Designing multiscale hybrid platform for testing and evaluationg IoT systems

Osama Abu Oun

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

Conception D'une Plate-Forme Multi-Échelle Hybride pour Évaluer Les Performances de Systèmes Orientés Internet des Objets

OSAMA ABU OUN

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

N° 2015059

THÈSE présentée par

OSAMA ABU OUN

pour obtenir le

Grade de Docteur de
l'Université de Franche-Comté

Spécialité : **Informatique**

Conception D'une Plate-Forme Multi-Échelle Hybride pour Évaluer Les Performances de Systèmes Orientés Internet des Objets

Soutenue publiquement le 09 Octobre 2015 devant le Jury composé de :

JULIEN BOURGEOIS	Président	Professeur à l'Université de Franche-Comté
THIERRY DIVOUX	Rapporteur	Professeur à l'Université de Lorraine
FRÉDÉRIC WEIS	Rapporteur	Maître de Conférences HDR à l'Université de Rennes
BENOÎT HILT	Examinateur	Maître de Conférences à l'Université de Haute-Alsace
FRANÇOIS SPIES	Directeur de thèse	Professeur à l'Université de Franche-Comté
CHRISTELLE BLOCH	Co-Directeur de thèse	Maître de conférences à l'Université de Franche-Comté

REMERCIEMENTS

Le bon déroulement et la rédaction de cette thèse n'aurait pu avoir lieu sans le soutien de l'ensemble de l'équipe du département DISC au Laboratoire Femto-ST à Montbéliard, je tiens à les en remercier.

Je remercie particulièrement:

- Monsieur **François SPIES**, Professeur des Universités, Université de Franche-Comté
- Madame **Christelle BLOCH**, Maître de conférences à l'Université de Franche-Comté

qui m'ont accueilli au sein de l'équipe de recherche de Montbéliard et suivi tout au long de ma thèse.

Je tiens à remercier:

- Monsieur **Thierry DIVOUX**, Professeur des Universités, Université de Lorraine.
- Monsieur **Frédéric WEIS**, Maître de Conférences à l'Université de Rennes.

pour l'intérêt qu'ils on manifesté pour mon travail et pour avoir accepté d'être les rapporteurs.

- Monsieur **Benoît HILT**, Maître de Conférences à l'Université de Haute-Alsace.
- Monsieur **Julien BOURGEOIS**, Professeur à l'Université de Franche-Comté.

pour avoir accepté d'être membres de mon jury.

Enfin, je remercie ma famille et mes amis pour leurs encouragements et leur confiance.

À ma mère, Rihab, qu'elle repose en paix

Contents

List of Abbreviations

6LoWPAN . . . . . IPv6 over Low Power WPAN
A-MSDU . . . . . Aggregate MAC Service Data Unit
AP . . . . . Access Point
ARM . . . . . Architecture Reference Model
BLE . . . . . Bluetooth Low Energy
BSS . . . . . Basic Service Set
BSSID . . . . . Basic Service Set Identifier
CEMAT . . . . . Cloud Environment for Mobile Application Testing
CLI . . . . . Command User Interface
CSMA/CA . . . . . Carrier Sense Multiple Access/Collision Avoidance
CoAP . . . . . Constrained Application Protocol
CORBA . . . . . Common Object Request Broker Architecture
CoRE . . . . . Constrained RESTful Environments
DAD . . . . . Duplicate Address Detection
DCE . . . . . Distributed Computing Environment
DCF . . . . . Distributed Coordination Function
DCOM . . . . . Distributed Component Object Model
DoD . . . . . U.S. Department of Defense
DSA . . . . . Digital Signature Algorithm
DSSS . . . . . Direct sequence spread spectrum
ESS . . . . . Extended Service Set
ESSID . . . . . Extended Service Set Identifier
FG . . . . . Functionality Group
FHSS . . . . . Frequency hopping spread spectrum
FOSS . . . . . Free and Open Source Software
JRE . . . . . Java Runtime Environment
GNSS . . . . . Global Navigation Satellite System
GPS . . . . . Global Positioning System
GUI . . . . . Graphical User Interface
HART . . . . . Highway Addressable Remote Transducer
HTML . . . . . HyperText Markup Language
HTTP . . . . . HyperText Transfer Protocol

HOTP . . . . . Hmac-based One-Time Password algorithm
IBSS . . . . . independent Basic Service Set
IC . . . . . Integrated circuit
IE . . . . . Information Element
IEEE . . . . . Institute of Electrical and Electronics Engineers
IETF . . . . . Internet Engineering Task Force
IoT . . . . . Internet of Things
IoT-A . . . . . Internet of Things - Architecture
IPS . . . . . Indoor Positioning Systems
ISM . . . . . Industrial, Scientific, and Medical
ISO . . . . . International Organization for Standardization
ITU . . . . . International Telecommunication Union
ITS . . . . . Intelligent Transportation Systems
FFD . . . . . Full-Function Device
GO . . . . . Group Owner
LAN . . . . . Local Area Network
LLN . . . . . Low Power and Lossy Networks
LR-WPAN . . . . . Low-Rate Wireless Personal Area Network
LW-Service . . . . . LightWeight Service
LWS . . . . . LightWeight Server
LWSB . . . . . LightWeight Service Beneficiary
LWSH . . . . . LightWeight Services Helper
MAC . . . . . Medium Access Control
MANET . . . . . Mobile ad hoc network
MBT . . . . . Model-Based Testing
MMPDU . . . . . MAC Management Protocol Data Unit
MSDU . . . . . MAC Service Data Unit
NFC . . . . . Near Field communication
NTP . . . . . Network Time Protocol
OMG . . . . . Object Management Group
ONC . . . . . Open Network Connectivity
OS . . . . . Operating System
OSF . . . . . Open Software Foundation
OSI . . . . . Open Systems Interconnection
OTP . . . . . One-Time Password
P2P . . . . . Peer-to-Peer
PC . . . . . Personal Computer

PCF . . . . . Point Coordination Function
PDA . . . . . Personal Digital Assistants
PDU . . . . . Packet Data Unit
PHY . . . . . Physical Layer
QoS . . . . . Quality of Service
REST . . . . . Representational State Transfer
RF . . . . . Radio Frequency
RFD . . . . . Reduced-Function Device
RFID . . . . . Radio Frequency Identification
ROLL . . . . . Routing Over Low Power and Lossy Networks
RPC . . . . . Remote Procedure Call
RPL . . . . . IPv6 Routing Protocol for Low-Power and Lossy Networks
RTLS . . . . . Real-Time Locating Systems
SDK . . . . . Software Development Kit
SGML . . . . . Standard Generalized Markup Language
SMS . . . . . Short Message Service
SOAP . . . . . Simple Object Access Protocol
SOTA . . . . . State Of The Art
SSID . . . . . Server Set Identifier
SUT . . . . . System Under Test
SVG . . . . . Scalable Vector Graphics
TC . . . . . Traffic Control
TCP . . . . . Transmission Control Protocol
Telnet . . . . . Telecommunications Network
TOTP . . . . . Time-based One-Time Password algorithm
UDP . . . . . User Datagram Protocol
UI . . . . . User Interface
UID . . . . . User Identifier
URI . . . . . Uniform Resource Identifier
URL . . . . . Uniform Resource Locator
USDL . . . . . Unified Service Description Language
WAP . . . . . Wireless Application Protocol
Wi-Fi . . . . . Wireless Fidelity
WLAN . . . . . Wireless LAN
WM . . . . . Wireless Medium
WoT . . . . . Web of Things
WPAN . . . . . Wireless Personal Area Network

WSDL . . . . . Web Service Definition Language
WWW . . . . . World Wide Web
XML . . . . . Extensible Markup Language

I

Principles and Fundamentals

1

INTRODUCTION

Internet is the physical layer or network made up of switches, routers, and other equipment. Since the first design during the ARPANET era, it does the same thing that it was designed to. It transports information from one point to another quickly, reliably, and securely. The technologies used to achieve this goal have been evolved, but the concept has been on a steady path of development and improvement. On the other hand, the range of applications that use Internet as an infrastructure, varies from the inter-linked hypertext documents and the World Wide Web (or Web), email and peer-to-peer networks, [2,3,N]-Tiers applications to file sharing and telephony. The technologies used in the Internet have been evolved to connect different type of devices and terminals, such as mobile phones and Tablets. However, all these devices are part of the virtual world of Internet where the main components are the computers. Since its existence, there were two separated worlds, the virtual world represented by the Internet and the real world represented by humans.
The Internet of Things (IoT) represents a vision in which the Internet extends into the real world embracing everyday objects (Figure 1.1). Physical items are no longer disconnected from the virtual world, but can be controlled remotely and can act as physical access points to Internet services [Mattern et al., 2010]. Within the IoT literally anything can be connected to a computer network, via an IP address like the one in your computer, and allowed to transfer data without the need for human-to-human or human-to-computer interaction. A "Thing" could be a car, an animal with a bio-chip transponder, a fitness band on your wrist, a refrigerator, the jet engine of an airplane, or your cat's collar. These objects, in addition to billions of others, could become connected to the Internet with the help of sensors and actuators [Morien, 2015]. The Internet of Things represents the first real change in the concept of the Internet.
According to Cisco Internet Business Solutions Group (IBSG), IoT is simply the point in

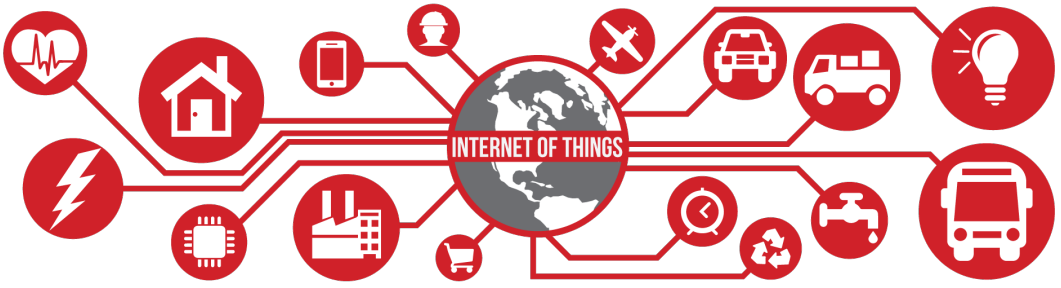

Figure 1.1: Internet of Things [Kirk, 2015]©

time when more "things or objects" were connected to the Internet than people. According to the statistics, the world already has arrived to this point between 2008 and 2009. Cisco projects that by 2020 there will be nearly 50 billion devices on the IoT [Evans, 2011].
IoT presents a grand scale of opportunities, at same time it presents challenges and different types of threats than those experienced on nowadays Internet. The challenges vary from software design, application testing, data storage, privacy and security, communication technologies to whole dimension of the IOT architecture.
One of the main points to be considered here that It isn't about new technologies being developed; it's about existing things and objects being integrated, being configured, and co-existing. In "Things", connectivity isn't built into them by design, but it should be added after the fact.
Note: In this thesis, our study focus on Internet and IoT, many of the technologies and the applications mentioned here can be used in Intranet, Extranet or and other types of networks. But in this document we will not refer to the differences between the different types of networks.

1.1/ OBJECTIVES OF THE THESIS

In the scope of this work, we address two sub-domains of the domain of IoT. Firstly, we provide a standard for a testbed for testing IoT systems, protocols and applications in a cloud. Secondly, we present our extension to the protocol IEEE 802.11 that enables exchanging data without connection. The second contribution is an example of the protocols and the systems that might be tested and evaluated using the testbed described in the first contribution. In the following, we introduce briefly the objectives of these two researches.

1.1.1/ INTERNET OF THINGS AS A SERVICE (IOTAAS)

In IoT, every object (thing) is going to be connected with several other objects. These objects could be computers, mobile devices, servers, sensors, actuators, cars, etc. Testing and evaluating applications and systems of these objects will interfere with developers' tasks. Different objects would have different systems. Even for objects which are of different types, but which perform similar tasks and use the same protocol, they would have different implementations. If one of these implementations behave differently in certain situation/case, applications might not work properly. Developers and testers should be able to test their applications in an appropriate environment equipped with the main types of objects which could communicate with their applications.
A similar problem was raised in the field of mobile applications. Developers and testers have to test their applications on several mobile devices. Since there are thousands of mobile devices in the market, even for the big enterprises, preparing a laboratory with all these mobile devices is a waste of money, time and resources. That led to the appearance of enterprises to provide Testing-As-A-Service (TaaS) and Mobile-Testing-As-A-Service (mTaaS) clouds where developers and users can choose between several models such as: pay-per-use, pay-as-you-go, etc. Mobile devices represent only one type of objects that could be connected to IoT, the field of mobile applications is a very small example. It is important to notice here that even for enterprises which offer mobile applications test-

ing, there is no standard architecture to build the testing cloud. Each enterprise uses its own architecture which means that a developer can't use resources from several enterprises at the same time to test certain scenario.
In IoT, There are many sub-domains such as: smart home, smart city, smart campus, etc. Each sub-domain is adopting its own methods to build the infrastructure and to test the applications. It is expected to have overlaps between these sub-domains.
The objectives of this research is to provide a standard architecture to test and evaluate IoT applications. Internet of Things-As-A-Service (IoTaaS) is an abstract of a component-based solution. IoTaaS is generic so it could be adopted by most of IoT sub-domains. A generic standard is a must because there are many unknowns in IoT. Flexibility has been taken into consideration so components could be added to add more functionalities without affecting the older ones. IoTaaS is scalable, which means that it fits a small environment and a large distributed one. Several IoTaaS environments can form one IoTaaS so the user can use resources from all of them at the same time.

