sono davvero molte persone che desidero ringraziare, senza le quali sarebbe stato molto difficile arrivare a questo grande traguardo. Primo tra tutti, desidero rivolgere un grato e sincero ringraziamento al mio relatore, Dott. Luca Veltri, che mi ha dato la possibilità di intraprendere questo percorso, guidandomi ed incoraggiandomi con competenza e pazienza nel corso degli ultimi anni. Un sentito ringraziamento lo devo al Prof. Gianluigi Ferrari, con cui ho avuto il piacere di collaborare e di scambiare suggerimenti utili ed interessanti per l'attività di ricerca durante il mio dottorato. Un grande e sentito ringraziamento lo devo a Simone Cirani e a Marco Picone, che hanno rappresentato una guida fondamentale per i primi anni del mio dottorato insegnandomi davvero molto e non mancando mai di consigli. Un grazie particolare lo devo rivolgere a Laura Belli (con cui ho avuto il piacere e la fortuna di dividere l'ufficio nell'ultimo periodo del dottorato), ad Andrea Gorrieri ed a Federico Parisi, con i quali ho condiviso la maggior parte del dottorato: grazie soprattutto per la pazienza, la competenza e la disponibilità. Grazie anche a tutto l'IoTLab (Marco Martalò, Nicolò Strozzi e Antonio Cilfone) ed a tutto il MultimediaLab (Carlo Tripodi e Davide Alinovi) per il fondamentale aiuto tecnico, per tutto ciò che mi avete insegnato e trasmesso, per i consigli e, non ultime, per tutte le risate che hanno accompagnato e reso divertente ogni pausa pranzo! Un sentito ringraziamento lo devo sicuramente al Consortium GARR, che ha finanziato parte dell'attività di ricerca e che mi ha permesso di approfondire argomenti attuali ed estremamente interessanti. Ultima, ma non per importanza, desidero ringraziare la mia famiglia, sempre pronta a sostenermi ed incoraggiarmi in qualsiasi momento della mia vita, e per aver sempre creduto in me.