1.1.2/ Connectionless Data Exchanges (COLDE)

Data is the main interest and the main target of Internet and IoT. The data exchanged on the Internet nowadays could be categorized into: public data and private data. This data is exchanged as services. Data is exchanged using networks and there are two main categories of networks, wired networks and wireless networks. Networks can't distinguish public data from private data. Wireless network is one of the most important enablers for IoT, that motivated researchers to present new ideas and to design new protocols for wireless networks. In IoT, several sub-domains are built mainly on the idea of public data, such as: Intelligent Transportation System (ITS), commercial advertising, emergency services, etc. Unfortunately, all proposed protocols don't provide a method to enable users and things to exchange public data differently from private data.
The objective of this research is to answer two questions. The first question is: how can we exchange public data without being associated to any Wi-Fi network ?. The second question is: what is the data that will be exchanged ? To answer the first question, we present *Connectionless Data Exchange (COLDE)* which is our IEEE 802.11 extension which enables users and things to exchange public data without being associated to any Wi-Fi network. The extension is compliant with IEEE 802.11 protocol and all its extension. The extension enables users to exchange public data with several Wi-Fi network at the same time. *LightWeight Services* is our design to answer the second question.

1.2/ Plan of the thesis

The rest of this thesis is organized as follows. In chapter 2, we describe the current state of the art in IoT testing architectures and in connectionless methods. This chapter also gives a brief survey of the communication methods mainly used in three Wi-Fi-based applications: broadcasting, indoor positioning and emergency evacuation. Indeed, we chose these three categories of applications to better validate and illustrate our contributions within experimental test campaigns. This choice was motivated by the fact that such types of mobile applications are among the most known and the most used services, either for their own interest or as sub-components of more complete innovative services. In chapter 3, we present the methods and techniques which are used for evaluation and

testing applications.
Part II explains our first contribution in details. Chapter 4 presents the design of our system for Internet of Things As-A-Service (IoTaaS). Chapter 5 describes Cloud Environment for Mobile Application Testing (CEMAT) which is our pilot implementation of IoTaaS.
Part III presents out second contribution. Chapters 6 and 7 introduce our extension for Connectionless Data Exchange and its implementation in the embedded systems, in addition to the applications which were tested using the extension.
Part IV concludes our researches. Chapter 8 presents the conclusion of IoTaaS contribution and the future work. In chapter 9, we discuss the results of COLDE and future prospects.
The annexes describe the fundamentals of the traditional Internet and the principles of the IoT. Annex A presents the evolution of network-based application and the methods used for data exchange. Annex B introduces a study about the IoT applications, the protocols that have been developed by IETF to address the different issues in IoT, in addition to IoT-A (Internet-of-Things Architecture) that was designed by the European Lighthouse Integrated Project.

2

State of the Art

The concept of the IoT was referred to by Mark Weiser in his 1991 paper, *"The Computer for the 21st Century"*. Mark Weiser often referred to as the father of *ubiquitous computing*, in which he coined the term in 1988. Kevin Ashton first coined the term Internet of Things in 1999. International Telecommunication Union (ITU) has adopted the following working definition of the Internet of Things (As of June 2012): A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving inter-operable information and communication technologies [ITU, 2012]. At the end of the 80s and beginning of the 90s, two concepts have started their evolution. Both of them were trying to bridge the gap between the virtual and the physical worlds, but at same time they are roughly on two opposite sides.
The first concept was *Ubiquitous Computing*, in which computing is made to appear everywhere and anywhere and technology recedes into the background of our lives, it is also known as "calm computing" and "calm technology" (two terms were [2]coined in 1995 by PARC Researchers Mark Weiser and John Seely Brown). The technology required for ubiquitous computing comes in three parts: cheap, low-power computers that include equally convenient displays, a network that ties them all together, and software systems implementing ubiquitous applications [Weiser, 1991].
The second concept was *Virtual Reality*, it is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence [Bryson, 2013].
The difference between the two concepts is that virtual reality puts people inside a computer-generated world, ubiquitous computing forces the computer to live out here in the world with people [Weiser, 2015].
The field of IoT covers wide range of domains, such as embedded systems, ubiquitous computing, augmented reality, communication technologies, semantic interoperability, operating platforms and security, identification techniques, software engineering, etc. A roadmap of key developments in IoT research in the context of pervasive applications is shown in (Figure 2.1), which includes the technology drivers and key application outcomes expected in the next decade [Gubbi et al., 2013] [Sundmaeker et al., 2010].
The state of the art discusses two domains. First domain covers the researches which have been conducted for testing applications. The second domain presents the researches about exchanging data without establishing a connection between a Wi-Fi client and an access point. The following Wi-Fi-based applications are discussed in detail: Broadcasting, Indoor positioning and localization, and Emergency evacuation. Broadcasting and indoor positioning are essential techniques for providing most of the public services. Emergency evacuation is an example of applications that should be provided as

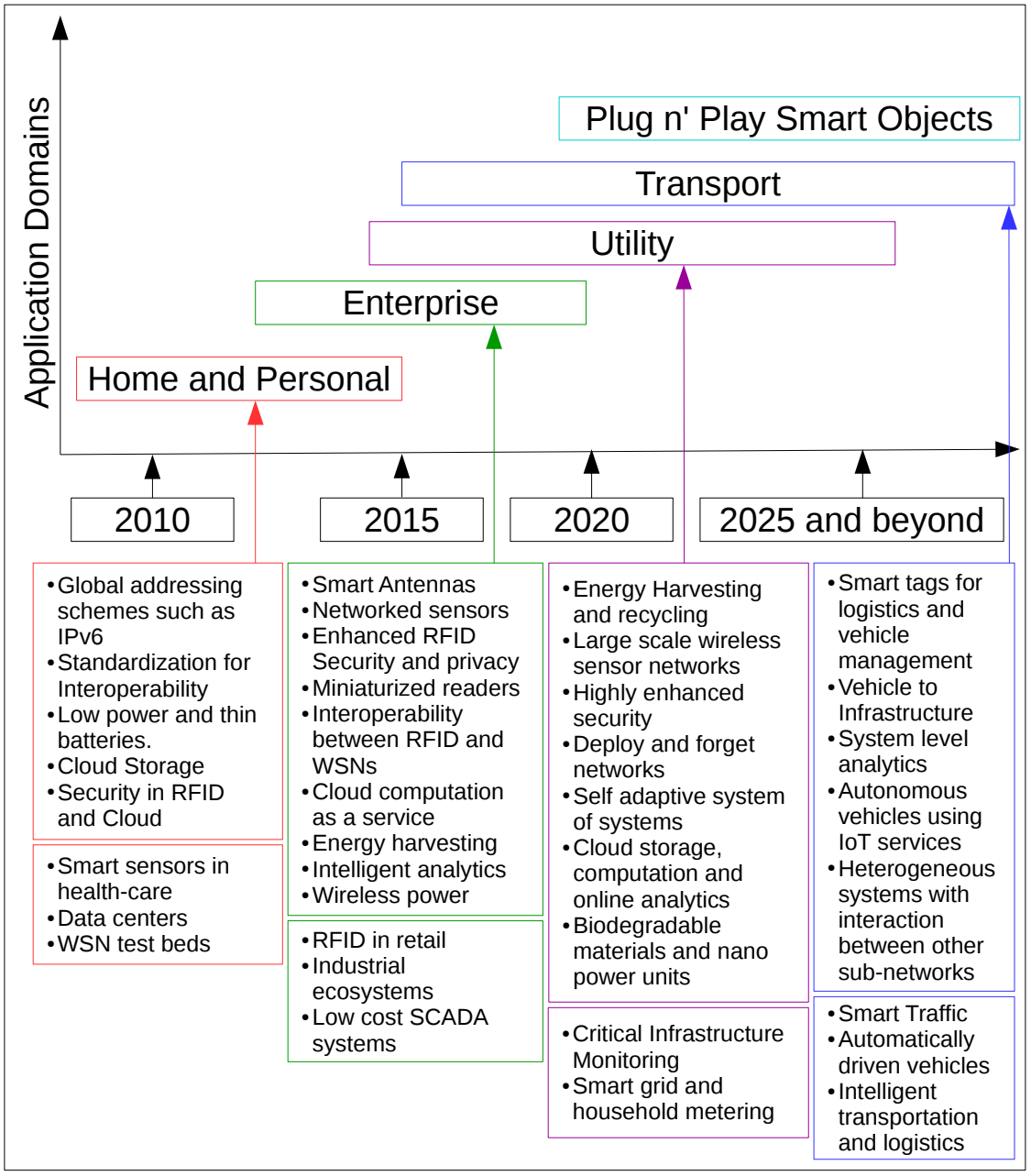

Figure 2.1: Roadmap of key technological developments in the context of IoT application domains envisioned

a public service, and it benefits from broadcasting and localization services. The state of the art reviews the techniques used in these applications and the ability to provide them to clients as public services without establishing a connection.

This chapter is organized as follows. Section 2.1 presents the related works about testing applications in Internet and IoT environments. Section 2.2 discusses the researches that have been conducted to enable using multiple Wi-Fi networks simultaneously. Section 2.3 reviews the main methods and techniques used for broadcasting messages in Wi-Fi networks. Section 2.4 presents the main methods used for locating Wi-Fi devices in an

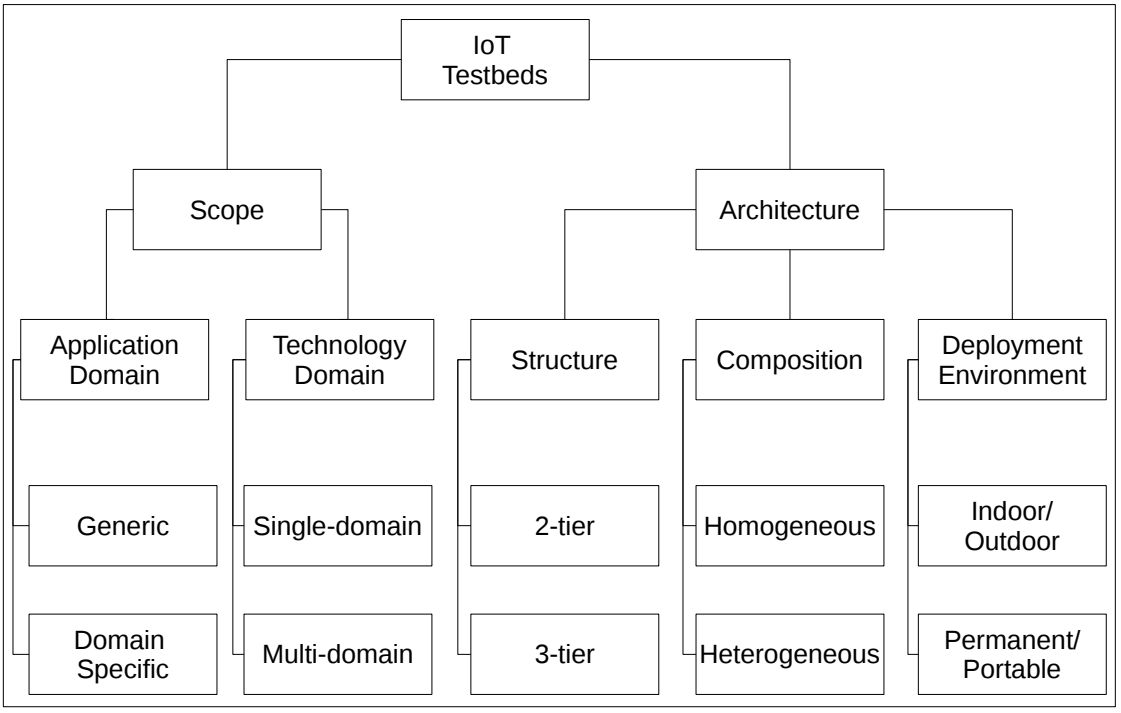

Figure 2.2: IoT Testbeds [Gluhak et al., 2011]

indoor environment. Section 2.5 discusses the Wi-Fi-based methods used in emergency evacuation.

2.1/ TESTING INTERNET OF THINGS

IoT is a recent field of research. Researches are conducted on sub-domains of IoT or on a subset of its functions. Testing IoT is considered to be one of the main challenges in this domain. Testing IoT has been added as one of the researches of many IoT work groups, such as: *Industry Work Groups* which is launched at May 28 2015 by the IoT Lab at the University of Wisconsin-Madison [Wisc, 2015]. The work group is backed by several large companies and organizations like At&T, Microsoft, IBM, Internet2, etc.
To the best of our knowledge, researches for providing a generic infrastructure, which could be used to build testbed for several IoT sub-domains at the same time and enables the resulted testbeds of exchanging services, have yet to conclude.
In [Gluhak et al., 2011], the authors categorized IoT Testbeds according to one of the following concept (Figure 2.2):

- **Testbed Scope:** It fixes domain which represents the target of the testbed. There are two types of scopes, technology-based and application-based scopes. Both scopes categorize IoT testbeds depending if they address a single domain or multiple domains.
- **Testbed Architecture:** This concept could be categorized into three subcategories. Firstly, structure-based which categorizes testbeds according to the existence of an IoT Gateway to rely between objects and servers (3-tier structure). Sec-

ondly, testbeds could be categorized depending their composition. Heterogeneous testbeds consist of several types of objects, while homogeneous testbeds consist of the same type of objects. Thirdly, environment-based concept categorizes testbeds depending on two factors: *indoor/outdoor* which specifies whether the objects are installed indoor or outdoor and *permanent/portable* which distinguishes between testbeds with permanent objects from testbeds with portable ones.

In the following, we list some of the main researches which have been conducted on testing sub-domains of IoT or on techniques and methods used in testing.
In [Younan et al., 2015], the authors propose a testbed environment for the Web of Things (WoT) which focuses on application layer. The proposed testbed allows building a testing environment for WoT. It provides description for components to give search engine spiders the ability to crawl them in addition to the ability given to users to perform live monitoring of their environment.

Since that IoT is the domain which connects all other domains, new techniques and methods for testing applications and systems should be designed by researchers and enterprises. But it is also expected that techniques and methods currently used in each domain would be evolved to meet the new requirements of the IoT. This suggestion is supported by the projects which were launched by the leader enterprises to evolve their operating systems and their frameworks to be able to deal with IoT challenges.
One of the most important techniques which has been the target of many researches is *Test Automation*. We believe that test automation techniques and methods which have been developed for mobile applications would be the base stone in developing test automation for most of IoT objects and systems. Researches on Test automation are discussed in subsection 2.1.1. Wireless Sensor Network (WSN) is one of the earliest application and systems to be categorized as a sub-domain of IoT. We cover the most important testbeds for WSN in subsection 2.1.2 There are several sub-domains such as: Smart Home, Smart Campus, Smart City, etc. We present Smart city as a representative of all other IoT *smart* sub-domains. A number of projects on smart cities have been launched since the appearance of the IoT. Open Ubiquitous Oulu and SmartSantander are among the most important projects in this domain. Subsection 2.1.3 discusses these projects in details.

2.1.1/ TEST AUTOMATION

In [Amalfitano et al., 2011], the authors propose a GUI-crawler-based technique for automatic white-box testing of Android mobile applications. The GUI crawler is used to obtain the test cases of an application. Depending on these cases, event sequences can be fired on the application GUI. This technique could be use to find runtime crashes or user-visible faults.

In [Aho et al., 2013], the authors utilize Model-based testing (MBT) in a platform-independent industrial approach and Murphy tool set for automatically extracting finite state machine (FSM) based models for testing GUI applications. MBT is a technique of generating test cases from behavioral models of the system under test (SUT). Monkeyrunner is a tool for writing test scenarios using Python for Android OS. The test scenario should be executed from outside of Android code. UiAutomation is a class for interacting with the device's UI by injecting user actions and introspection of the screen

content. In [Starov et al., 2013], the authors present Cloud Testing of Mobile Systems (CTOMS) which is a cloud to run tests using Android Monkeyrunner. It supports automate functional testing of Android applications and detection of defects in user interfaces. CTOMS consists of three subcomponents: a master application which should be deployed at Google App Engine cloud, a slave to be installed on a server at the client's site and the mobile devices that should be connected to the server. Appium [Appium, 2015] is an open-source tool for automating native, mobile web, and hybrid applications on iOS and Android platforms. It depends on writing test scenarios using one of programming languages, such as: Java, Ruby, etc. The server processes the source code in order to generate the UI commands of the test scenario. The server sends the resulting commands to the client in order to be executed. The client can send results using HTTP connection. Mobile Testing Framework (MFT) [MFT, 2015] is an alpha-state open source project to automate GUI/System tests for iPhone/iPad applications. MTF is based on the automation tool Sikuli which allows to interact with the User interface with python scripts and screenshots. Calabash [Calaba, 2015] is an automated testing technology for Android and iOS native and hybrid applications. Test scenarios should be written using Ruby API. The server generates a new instrument for each scenario and for each mobile application. CATJS [CATJS, 2015] is an automation framework for web and mobile-web applications testing which supports iOS, Android and any HTML5 Browser. Robotium [Robotium, 2015] is an open source Android test automation framework. Test scenarios should be written in Java and it generates an Android instrument for each scenario and for each application (the instrument should have the same signature of the android app). Robotium Recorder is a tool capable of registering user actions. The tool can automatically generate the Android instrument signed with the same signature of the android apk. It is available as a plugin for Android Studio and Eclipse.

2.1.2/ Wireless Sensor Network (WSN)

MoteLab [Werner-Allen et al., 2005] is one of the most important WSN testbeds.It is a Web-based testbed which consists of a set of deployed sensor network nodes connected to a central server which gives the ability to: handling reprogramming and data, creating and scheduling jobs on the testbed and automating data logging.
Kansei [Ertin et al., 2006] supports complex experimentation by integrating dedicated node resources for local computation in a heterogeneous hardware infrastructure. It provides methods for sensor data generation and real-time data and event injection. It supports utilizing real hardware resources and data generation and simulation engines.
IoT-LAB Testbed [IoT-LAB, 2015] provides over 2700 wireless sensor nodes (fixed/mobile) spread across six different sites in France. IoT-LAB offers web-based reservation and tooling for applications development, along with direct command-line access to the platform.

2.1.3/ Smart Cities

Open Ubiquitous Oulu [panOULU, 2015] is a smart-city environment located in Oulu, Finland. The UrBan Interactions (UBI) research program has created a middleware layer on top of the Public Access Network OULU (panOULU) wireless network and opened it up to ubiquitous-computing researches [Gil-Castineira et al., 2011]. SmartSantander project

aims at the creation of an experimental test facility for the research and experimentation of architectures, key enabling technologies, services and applications for the Internet of Things in the context of a city [SmartSantander, 2015] [Silva et al., 2014]. SmartSantander's architecture consists of 3 tiers:

- **IoT nodes:** The project aimed at installing 20,000 sensors in Belgrade, Guildford, Lübeck and Santander. Sensors are responsible for sensing the corresponding parameter (temperature, CO, noise, light, car presence,etc.). There are two types of sensors. First, sensors can communicate directly with gateways without passing by an external repeater. Second, sensors send their data to gateways through repeaters.
- **Repeaters:** They are forwarding points which can forward data received from sensors to gateways.
- **Gateways:** Nodes and repeaters use protocol 802.15.4 to send data to gateways. Gateways can store data locally or they can send them to servers. Gateways allow virtualization of IoT devices. This enables the instantiation of emulated sensors or actuators that behave in all respects similar to the actual devices [Gutiérrez et al., 2013].

The SmartSantander consists of four subsystems:

- **Authentication, Authorization and Accounting (AAA) subsystem:** It manages access to the testing environments.
- **Testbed Management Support Subsystem (MSS):** It manages adding/removing and configuring the resources in SmartSantander.
- **Experimental Support Subsystem (ESS):** It is responsible of reserving nodes, configuring and deploying experiments, running experiments, collecting and analyzing the produced results.
- **Application Support Subsystem (ASS):** It is responsible of facilitating the development of services and providing the possibility for lookup for specific resource.

SmartSantander project provides several services, such as [Krčo et al., 2012]:

- **Outdoor parking management:** Ferromagnetic wireless sensors were buried in parking places around the city. These sensors detect when a car is parked and transmit this information to the closest repeater.
- **Environment monitoring:** Measurement stations at fixed locations were installed. Environmental monitoring devices have been installed on lamp post, buses and police cars.
- **Participatory sensing:** Habitants of the city can interact with the system by submitting values from their own sensors using mobile phones or by receiving alerts about events in the city.

SmartSantander is an experimental facility for deploying, and assessing new services and applications. Internet researchers can validate their cutting-edge technologies (protocols,

algorithms, radio interfaces, etc.) in the domain of smart cities. SmartSantander depends on real-world sensors. The announced architecture doesn't include functions to simulate certain environment scenarios and It doesn't provide an interface to connect with other IoT testing cities.

2.2/ Wi-Fi-Based Communication Methods

In wired-based networks, clients can't exchange data with other devices of a given network unless a wire is connecting the client's Network Interface Card (NIC) to the network. At the same time, the NIC can never be connected to more than one network. In Wi-Fi-based communication, each Wireless Network Interface Card (WNIC) could be connected to one Access Point (AP), knowing that there is no physical obstacle like the one in wired-based network (e.g. cable). A client could be located in the coverage area of several APs at the same time, but the protocol IEEE 802.11 doesn't provide a method to use services from several APs concurrently. Many researches have been conducted to design solutions which enable clients to exchange data with several APs concurrently, and to enable clients to exchange data without being connected to the network. In the following, we discuss the most important works among these researches. These works could be categorized into two categories: *Multiple-Connections Wi-Fi* and *Connectionless Wi-Fi*. In the following, we discuss these two categories.

2.2.1/ Multiple-Connections Wi-Fi

Several researchers have studied the possibility of maintaining several connections between a Wi-Fi client and several APs at the same time. The Multi-Radio Unification Protocol (MUP) [Adya et al., 2004b] was one of the earliest research in this domain which proposed a design on the data link layer to build scalable multi-hop wireless network with existing IEEE 802.11 hardware. The research proposed using at least two Wireless Network Interface Card (WNIC) on each node in order to build a mesh-based community network for a neighborhood. The design depends on two main factors: static APs and unlimited power for the routers. In addition to the financial cost of using multiple WNICs, all mobile objects can't meet the main factors of this design because they aren't static and they are battery-based objects (limited power).
MultiNet [Chandra et al., 2004] was the first research to propose using one WLAN to connect to multiple network at the same time. MultiNet design consists of multiple virtual adapters for each underlying wireless network card. MultiNet is a software-based approach which introduces an intermediate layer below Network Layer which should be integrated on the Wi-Fi client-side only. MultiNet switches between APs without disconnecting by sending a fake Power Saving Mode (PSM) request to all APs except the one which it needs to keep connected to. When a client-A sends a PSM to an AP, the AP will consider Client-A as asleep and it start buffering all the frames directed to client-A in order to be sent later.
Flex-Wi-Fi [Parata et al., 2007] proposes a design which mixes the two main modes in IEEE 802.11, infra mode and ad hoc mode. Flex-Wi-Fi allows two users be associated to an AP and at the same time to establish a direct ad hoc connection at the same time. This design doesn't work with multiple APs but it provides a solution which depends on using different channel than the AP's one, subsequently that would increase the network

performance since it multiplies the system bandwidth and frees the channel. The main disadvantage of this design that the clients can't use services provided by other APs.
FatVAP [Kandula et al., 2008] and WiSwitcher [Giustiniano et al., 2009] present two designs which assume that a client has access to several APs at the same time. But since that not all APs are equal, FaTVAP and WiSwitcher aggregate the bandwidth available at APs and balance load across them. In these designs, switching between APs is transparent to the application, so they don't discuss the possibility of adding another layer at the application level so the applications can choose certain AP for using certain service (in case that not all APs provide the same services).
Juggler [Nicholson et al., 2010] depends also on the virtual adapters. One of the most important problems in using multiple APs which is addresses by Juggler is *Switching Time*. Juggler could enhance *Switching Time* between two APs/endpoints to 3 ms if they were on different channels and to 400 μs when they are on the same channel.

2.2.2/ CONNECTIONLESS WI-FI

Management frames are non-encrypted frames which are used for managing communications between APs and stations (Scanning, (re)association, dissociation, (de)authentication).
There are many types of management frames, one of these types is called *Beacon frames*. Beacon frames could be sent periodically by an AP in order to announce its presence to clients in its coverage area. All clients with WNIC in the coverage area of a given AP would be able to receive the beacons sent by this AP. By listening to received beacons, the clients can recognize nearby APs without sending any request. Clients don't have to confirm or acknowledge receiving these beacons. This method is called *Passive Scanning* and that because clients don't send any type of frames to scan the coverage area. Beacon-Stuffing [Chandra et al., 2007] is one of the earliest attempts to broadcast data using beacons frames. It is a low bandwidth communication method which depends on adding small amount of data into beacon and specifically in *Information Element (IE)* fields. The AP broadcasts the beacon, so all clients in the coverage area of this AP can receive this data even if they aren't associated with the AP, or even if they are associated with other APs different from the sender one. Beacon-stuffing provides a way to broadcast data only from an AP to a client.
BOWL [Muralidharan et al., 2008] proves the possibility of using several beacons to transmit large files between two Wi-Fi clients directly without being associated to each other or to any other AP. The transmitter operates in ad hoc mode so it can broadcast beacons while the other client operates in client mode so it listens to beacons. The client extracts file's portions from the received beacons and assemble them.
Information Embedding [Gupta et al., 2012b] and *Bit-Stuffing* [Gupta et al., 2013] extend Beacons-Stuffing by using other fields other than IE fields. They proved the possibility of adding extra information into Service Set Identifier (SSID) and Length fields.
All these researches depend on beacons which are sent only by an AP which means that Wi-Fi clients can't send any data to APs (one-way communication).

Beacon are sent only by an AP which means that Wi-Fi clients can't send any data to APs (one-way communication).

Silent broadcast [Yun et al., 2012] proved the possibility of exchanging data between two Wi-Fi clients using the vendor specific field of Wi-Fi P2P frame which could be sent by Wi-Fi clients.
In [Schauer et al., 2013], the authors utilize the NULL and the ACK frames in indoor positioning. NULL frames represent a special type of IEEE 802.11 data frames, because they merely carry a power management bit while the data field is being left empty. NULL frames have to be acknowledged by the AP. A station sending a NULL frame to an access point does not have to be associated with the latter. NULL frames are unicast frames, so a null frame can be sent to an AP. That means a station should perform active or passive scanning in order to have a list of the APs.

2.3/ BROADCASTING SOLUTIONS

Multi-tier Broadcasting using tree structure established in a network is a well-known and widely used technique in Mobile ad hoc Network (MANET) as the TreeCast [Juttner et al., 2005] method, which is based on a fully distributed, decentralized and resource-efficient algorithm that maintains a spanning tree. A MANET is an autonomous collection of mobile users that communicate over relatively bandwidth constrained wireless links. Since the nodes are mobile, the network topology may change rapidly and unpredictably over time. The network is decentralized, where all network activity including discovering the topology and delivering messages must be executed by the nodes themselves, i.e., routing functionality will be incorporated into mobile nodes [NIST, 2015]. VANET is the technology of building a robust ad hoc network which comprises vehicle-to-vehicle and vehicle-to-infrastructure communications based on wireless local area network technologies. [Hartenstein et al., 2008]. MANET and VANET are self forming network, which can function without the need of any centralized control. Each node in the network acts as both a data terminal and a router.
Broadcasting is defined to be an one-to-all communication, which means that all messages sent from a mobile node should be received by all other nodes in the same network. MANET depends mainly on broadcasting mechanism for announcements and routing protocol such as Dynamic Source Routing (DSR), Ad Hoc On Demand Distance Vector (AODV), Location Aided Routing (LAR) and Zone Routing Protocol (ZRP) depending on broadcasting mechanism.
Many studies have been conducted on broadcast methods in MANET/VANET. Many methods have been proposed and tested. These methods can be categorized in many ways, in this study we are going to use the following categorization [Williams et al., 2002]: Blind Flooding Methods, Probability-Based Methods, Area-Based Methods and Neighbor Knowledge Methods. In the following, we present the most important algorithms in each category.

2.3.1/ BLIND FLOODING METHOD

This is the simplest broadcasting method, in which each node rebroadcasts the packet whenever it receives it for the first time [Obraczka et al., 2001]. Each node might receive same packet from several nodes which causes bandwidth congestion and degrades nodes performances. *Broadcast Storm* problem is one of the consequences of

using blind flooding which is caused by the high number of redundant broadcast packets [Williams et al., 2002].

2.3.2/ PROBABILITY-BASED METHODS

In [Tseng et al., 2002], the authors present 5 schemes which differ in how a mobile host estimates redundancy and how it accumulates knowledge to assist its decision. These schemes were designed to reduce redundancy, contention, and collision. In the following, we present a brief of these schemes:

- **Probabilistic Scheme:** The scheme depends on probabilistic rebroadcasting which defines probability P. A node rebroadcasts a packet received (for the first time) with certain probability. When $P = 1$, this scheme behaves exactly as in blind flooding.
- **Counter-Based Scheme:** In this scheme, each node defines two variables, c which presents the number of times the broadcast is received and C which presents a counter threshold. A node can't rebroadcast a packet (message) which has a counter $c \geqslant C$. Whenever a node receives a message for the first time, it initializes the counter c, it will rebroadcasts the message if it wasn't heard after waiting for random number of sluts. With every new redundant message received, the node increases the counter c. The procedure will exit either by rebroadcasting the message or by hitting the threshold.
- **Distance-Based Scheme:** Each node maintains a database of distances to the other nodes. The node rebroadcasts a message only if the distance d between the sender and the receiver is larger than a distance threshold D.
- **Location-Based Scheme:** The node rebroadcasts a message only if the additional coverage due to the new emission is larger than A ($0 < A < 0.61$).
- **Cluster-Based Scheme:** In this scheme, the network is divided into clusters. Each cluster has a head and members. The head is responsible of rebroadcasting the messages to the members in its cluster, in addition to communicate with heads of other clusters [WANG et al., 2010].

The first four schemes operate in a fully distributed manner, while the fifth operates on some local connectivity information.

2.3.3/ AREA-BASED METHODS

These methods consider the coverage area of a transmission instead of considering whether the nodes exist within that area. There are two subtypes of these methods:

- **Distance-Based Scheme:** Each node depends on the distance between itself and each neighbor node that has previously rebroadcast a given packet.
- **Location-Based Scheme:** Nodes depend on Global Navigation Satellite System (GNSS) coordinates.

The concept of these methods is that when a *node A* rebroadcasts a message, it would cover a small additional area if the message was received from a *node B* which is located near *node A*. It is the opposite when *node A* and *node B* are far away from each other.

2.3.4/ Neighbor Knowledge Methods

In these methods, nodes should have knowledge of the area. In the following, we discuss two subtypes of these methods.

- **Flooding with Self Pruning:** Every node broadcasts a list containing its neighbors to all other nodes. A node would rebroadcast a message only if it has more neighbors than the sender node [Lim et al., 2000].
- **Scalable Broadcast Algorithm (SBA):** In this method, all nodes should have a list of their neighbors within a two hop radius. The list could be built using *Hello* packets, which each one should contain the neighbors of the sender. As a result, each node considers itself the center and it has a list neighbors and their neighbors. A *node A* would rebroadcast a message received from *node B* only if it has more neighbors than *node B*. *Node A* would repeat the same procedure for redundant messages from another neighbor.

2.4/ Wi-Fi-Based Indoor Positioning and Localization

Indoor positioning using the IEEE 802.11 protocol has undergone considerable progress in the past decade. Indoor positioning became one of the essential technologies for many applications, such as disaster rescue, indoor navigation and advertising. Several Indoor Positioning Systems (IPS) have been presented and implemented. In general, indoor positioning needs a number of calculations which differs according to the methods used. There are two ways to perform these calculations. One way is to perform them on a mobile device, while the other way is to perform them on a server. Performing the calculations on a mobile device consumes the device's battery, and since mobile devices are normally battery-driven, energy efficiency is a very important consideration in Wi-Fi localization systems [Niu et al., 2013]. Some methods such as Wi-Fi fingerprint-based localization solve part of this problem by sending the needed parameters to a server in order to perform the calculations.
These systems can be categorized into three groups according to their methods [Zahid Farid, 2013]: Proximity Detection, Fingerprinting (Scene Analysis), Trilateration and Triangulation. In the following, we present the main methods and algorithms in each group.

2.4.1/ Proximity Detection

It is one of the simplest method to implement. The position (or location) of a wireless device is the same position as those of the AP from which it receives the strongest signal. This method is implemented in several system such as: Infrared Radiation (IR), Bluetooth, Radio Frequency Identification (RFID), GSM systems, etc.

2.4.2/ FINGERPRINTING

Fingerprinting (Scene Analysis) depends on fingerprints. A fingerprint means a signature of environment features consistently and strongly depending on the physical location [Deak et al., 2012]. These features could be any parameters in the environment such: Received Signal Strength Indicator (RSSI), GSM Signals, Bluetooth signals, etc. Fingerprinting consists of two phases [Szabolcs Karsai, 2014]:

- **Off-line Phase:** In this phase, a database of fingerprints will be built by collecting environment features of the site. This phase should be repeated every time the environment features are changed.
- **On-line Phase:** The system matches the fingerprints collected by the clients to the database in order to calculate their locations or positions.

It is virtually impossible to use this method without a significant error, because it doesn't take into consideration the interference or the obstacles in the area, such as walls, furniture, and even other people in the building.

2.4.3/ TRILATERATION AND TRIANGULATION

Trilateration depends on geometric properties of triangle which are the distances between transmitters and receivers. This method starts by building a map of the distribution of APs in a building. The location or the position of the wireless device is determined depending on the lengths between each detected AP and the wireless device. This method requires at least three location-known APs to be detected by the wireless device but more APs could give more accurate location or position [Muthukrishnan et al., 2005]. In the following, we review the most important techniques used to measure lengths between transmitters and receivers:

- **Time-Based Methods:**
  - **Time of Arrival (ToA)/Time of Flight (ToF):** It depends on the accurate synchronization of the arrival time of a time-stamped signal transmitted from wireless device to several APs. The distance is calculated using the speed of signal and the transmission time delay. This method requires APs and wireless devices to be time synchronized.
  - **Time Difference of Arrival (TDoA):** TDoA requires the APs to be time synchronized but not the wireless devices. It depends on multiple pairs of APs with known locations and use relative time measurements at each AP in place of absolute time measurements.
  - **Round Trip Time (RTT)/Round-Trip Time of Flight (RToF):** In this method, one node can record the transmitting and arrival time. The advantage of this method over ToA methods is the needless of time synchronization.
- **Single-Property-Based Method:** This method depends on measuring the attenuation of received signal strength. Received Signal Strength Indicator (RSSI) depends on the environmental interference [Bahl et al., 2000] [Tian et al., 2013].

Triangulation depends on geometric properties of triangle, same as Trilateration. The difference is that Trilateration depends on distances between transmitters and receivers and Triangulation depends on measuring angles of arrival of the signals. Measurements are done using Angle of Arrival (AoA) which requires additional antennas capable of measuring the angle of arrival of a wireless signal received from a known location [Zhang Da, 2010].

2.5/ WI-FI-BASED EMERGENCY EVACUATION

In case of a catastrophic disaster, emergency evacuation is very critical to many lives. Communication networks for emergency warning systems could be categorized into four groups labeled as Wi-Fi, P2P, Cellular Network, and Satellite [Li, 2011].
Many researches have been conducted to design systems for broadcasting alerts and helping evacuation teams by providing information about people in the stricken areas.
In [Fujiwara et al., 2004], the authors propose a schema of a multi-hopping hybrid wireless network. It aims at maintaining the connection between a cellular base station (BS) and nodes. In case of losing the direct link between BS and a node, the node tries to access BS indirectly (via another node) by switching modes to ad hoc. The authors propose a routing protocol and a MAC protocol. The routing protocol is capable of building a route using unicast-based route discovery process without route request flooding. The MAC protocol maintains accessibility and a short delay in emergency circumstances.
In [Fantacci et al., 2010], the authors presented Integrated System for Emergency (In.Sy.Eme.) which integrates the mobile grid paradigm in the infrastructure. They suggest Wireless Sensor Network (WSN) and MANET for monitoring and WiMAX with suitable Quality of Service (QoS).
In [Bai et al., 2010], the authors provide a design which depends on MANET and WSN for monitoring and a satellite link for communicating with other sites.
In [Simmel, 2012], the authors used Beacon-Stuffing in evacuation system. The smart device of the person who requests help broadcasts emergency message which consists of the identifier of the person's device and GPS coordinates (if exists). The emergency messages could be detected by drones (e.g., quadcopters) which relay these requests to the authorities. If an emergency message contains no GPS coordinates, the drone would add its current GPS coordinates instead. We should notice here that the device which is broadcasting the emergency messages should be in AP mode in order to be able to do so.
Except for the last research, we can notice that most of the researches in this domain concentrate on building MANETs or WSNs.
For our best knowledge, there is no extension for the protocol IEEE 802.11 to deal with emergency situations. The protocol isn't designed to give exceptions in time of catastrophic disasters. For example, there is the number 112 in the mobile communications which allows a callet to contact emergency services even if the mobile has no subscriber identification module (SIM) card. This is an exception which has been integrated in mobile networks in many countries. Wi-Fi networks have not integrated such an exception to deal with emergency situations.

2.6/ CONCLUSION

The current state of the art in IoT testing has been discussed in this chapter. IoT is integrating all other domains. In the domain of testing, it is expected that all techniques and methods of testing would be customized and integrated in order to provide a testing environment for IoT. We presented the most important methods for Test Automation. In each IoT sub-domain, several studies have been conducted so as to develop a testbed for that sub-domain precisely. We have reviewed some examples on testbeds for Wireless Sensor Network (WSN) and smart cities. There are many other testbeds for smart homes, smart campus, smart grid, etc. Since that all these testbeds have been developed separately, it is unlikely that they would be able to be integrated easily in one testbed. A generic architecture for testing IoT has yet to be described in literature. This testbed might help the enterprises and the developers in testing their IoT systems, IoT protocols and IoT applications. The second contribution in our thesis is an example of the protocols that might be tested and evaluated using this testbed.
Several protocols and standards have been provided in order to enable IoT. We presented the state of the art of techniques and methods which have permitted to use multiple Wi-Fi connections concurrently. Researchers have developed solutions depending on one of the following techniques: using multiple WNICs, virtualizing WNICs, mixing infra mode and ad hoc mode and finally switching between multiple APs. Other researchers have developed techniques in order to piggyback beacon frames so as to send data to clients even if they aren't associated with any AP.
We discussed the state of the art of three Wi-Fi-based applications: Broadcasting, Indoor positioning and Emergency Evacuations. These applications depend on infra mode or ad hoc mode in which clients should be associated to an AP. We showed that connectionless methods haven't been adopted by these applications (and many other). These applications provide public and non-confidential data (most of the time). In IoT, public data would represent a considerable amount of exchanged data (transport, emergency, positioning and localization, etc). Providing these public services to all clients in a given area, even if they are not connected to same network or not connected to any network at all, would be the easiest and the most secure method.
In this thesis, we present a testbed for testing IoT systems and protocols, in addition to an extension to the protocol Wi-Fi (Connectionless Data Exchange) to be tested using the testbed. Since that there are a lot of applications which might be developed using this extension, we present a study for three applications: Broadcasting, Indoor positioning and Emergency Evacuations. The two first applications (Broadcasting and Indoor positioning) are important enablers to the development of the third one (Emergency Evacuation).

3

Fundamentals of Application Testing and Evaluation

Software testing and software evaluation are the processes of verifying and validating that a software application or program meets the business and technical requirements that guided its design and development and works as expected. They also identify important errors or faults categorized per the severity level in the application that must be fixed [John E. Bentley, 2005]. It would not be right to say that testing is done only to find faults. Faults will be found by everybody using the software. Testing is a quality control measure used to verify that a product works as desired [Quadri et al., 2010].

There are several application development methodologies in use today. Mainly there are two kinds of methodologies: heavyweight and lightweight. Heavyweight methodologies, also considered as the traditional way to develop software, claim their support to comprehensive planning, detailed documentation, and expansive design. The lightweight methodologies, also known as agile modeling, in which it employs short iterative cycles, and rely on tacit knowledge within a team as opposed to documentation [Awad, 2005]. All these methodologies acknowledge that testing and evaluation form an important phase for assessing the quality of an application.

Two common types of testing are black-box and white-box testing. The basic difference between the two classes is clarified by the definitions below [IEEE, 1990]:

- **Black-Box Testing *(Functional Testing)*** Testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generated in response to selected inputs and execution conditions. It should be the focus for testers.
- **White-Box Testing *(Structural Testing)*** Testing that takes into account the internal mechanism of a system or component. Types include branch testing, path testing, statement testing. It should be the focus for developers.

In the following, we summarize evaluation and testing methods and techniques which could be categorized according to three aspects: *General Categorization* which is covered in section 3.1, *Test-Case-Based Categorization* is reviewed in section 3.2 and *Application-Based Categorization* which presents in section 3.3 the differences between the traditional applications, web applications and mobile applications from evaluation and testing point of view.

3.1/ General Categorization

There are several types of testing that should be done on a large software system. Each type of test has a *"specification"* that defines the correct/incorrect behaviors. Each type has four attributes [Williams, 2004] [IEEE, 1990]:

- **Opacity:** The tester's view of the code (is it white or black box testing).
- **Scale:** Whether the tester is examining a small bit of code or the whole system and its environment.
- **Specification:** What we look at to develop the tests.
- **Tester:** The programmer who wrote the code, independent tester or a customer.

We present the main types of testing in the following list:

- **Unit Testing:** Unit testing is the testing of individual hardware or software units or groups of related units. Testers verify that the code does what it is intended to do at a very low structural level.
- **Integration Testing:** Integration test is testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them. The tester verifies that units work together when they are integrated into a larger code base.
- **Functional and System Testing:** Functional testing involves ensuring that the functionality specified in the requirement specification works. System testing involves putting the new program in many different environments to ensure the program works in typical customer environments with various versions and types of operating systems and/or applications. Several classes of testing can be done that can examine non-functional properties of the system:
  - **Stress Testing:** Testing conducted to evaluate a system or component at or beyond the limits of its specification or requirement.
  - **Performance Testing:** Testing conducted to evaluate the compliance of a system or component with specified performance requirements.
  - **Usability Testing:** Testing conducted to evaluate the extent to which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
- **Acceptance Testing:** Acceptance testing is formal testing conducted to determine whether or not a system satisfies its acceptance criteria (the criteria the system must satisfy to be accepted by a customer) and to enable the customer to determine whether or not to accept the system.
- **Regression Testing:** Regression testing is selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements.

- **Beta Testing:** The development organization can offer it free to one or more potential users. These users use it with the understanding that they will report any errors revealed during usage. These users are usually chosen because they are experienced users of prior versions or competitive products.

Table 3.1 presents the differences between these types.

Testing Type	Opacity	Specification	Scope	Tester
Unit Testing	*White box*	Low-level design and/or code structure	Low-Level Design, Actual Code Structure	The programmer who wrote the code
Integration Testing	*Black- and white-box*	Low- and high-level design	Low-Level Design, High-Level Design	Independent tester
Functional and System Testing	*Black-box*	high-level design, requirements specification	High-Level Design	Customer
Acceptance Testing	*Black-box*	Requirements specification	Requirements Analysis	Customer
Regression Testing	*Black- and white-box*	high-level design	Low-Level Design, Actual Code Structure	Customer
Beta Testing	*Black-box*	None	Changed Documentation, High-Level Design	Programmer(s) or independent testers

Table 3.1: Types of testing

3.2/ TEST-CASE-BASED CATEGORIZATION

Testing process consists of series of actions and events, in which testers would use them as an input. There are two models to generate these series, Scenario-Based methods (also known as structure-based methods) and random methods. In the following subsections, we discuss each of them briefly.

3.2.1/ RANDOM TESTING

Random testing means that test inputs are randomly selected from the input space. The main idea of random testing is that randomness is not influenced by the tester [Dadeau et al., 2008]. Although random testing is able to discover some errors quite quickly, it often needs to be complemented with other techniques to increase test coverage. The effectiveness of such more systematic testing methods is usually evaluated

by comparison to random testing [Delgrande et al., 2011].

3.2.2/ Scenario-Based Testing (Structure)

A scenario is a description of an imaginable or actual action and event sequence. Scenarios facilitate reflections about (potential) occurrences and the related opportunities or risks. Furthermore, they help to find solutions or reactions to cope with the corresponding situations [Strembeck et al., 2004]. Scenarios (Use cases) are used to describe the functionality and behavior of a (software) system in a user-centered perspective [Ryser et al., 1999]. In other words it concentrates on what the user does, not what the product does.
Scenarios can be designed by utilizing one of the following methods:

- **Behavior Prediction:** The testers imagine series of actions and events that could be applied by a user. In small and mid-size software system, this method would be feasible, especially if the user-input is limited.
- **Capturing Events:** In the large software systems, there are thousands of possible scenarios, especially in the softwares that analyze huge volume of user input. The testers capture series of user actions and analyze them in order to regenerate them either by changing some input or by testing them on different platforms.

3.3/ Application-Based Categorization

Testing can be categorized depending on the targeted application. The traditional applications use the methods mentioned in (Subsection 3.1). In this subsection, we are presenting the particularity of testing the other types of applications (Web applications and mobile applications).

3.3.1/ Web Application Testing

Due to the particularities of web applications, testing them may be even more difficult than testing traditional ones. Web applications have strict requirements of reliability, usability, interoperability and security. Due to the exponential growth of web-based applications and the vast number of users, testing is often neglected by developers. The following list summarizes the characteristics of web applications that make them different from traditional applications, from the point of view of testing [Di Lucca et al., 2006]:

- **Users:** Wide number of users distributed all over the world and accessing it concurrently.
- **Hardware:** Heterogeneous execution environments composed of different hardware (Servers, Routers, Firewalls), network connections, etc.
- **Software - Server Side :** Different server operating systems, web servers, database management systems, different versions of systems and models.

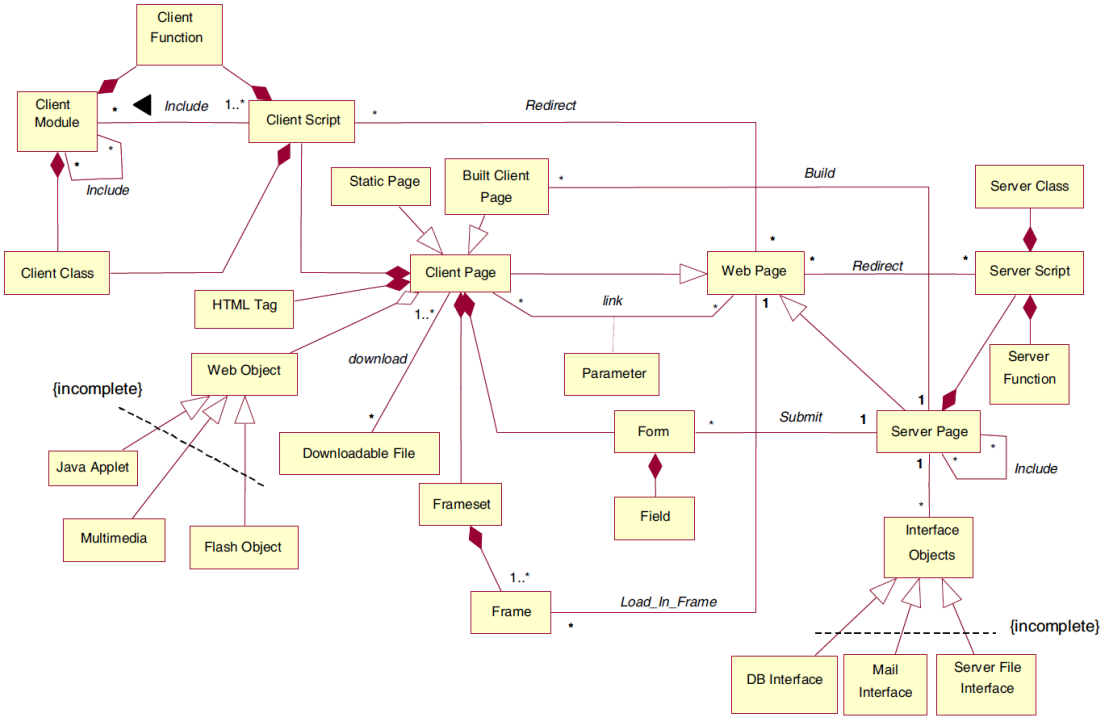

Figure 3.1: The meta-model of a Web Application [Di Lucca et al., 2004]

- **Software - Client Side :** Different users operating systems, web browsers, different localization settings, different languages and models.
- **Contents (Pages) :** Static contents, dynamic contents (per user, per network, per country, per language, etc).

Figure 3.1 presents the possible items that can be identified in a web application. These elements can be Web pages, or scripting modules, forms, applets, servlets, or other Web objects. Web pages can be static or dynamic. While the content of a static Web page is fixed, the content of a dynamic page is computed at run time by the server. There are two types of testing techniques [Ricca et al., 2001]:

1. **Static verification:** analyzers scan the HTML pages in a Web site and detect possible faults and anomalies.
2. **Dynamic validation:** It depends on White-box testing, it aims at exercising the system by supplying a vector of input data (test case) and comparing the expected outputs with the actual ones after execution.

3.3.2/ MOBILE APPLICATION TESTING

As mobile applications become more and more complex and ubiquitous, testing mobile applications becomes a non-trivial process that takes more of time, effort and other resources. Developers want to support as many devices as possible. Testers need to have different types of mobile devices in order to test and validate mobile application.

There are many challenges in mobile application testing due to the nature of these applications [ARZENŠEK et al.,]:

1. **Mobile Connectivity:** Mobile applications connect to mobile networks, which can vary in speed, security and reliability.
2. **Resource constraints:** Mobile applications use the resources of mobile devices, which are very limited. The excessive use of resources can reduce the performance of mobile devices and can cause malfunctions in the mobile application. During the testing process the consumption of resources must be constantly monitored.
3. **Autonomy:** Different activities have a different impact on autonomy and during the testing process all have to be monitored. All the device's resources and activities use energy but not equally. GPS sensors, data transfer and video editing are activities that use more energy than others.
4. **Diversity of user interfaces:** Mobile operating systems have different user interfaces, which are defined by rules and guidelines. Different mobile devices can react differently to the same application code, which must be tested with Graphical User Interface (GUI) testing.
5. **Context awareness:** Context aware mobile applications adapt and evolve based on the data obtained from the environment. To insure the correctness of applications operation, context-specific test selection techniques and coverage criteria have to be produced.
6. **Diversity of mobile devices:** There are many different mobile devices, made by different vendors, which have different hardware and software settings. The diversity of mobile devices can also increase the costs and duration of the testing process.
7. **User experience:** The adequacy of the user experience cannot be directly tested because of the subjective nature of the entire process.
8. **Touch screens:** Touch screens enable the display and input of data as individual values or as a group of data. Touch screen is also dependent on the mobile device's resources, in which it varies from device to another and from a user to another.
9. **New programming languages and mobile operating systems:** Programming languages for mobile applications have been designed to support mobility, resource management and new graphical user interfaces. Mobile operating systems are new and still only partially reliable. To analyze the code it is necessary to be aware of the specifics of the programming languages and how they operate, in addition to the mobile operating system and its updates.

There are four approaches for testing mobile applications [Shaw, 2014] [Gao et al., 2014]:

1. **Emulation-based testing**
   Emulation is to emulate (imitate) the behavior of a hardware device in software or with a different hardware, or to emulate the behavior of a piece of software either with another hardware or software. The emulator is the hardware of the software that does the emulation [Del Barrio et al., 1998].

The emulation-based testing approach involves using a mobile device emulator (also known as a device simulator), which creates a virtual machine version of a mobile device for study on a personal computer. It is often included with a mobile platform's software development kit (such as Android SDK). It is relatively inexpensive because no testing laboratory is needed and no physical devices have to be purchased or rented. Often functionality is limited on these devices. For example, testing gesture recognition is difficult or impossible on emulators. Another limitation is its limited scale for evaluating QOS.

2. **Device-based testing**
   The device-based testing approach requires setting up a testing laboratory and purchasing real mobile devices. The number of different models of mobile devices on the market today is already large and growing fast. Building a testing laboratory using actual mobile devices equipped with different operating systems isn't feasible solution even for some large enterprises. These experimental laboratories typically occupy a large amount of lab space, cost a lot of money to construct, update, and require considerable human expertise to operate. In addition, given their experimental nature, there is little or no opportunity to recapture the costs through the applications. But on the other hand, it overcomes the limitation of testing gesture-based application in *Emulation-based testing*. But it doesn't solve the problems related to system QoS because large-scale tests require many mobile devices, which is usually impossible for enterprises.

3. **Cloud-based testing**
   The basic idea is to build a mobile device cloud that can support testing services on a large scale.
   A research from Fujitsu [Fujitsu, 2010] suggests that testing and application development rank second (57%) as the most likely workload to be put into the cloud after Websites (61%). There are several factors that account for the migration of testing in the cloud [Priyanka et al., 2012]:

   - Testing is a periodic activity and requires new environments to be set up for each project.
   - Moving testing to the cloud is seen as a safe bet because it doesn't include sensitive corporate data and has minimal impact on the organization's business-as-usual activities.
   - Applications are increasingly becoming dynamic, complex, distributed and component-based, creating a multiplicity of new challenges for testing teams.

   Cloud-based testing addresses the significant increase in demand for mobile testing services by using a pay-as-you-go business model.

4. **Crowd-based testing**
   The crowd-based testing approach involves using freelance or contracted testing engineers or a community of end users. Crowd-based testing requires:

   - Testing infrastructure.
   - Service management server to support diverse users

   This approach offers the benefits of in-the-wild testing without the need to invest in a lab or purchase or rent devices, but at the risk of low testing quality and an uncertain validation schedule for the other types of testing, depending on the geographic

distribution of the testers. This method could give more accurate results in testing LBS (Location-Based Services) applications [Gao et al., 2014].

Having some criteria for selecting mobile application testing tools based on identified challenges and issues is crucial condition for testers. (Table 3.2) [ARZENŠEK et al.,] proposes a list of criteria that are defined based on the challenges.

Challenge / Testing strategy	**Properties**	**Values, range**	**Supported feature**
Mobile Connectivity - *Connectivity testing*	Data transfer speed	Range of speeds (2G, 3G and 4G)	Supports changing or limiting the data transfer speed
Mobile Connectivity - *Functional testing*	Mobile network	Constant, partial, none	Supports changing the consistency of the mobile network
	Bluetooth	Enabled, not enabled	Supports Bluetooth connectivity
	NFC	Enabled, not enabled	Supports NFC connectivity
	Wi-Fi	Enabled, not enabled	Supports Wi-Fi connectivity
	Wi-Fi Direct	Enabled, not enabled	Supports Wi-Fi Direct connectivity
Resource constraints - *Performance testing*	CPU	1 core, 2 core, 4 cores	Supports changing or limiting the operation of CPU cores
	CPU Speed	(1Mhz to 2500Mhz)	Supports changing or limiting the operation of CPU speed cores
	RAM	(16Mb to 4Gb)	Supports changing or limiting amount of RAM
	Memory	(16Mb to 128Gb)	Supports changing or limiting the amount of memory
Autonomy - *Load testing*	Consumption	Percentage of the total battery capacity	Supports monitoring the battery consumption
	Duration	Time of the total battery capacity	Supports changing or limiting the operation of CPU speed cores
Diversity of user interface - *Usability testing*	Guideline checker	Mobile platform specific Rules and guidelines	GUI guideline checker
Context awareness - *Functional testing*	GPS	Simulated, real data, not enabled	Simulate data from the GPS

Continued on next page

Continued from previous page

Challenge / Testing strategy	Properties	Values, range	Supported feature
	Neighbor devices	Simulated, real data, not enabled	Simulate data from the neighbor device
	Altitude	Simulated, real data, not enabled	Simulate data from the barometer
	Brightness	Simulated, real data, not enabled	Simulate data from the light sensor
	Temperature	Simulated, real data, not enabled	Simulate data from the temperature sensor
	Context	Simulated, real data, not enabled	Simulate data from the environment and the user
	Context adaption	Enabled, not enabled	Simulate data from the context in real time
Diversity of mobile devices - *Functional testing*	Vendor and model	Enabled, not enabled	Simulation of a specific mobile device
	Operating system	Android, iOS, BlackBerry, Windows Phone 7 and 8	Supports changing mobile device platform
	Operating system versions	Enabled, not enabled	Supports changing mobile device platform to different versions
Diversity of user interface - *Usability testing*	Screen dimensions	Small (at least 426dp x 320dp), normal (at least 470dp x 320dp), large screen (at least 640dp x 480dp), extra large screen (at least 960dp x 720dp)	Supports changing screen size
User experience - *Usability testing*	Layout checker	Enabled, not enabled	Simulate data from the GPS
	Text visibility	Supported, not supported	Supports internationalization

Continued on next page

Continued from previous page

Challenge / Testing strategy	Properties	Values, range	Supported feature
	Text grammar	Simulated, real data, not enabled	Simulate data from the barometer
	Notifications	Enabled, not enabled	Supports notification management
	Interruptions	Enabled, not enabled	Supports interruptions management
Touch screens - *Usability testing*	Responsiveness	Enabled to measure, not enabled to measure	Supports measuring the responsiveness of the screen
	Gestured	Enabled, not enabled	Supports gesture recognition
	Multi touch	Enabled, not enabled	Supports multi touch recognition

Table 3.2: Criteria defined based on challenges

II

CONTRIBUTION - IOTAAS

4

Internet of Things Testing As A Service (IoTaaS)

4.1/ Introduction

As mentioned in subsection A.5, there are five acknowledged types of cloud services offerings: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS), Testing as a Service (TaaS) and Mobile Testing as a Service (mTaaS). Actually researchers and enterprises have defined many other types of services such as: Storage-as-a-service, Database-as-a-service, Information-as-a-service, Process-as-a-service, Integration-as-a-service, Security-as-a-service and Management/governance-as-a-service [Linthicum, 2009]. Designing an architecture for testing and evaluating IoT systems requires more than providing a server and several servers. It requires the ability to build all the details of the real environment where the system will be deployed/run. Such environment could be built using a mix of the following items: real things (devices), simulators and emulators. In this chapter, we will present our architecture for building a cloud environment for offering IoT testing as a service. Section 4.2 discusses the main concept and the targets of building an IoTaaS. Section 4.3 presents the main architecture used to build an IoTaaS. Sections 4.4 to 4.9 describe the different components of the proposed architecture.

4.2/ IoTaaS Concept

Having one environment for testing all IoT systems would be an impossible target to realize. Any IoTaaS architecture (Figure 4.1) should give the possibility to have several separated and distributed IoTaaS clouds. Each cloud should be manageable separately from the other IoTaaS, and it should consist of all the components needed for testing certain IoT system(s). An IoTaaS could communicate with another IoTaaS(s) in order to demand a service or the usage of a certain resource. Each IoTaaS should be able to permit/deny access to its services and resources depending on a security model (Authentication, Authorization, Encryption). The inter-IoTaaS services could be paid ones. An IoTaaS might be formed of two or more IoTaaSs.
IoTaaS design aims at providing the researchers and the enterprises with a standard to build their own clouds for testing IoT systems. At the same time, they can cooperate among them by using some services or resources from certain IoTaaS which might

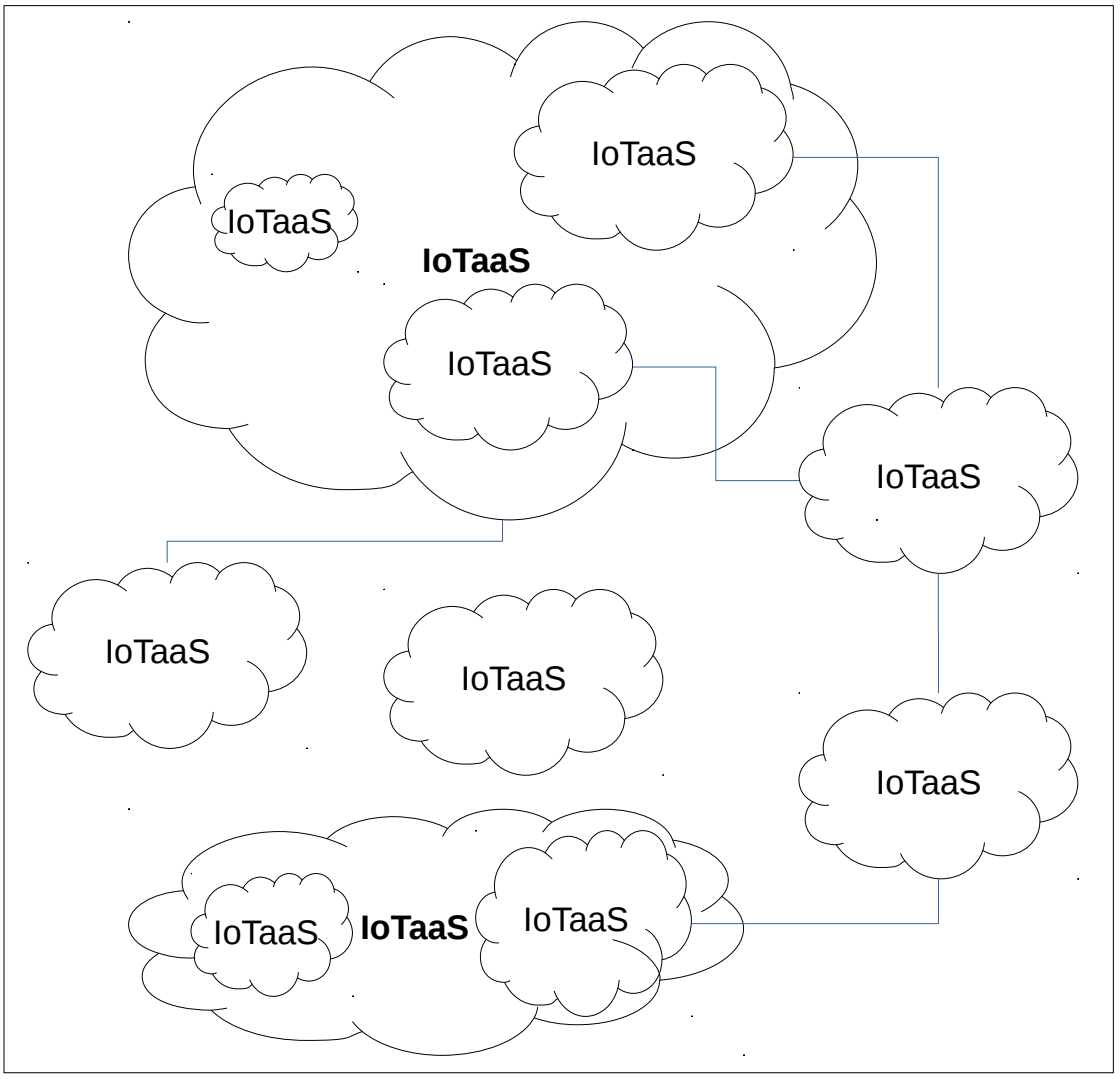

Figure 4.1: IoTaaS Concept

not be available in other IoTaaSs. IoTaaSs could utilize services from present-day cloud computing providers, such as Amazon, Google, etc. It is important to mention here that IoTaaS is an abstract design dedicated for testing IoT, this design could be customized and integrated in any other cloud computing frameworks.

4.3/ IoTaaS Architecture

An IoT environment would consist of any device might be able to exchange data directly/indirectly with Internet/Intranet. In general, a simple IoT environment is formed of:

- **Sensor:** It is a device which can detect a physical state and convert it into data (readable by computer).
- **Actuator:** It is a device which can change a physical state.

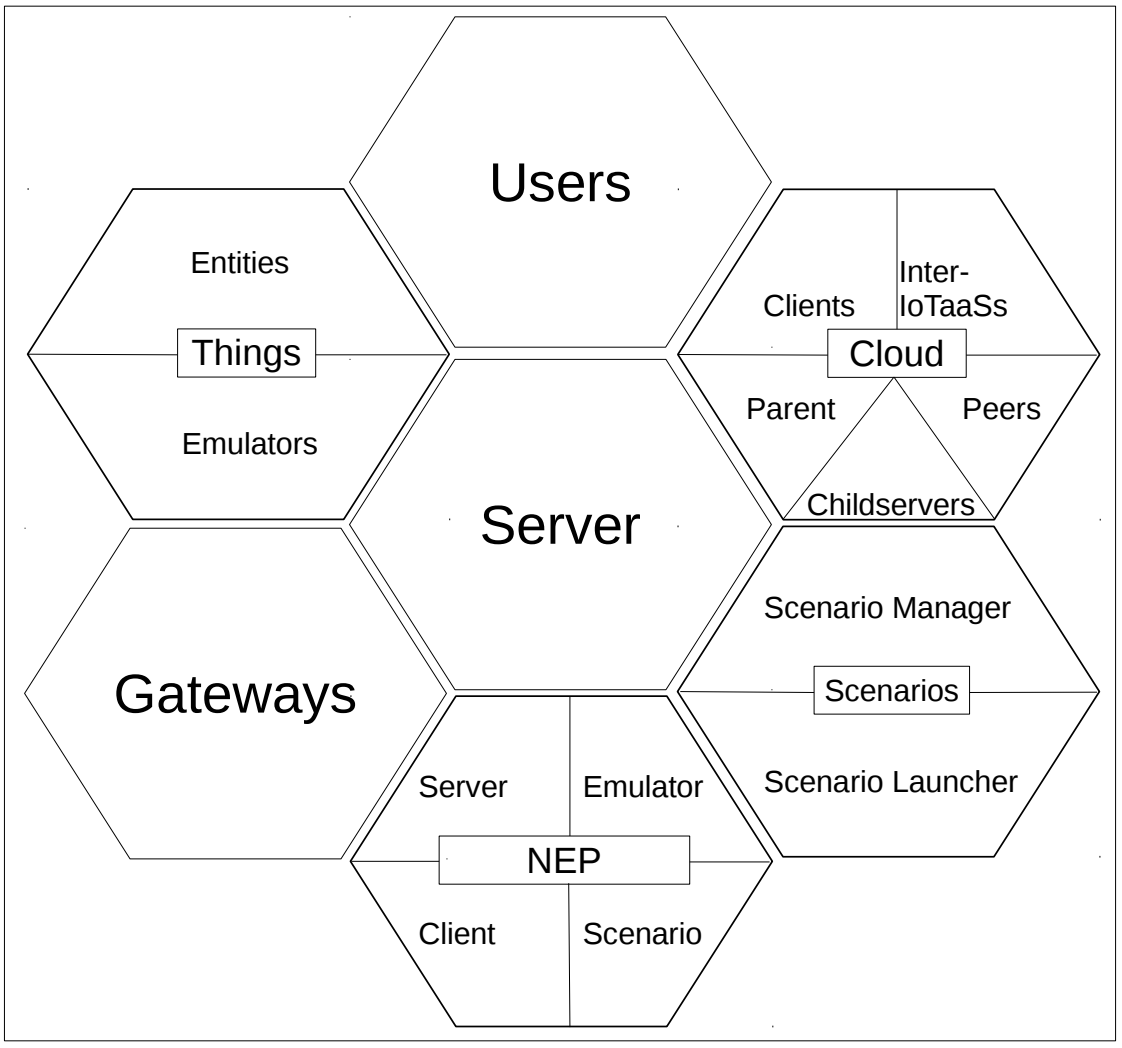

Figure 4.2: IoTaaS Architecture

- **Gateway:** It is a dedicated device or an application to read data from sensors, send data to actuators.
- **Network:** Many types of networks would exist in an IoT environment. Some networks would be traditional ones, such as Wi-Fi, while others would use technologies which have been developed for IoT.
- **Application:** It is the software which processes data. It could be hosted on any type of devices, such as: server, personal computer (PC), mobile, tablet, etc.

The existence of these components could be realized by providing the devices (ex. sensors), or by using emulators. An IoTaaS might be used to test and evaluate any component of an IoT. For example, new sensors should be tested with certain IoT frameworks or with certain gateways, an updated IoT frameworks should be tested in order to test their compatibility with the old actuators, etc. The proposed IoTaaS architecture consists of 7 modules (components) (Figure 4.2). These components are the following: Things, Gateways, Emulators, Network Emulation Protocol (NEP), Scenarios, Cloud and the Server. All the IoTaaS components are described in details in the following sections.

4.4/ THINGS

Things are literally all objects in our physical and virtual worlds. In this section we divided these objects to two types, Entities and Emulators. In the following we discuss each one of these types.

4.4.1/ ENTITIES

These are the main components in any IoT architecture. Internet-of-Things-Architecture (IoT-A) B.3 differentiates between two main types of entities, Physical Entity and Virtual Entity. A Virtual Entity is the representation of a Physical Entity in the digital world. An Augmented Entity is the composition of one Virtual Entity and the Physical Entity it is associated to. The IoT can communicate with physical entities and with Augmented Entity through their virtual entities. So *entities* here mean Virtual Entities. For example, in a mobile device, the part, that IoT architecture is interested in, is the operating system that can control the device itself, or any application can perform certain function on the device. An entity could be any object which is communicable directly by the IoT's system. Computers and mobile devices are the classical examples of entities. In the last years, hundred of objects and devices have been evolved to be connectable entities, such as watches, home appliances, cars, medical instruments, industrial machines, etc.
We should mention here that entities include only the objects and the devices that are communicable *directly* using their own virtual entities. For other devices which aren't capable to do so, IoT defined the concept of *Gateways* which is covered in section 4.5. The operating systems used in entities would be any traditional system, such as Linux, Unix, Windows, IOS, OSX, Android, etc. The system could be an IoT OS such as Contiki, Tiny OS, RIOT, etc. Applications of entities would be developed using any programming language.

4.4.2/ EMULATORS

An emulator is hardware or software that enables one computer system (called the host) to behave like another computer system (called the guest). An emulator typically enables the host system to run software or use peripheral devices designed for the guest system. Researchers and developers agree that IoT will have many operating systems, many devices architectures, many programming languages. One IoT rarely will consist of one type of objects, or of similar objects (functionally) having the same operating systems for example. The variety is one of the main points to be considered in IoT architectures.
Building an environment for testing IoT applications would require mimicking thousands of objects using emulators. Emulators are an important part in any environment for testing applications, because it wouldn't be possible for one framework to include all types of objects for many reasons, such as:

- The cost.
- Installation and operational efforts.
- Special environmental conditions could be needed.

- The complexity of testing some application in the real environment while covering all situations, such as medical applications.

One of the main points to be considered in any IoT architecture is the ***Heterogeneity***

4.5/ GATEWAYS

An IoT Gateway is a joint point that help to connect an object or a set of objects with the IoT infrastructure. A gateway can be a hardware or a software, and it should be able to translate the data from/to the objects which are connected to. Gateways can be used to do many functions, such as:

- Reading data from sensors and send it to a server or to other objects.
- Relaying orders from a server or other objects to actuators.
- Processing data from a set of objects and sending the result to the concerned entity.
- Providing a level of security by encrypting data or by controlling access to the objects.
- Storing data in case of losing connection with the IoT infrastructure.
- Taking local decision in case of emergency or the need to take an instant decision.

Gateways enable the connectivity of legacy devices by providing the needed interface. They enable the next generation intelligent infrastructure to be as simple as possible by providing important functions which could add many level of complexity in case of integrating them into the objects directly.

4.6/ NETWORK EMULATION PROTOCOL (NEP)

Network in IoT infrastructure can be of any type. Classical technologies in nowadays Internet and technologies designed specifically for connecting IoT objects coexist in the same infrastructure. Traditional Internet consists of billions of devices (computers, mobile devices, printers, cameras, etc.) connected using different types of technologies, such as: Ethernet, FastEthernet, GigaEthernet, IEEE 802.11, xDSL,2G,3G,4G, etc. Some protocols and technologies have been designed for IoT, such as: IEEE 802.15.5, NFC, Bluetooth Low Energy (BLE),etc. Each one of these technologies has different characteristics (Bandwidth, throughput, latency, jitter, error rate, power consumption, etc.). These technologies are mainly used as follows:

- **Server to Infrastructure:** Testing environments are usually equipped with high speed networks in order to give the best performance for exchanging data between the servers and the infrastructure. Classical technologies are usually used.
- **Object to Infrastructure:** The technology used differs according to the object. All traditional and IoT technologies are used.

- **Object to Object:** Technologies based on Ad hoc and mesh are mostly used to exchange data between objects directly.

In the real environments, network are subject to performance problems such as: random delays, transmission errors, packet loss, uncontrolled congestion, etc. In testing environments, generally network don't have such problems. Testing applications in such environments provide misleading results.
In simulators, simplified mathematical models of data sources, channels and protocols are applied to change the characteristics of a network in order to degrade its performance. In network emulation, one simulates the properties of an existing or planned network using emulation of specific network equipment.
IoT applications are meant to be connected permanently. Objects are connected using different technologies, and the generated data goes through different segments of network. Testing environments should have a distributed system for network emulation integrated in testing servers. This system should be capable to perform the following tasks:

- Controlling traffic in different segment of networks.
- Classifying traffic into multiple flow according to OSI Layers 2→7.
- Emulating specific network access (Zigbee, IEEE 802.11, 2G, 3G, etc).
- Analyzing response time and packet loss between network devices.

Our proposal for this system consists of four main components (Figure 4.3): NEP Server, NEP Emulator, NEP Client and NEP Scenario. In the following subsection, we are describing the functions of these components. It is important here to mention that these components could be distributed over different networks (PAN,LAN,WAN,etc). Network segment could consist of many physical segments.

4.6.1/ NEP SERVER

It is the coordinator that controls all other components. NEP Server can run any operating system on a dedicated machine, or along with a NEP Emulator. It can be located in any network segment as long as it is accessible by all NEP Emulators. NEP Server hosts the emulation database which contains the following data:

- NEP Emulators data, such as: network address, network connection, network characteristics, GNSS coordinates, etc.
- Network segment statistics.
- Scenarios: Time-based, segment-based and application-based scenarios.

NEP Server sends scenarios to NEP Emulators through their NEP Updater subcomponent. Each testing environment needs at least one NEP Server.

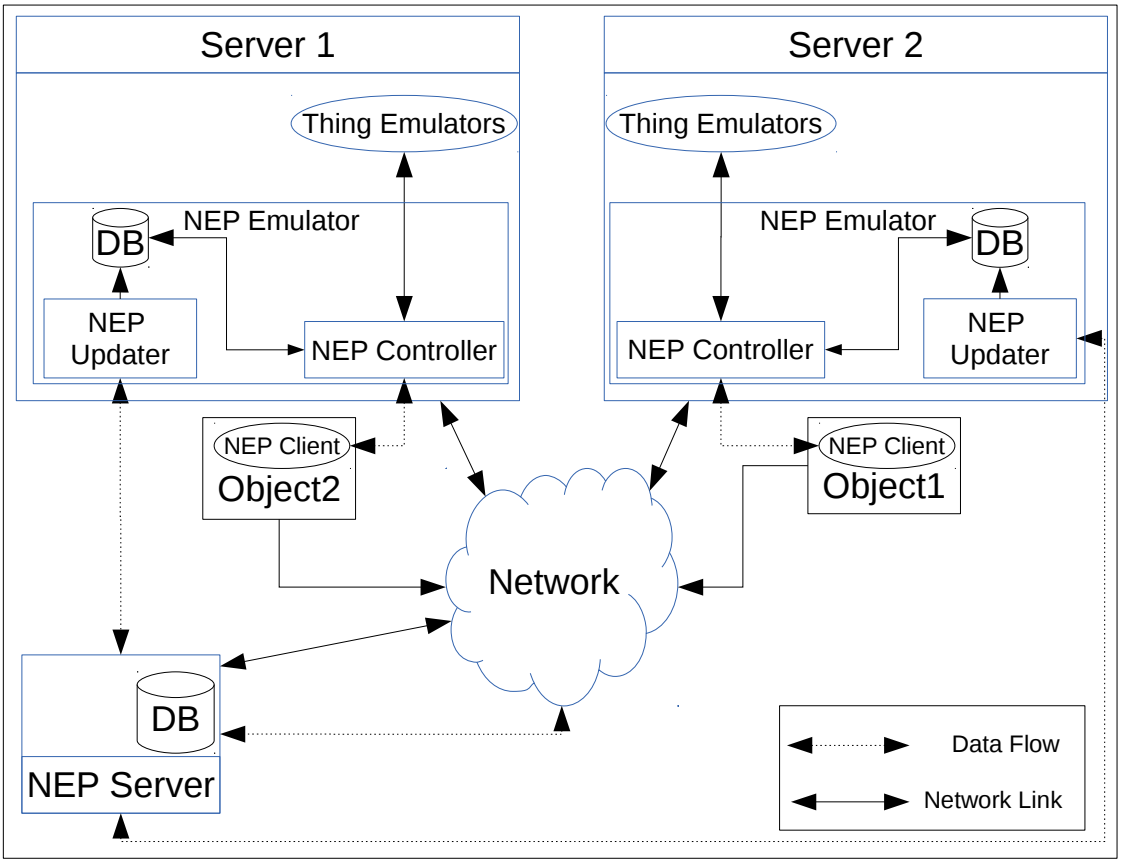

Figure 4.3: IoTaaS NEP Architecture

4.6.2/ NEP EMULATOR

In distributed testing environments, there are servers to communicate with emulators and devices. Each testing server should have a NEP Emulator module installed on it. NEP Emulator consists of two main subcomponents: NEP Controller and NEP Updater. In the following, we are presenting these two subcomponents.

4.6.2.1/ NEP CONTROLLER

NEP Controller is responsible of the following tasks:

- Building and updating local database of network statistics.
- Conducting periodic tests to collect network statistics between the server and each NEP Controllers and NEP Clients individually.
- Checking the local database to retrieve the network scenarios which should be applied.
- Applying network scenarios on the application traffic exchanged with other devices.

NEP Controller can use static scenarios, or it can control the traffic depending on a predefined propagation model. For performance reasons, NEP Controller should be written

using a programming language capable of accessing the network drivers directly without a mediator system.

4.6.2.2/ NEP UPDATER

NEP Updater maintains the connection with the NEP Server. The data exchanged between the two components consists of:

- To NEP Server: The network statistics which are collected by NEP Controller.
- From NEP Server: The network scenarios which should be added to the local database in the NEP Emulator.

NEP Updater doesn't need to direct access to the network, so it could be developed using any suitable programming language.

4.6.3/ NEP CLIENT

The NEP Emulator is responsible for many tasks and it needs direct access to the network. These tasks could increase device power consumption, use more memory and keep the CPU busy. Therefore it would overload the small devices (things). NEP Client can be used in these devices. It doesn't initiate any test or communication with other NEP entities. NEP Client's function is to reply to the requests from the NEP controller. This request/response exchange is used to calculate the network performance between the device and the NEP Emulator.

4.6.4/ NEP SCENARIO

It defines the network characteristics which should be applied by NEP Controller. Each scenario should specify the time period of execution, in addition to designate the traffic which would be manipulated depending on network address, application port, application signature, transport protocol, etc. The scenario can be one of many forms, such as:

- **Static Form:** In this form, the scenario would define the general characteristics for the network, such as: bandwidth, error rate,etc. The NEP Controller will apply these characteristics statistically on the designated traffic, during the specific time period, regardless of the network statistics in the local database.
- **Dynamic Form:** The scenario defines the performance needed for the designated traffic. The NEP Controllers use the network statistics to delay certain packets or to give them a priority to achieve the performance defined in the scenario. Propagation models could be used here in order to simulate certain network behaviors.

In static forms, the scenario could be applied on outgoing/incoming packets. It isn't the case for the dynamic forms, where delaying packets should be applied on incoming packets, while raising the packet priority could be applied on outgoing/incoming packets. Developers and testers can define new forms or subcategories for these forms.

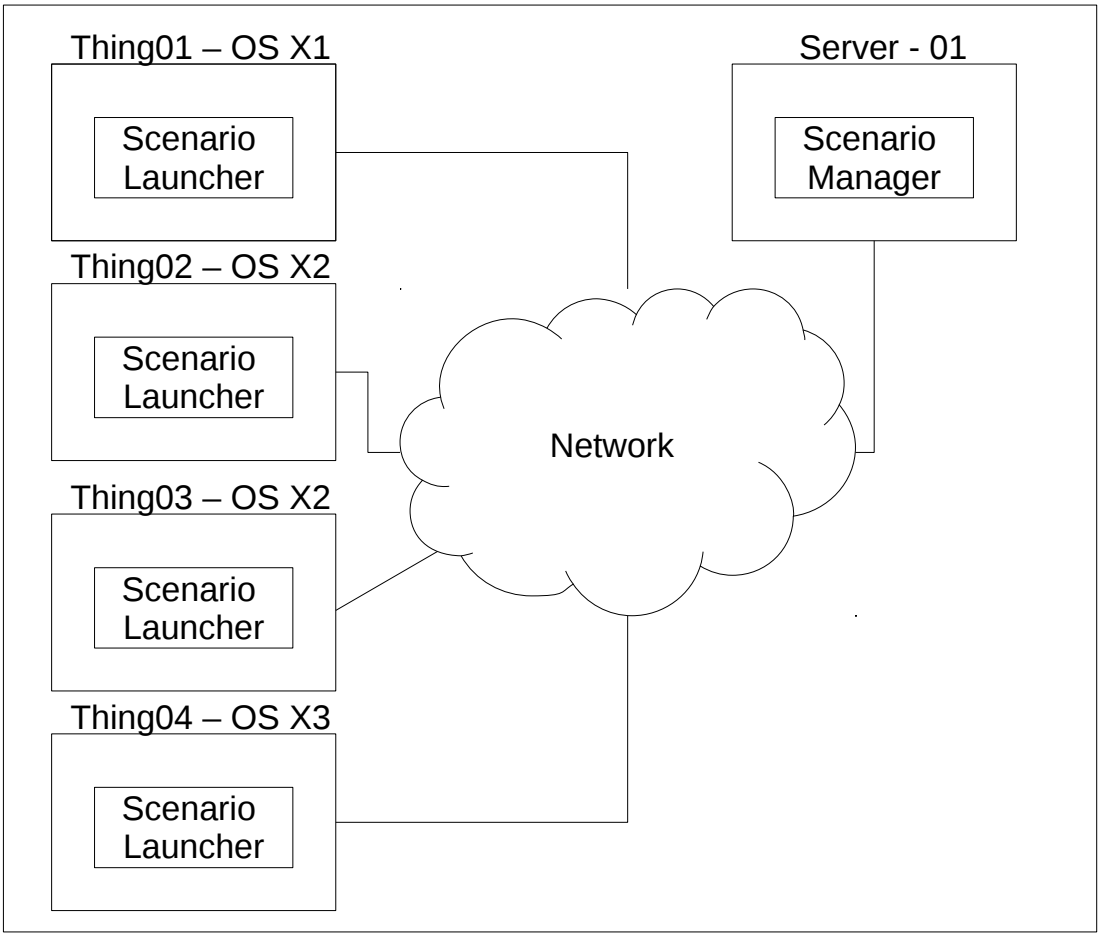

Figure 4.4: IoTaaS - Scenario's General Architecture

4.7/ SCENARIOS

A scenario is an outline or model of an expected or supposed sequence of events. For any application, there is a certain number of sequences of actions and events. This number varies depending on the application itself. For some applications, it could be infinite number of scenarios.
For standalone applications, it is possible to prepare a testing scenario in order to test some (or all) sequences of events. The same application could have different versions for same operating systems, or different versions for different operating systems. In this case, each version should have its own scenario.
It is getting much more complicated when talking about testing network-based applications, where events depend on results or actions from other applications on the same device or on a different device. In nowadays applications, most of testing scenarios depend on user input. In IoT applications, the majority of events and actions take place between devices and applications directly (user input isn't needed). A standalone-based testing wouldn't be sufficient to cover all possible scenarios for a system, nor for an application.
The proposed solution is client-server-based scenario module (Figure 4.4). The module consists of three main components: Scenario Files, Scenario Manager (Server) and

Scenario Launcher (Client). The main idea of this module is to separate the actions on a given framework from the sequence of events (scenario) for an application. In other words, the same scenario file (XML file in this design) could be sent to different systems in order to run the same sequence of events. In the following subsections, we discuss in details these three components.

4.7.1/ SCENARIO FILES

Scenarios should be written and described using a cross-platform machine-readable language, such as XML. The following attributes define a general ***Trigger*** used in this file (repeatedly):

- **Type:** In general, there are 5 different actions which could be used as a trigger:
  1. **counter:** a counter is set to 0. The countdown will be initialized starting from the integer value in the ***Value*** attribute.
  2. **time:** reaching the time specified in the ***Value*** attribute. Time format is YYYY-MM-DDThh:mm:ss.sTZD (ISO 8601).
  3. **action:** an action is triggered. The ***Value*** attribute should contain this action name or ID.
  4. **delay:** a countdown timer has reached zero. The ***Value*** would have this time in the format PnYnMnDTnHnMnS (ISO 8601).
  5. **never:** infinity.
- **Value:** This attribute could have different type of values depending on ***Type***'s attribute.

In the following, we list the main tags and attributes in this file (Code 4.7.1):

- **Scenario**: It consists of the attributes which define the scenario in general. This is a mandatory tag. It has 5 main attributes:
  - **ID:** It is the scenario identifier. Each scenario should have a unique identifier. This identifier is used to collect the results and to prepare the statistics.
  - **Name:** It is human-readable name, it could be used to distinguish different scenarios.
  - **Type:** In case of having different categories of scenarios, this attribute could be used to indicate the type.
- **Start**: This is an optional tag. By default, the scenario will be executed directly unless of using this tag in order to fix a condition. It has a trigger **StartTrigger (Type/Value)** which is used to define when the scenario should be started.
- **Loop**: A scenario could be executed more than once. This is an optional tag (in case of its absence, scenario will be executed only once). The following attributes define loop settings:
  - **Enabled:** This is a boolean attribute determining whether the scenario will be applied once (=0), or it would be repeated (=1).

  - **StopLoopTrigger (Type/Value):** It is a trigger used to define when the scenario should be stopped.

- **End**: This is an optional tag. By default, the scenario will be stopped directly after the last action is applied (loop isn't defined). It has a trigger **StopTrigger (Type/Value)** which is used to define when the scenario should be stopped. In case of using this tag, the scenario will be stopped as soon as the trigger is hit, even in the middle of execution of the scenario. This tag should have a priority over all other events.

- **action**: Each scenario consists of a sequence of actions. Each action has 4 main attributes

  - **ID:** A unique identifier for each action is required. This ID is usually given by the framework.
  - **Name:** It could be the same as the ID. It could be a human-readable name.
  - **Command:** In case that the action isn't predefined, a command could be provided in order to be executed.

  An action could have parameter(s). Each parameter has the following attributes:

  - **ID:** This is a local identifier. Each parameter of the same action should have its own ID.
  - **Name:** It is a human-readable name. It can have the same value as ID.
  - **Type:** It defines the type of the parameter. The type could be any simple type, such as: integer,short, float, string, etc. The complex types could be: file or server, which means that it should read the parameter from a file, or it should contact an external server.
  - **Value**: In case of simple types, this attribute contains the value of this parameter. Otherwise, this attribute will have file path or server URL. This attribute could be omitted in case of having an array of values.

  A parameter could have an array of values. Each value is defined as follows:

  - **ID:** A unique identifier for each value is required.
  - **Name:** It is a human-readable name. It can have the same value as ID.
  - **Value:** In case of simple types, this attribute contains the value of this parameter. Otherwise, this attribute will have file path or server URL.

Other tags and attributes could be defined. It is important to mention that the same file should be used for all platforms.

Code 4.7.1 - Scenario File Architecture (XML)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<scenario id="" name="" type="">
  <start startTriggerType="" startTriggerType=""/>
  <loop enabled="0/1" stopLoopTriggerType="" stopLoopTriggerValue=""/>
  <end type="" value=""/>
  <actions>
     <action id="" name="" command="">
           <parameters>
                 <parameter id="" name="" type="" value=""/>
                 <parameter id="" name="" type="" >
                    <values>
                         <value id="" name="" value=""/>
                         <value id="" name="" value=""/>
                    </values>
                 </parameter>
           </parameters>
     </action>
  </actions>
</scenario>
```

4.7.2/ SCENARIO MANAGER

This component performs all functions on server's side. Scenario Manager could be written in any programing language and it can run on any operating system. There are four main subcomponents which are responsible of performing server's functions (Figure 4.5):

- **Dispatcher:** The principal function of this subcomponent is to maintain connections with scenario launchers (clients) in order to exchange data. The data exchanged is divided into four types:
  - **Scenario Files:** Dispatcher sends to launcher a scenario file (if exists). Scenario files could be stored directly on the server, separated database or on external storage server. Each scenario file should be categorized according to: the application, the scenario and the client.
  - **Parameters:** Scenario Launchers would need certain parameters from the scenario manager. Scenario Launchers send a request, the dispatcher sends back the requested parameter.
  - **Dynamic Actions:** Dispatcher can send certain actions to certain clients. These actions would be obtained from the application analyzer.
  - **Results:** A Scenario Launcher sends the results of executing certain scenario (or certain dynamic actions) as soon as it has a connection with its own scenario manager. The results should be categorized according to: the application, the scenario and the client.

  Dispatcher and Scenario Launchers can communicate using a cross-platform distributed application protocol, such as: Web Services.

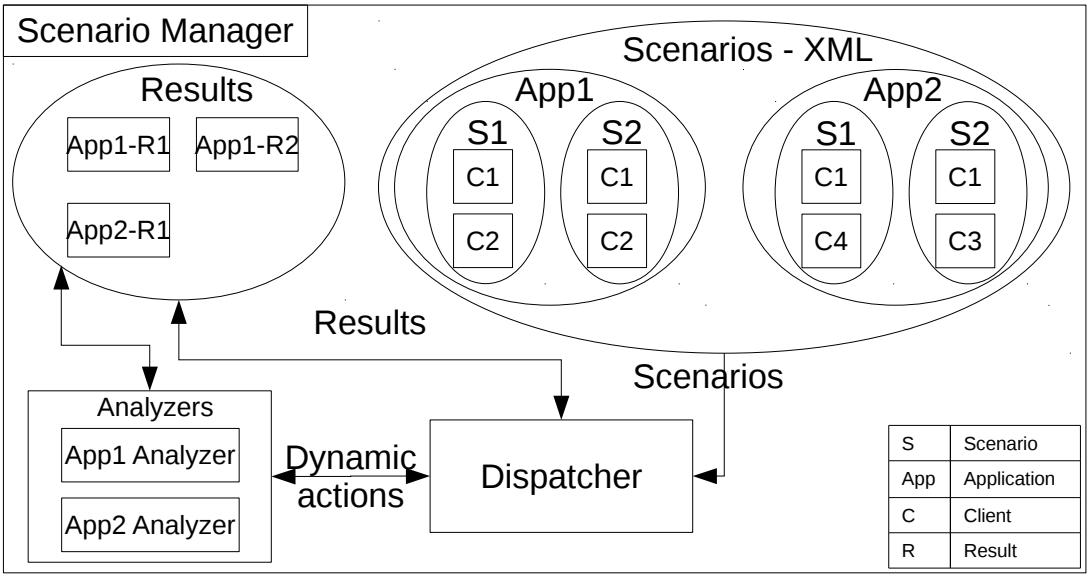

Figure 4.5: IoTaaS - Scenario Manager

- **Results:** Results are all data received from Scenario Launchers as a result of executing a scenario or a dynamic action. Dispatcher receives these results and store them directly on the server, dedicated database or on an external server.
- **Analyzers:** Scenario Manager could have analyzers for certain application. An analyzer would analyze the results received from the Scenario Launchers. Depending on this analyze, it could send direct actions to certain clients through the Dispatcher. Analyzers can be a good tool in order to give dynamicity to a testing scenario. Analyzer output could be added as results.

Several Scenario Managers could be used in a given environment. Scenario Launchers could be configured to have a backup Scenario Manager. Other solutions could be provided using load-balancers techniques.

4.7.3/ SCENARIO LAUNCHER

Scenario Launcher is the component on the client device that perform testing and evaluating functions. This component varies depending on the device's framework. Each framework (or each version of framework) should have its own version of Scenario Launcher, it should be developed using a programing language supported by the framework, such as Java for Android and Objective C for OSX and iOS, etc.
There are four main subcomponents in any Scenario Launcher. In the following, we present these subcomponents (Figure 4.6):

- **Connector:** This subcomponent is responsible of maintaining the connection with a Scenario Manager. This connection is used in order to exchange following types of data:
  - **Scenario Files:** Scenario Launcher receives the scenario file(s) from a Scenario Dispatcher. It stores Scenario File(s) locally. Each scenario file should

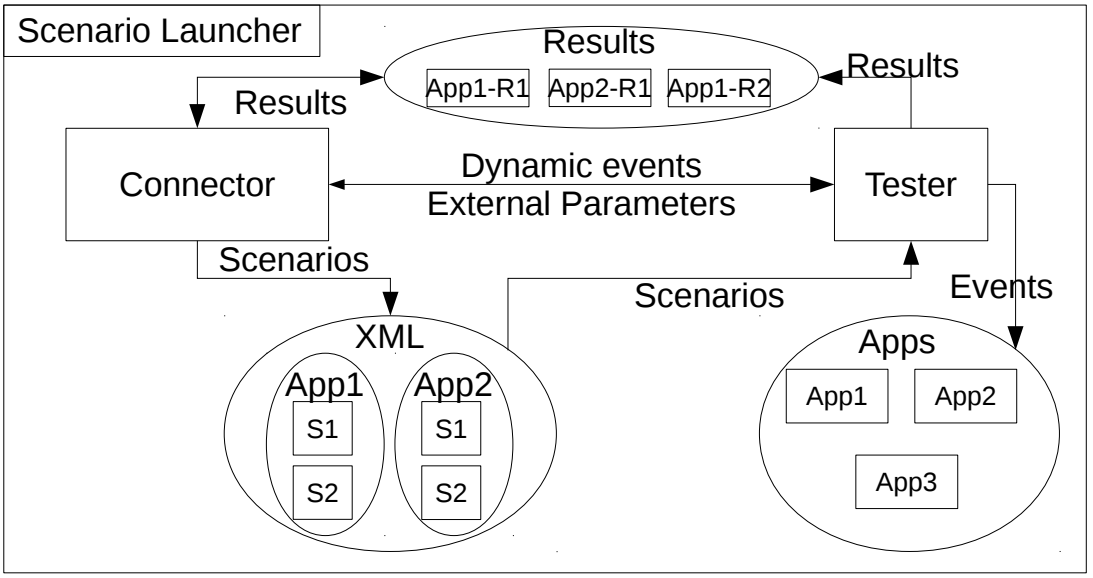

Figure 4.6: IoTaaS - Scenario Launcher

be categorized according to: the application and the scenario.

- **Parameters:** When Scenario Launcher needs certain parameters from the scenario manager. Scenario Launcher sends a request, the dispatcher sends back the requested parameter.
- **Dynamic Actions:** When the Scenario Launcher receives a dynamic action from the Scenario Manager, this action will have a priority over all other actions.
- **Results:** The results of scenario executed (or dynamic actions), they will be sent directly or from the local storage.

Connector could be considered as the bridge that connects between the Scenario Manager and the other subcomponents in the Scenario Launcher.

- **Tester:** It is responsible of executing the scenarios which have been received from the Scenario Manager. The tester should have the permissions to:
  - inject events into other applications on the same device.
  - listen to events dispatched by other applications or by the operating system itself.
  - read/write files from/to the local storage.
  - set/get operating system parameters.

  Tester should be multi-threaded application in order to execute multiple scenarios at the same time.

- **Results:** In case of having a connection with the Scenario Manager, the Scenario Launcher sends the results of executing certain scenario (or certain dynamic actions) directly, otherwise the results would be stored locally awaiting for a connection with a Scenario Manager. The results should be categorized according to the application and the scenario.

- **Applications (Apps):** On the same device, one application at least should be the target of testing and evaluating. The same application would have different versions depending on the device's framework. The testing application(s) should permit other applications to inject events.

For security reasons, and since the Scenario Launcher would have access to all the functions on the device, Scenario Launcher applications should run only in special mode (ex. debugging mode). This mode should be turned on/off manually by the user or the tester. This could protect normal devices from being hacked by using these privileges an permissions.

4.8/ CLOUD

An IoTaaS could consists of several servers. These servers could be organized in a tree-based structure, ad hoc structure or mesh structure. The structure is logical and it has no effect on how the servers are physically connected to the network(s). In case of an IoTaaS with only one server, this component would not exist. The Cloud manager might consists of 5 subcomponents. The first two subcomponents are *Parent* and *Childservers* subcomponents which exist only in a tree-based structure. *Peers* subcomponent exists in case of ad hoc or mesh structure. *Clients* subcomponent exists in case of commercializing the IoTaaS services. The *Inter-IoTaaSs Communication* maintains the connection with the other IoTaaSs. In the following, we are going to discuss each one of these subcomponents and their functionality.

- **Parent:** In a tree-based structure, every server (except for the root) would have a Parent server. In this case, the server is called a "Childserver". The Parent subcomponent is responsible of:
  - Maintaining the connection with the parent.
  - Transferring the requests originated from the server itself and received from the childservers to the parent.
  - Transferring the responses received from the parent to the server which has initiated the request.
  - Sending cloud updates to its childservers.
- **Childservers:** In a tree-based structure, a server could have childservers, and in this case it is called a Parent server. This subcomponent manages the childservers which are connected to this server. The childserver is responsible of:
  - Maintaining connections with the childservers.
  - Transferring requests originated from the server itself and received from the childservers to the parent.
  - Transferring responses received from the parent to the server which has initiated the request.
  - Sending cloud updates to its childservers.
- **Peers:** In ad hoc and mesh structures, all servers would be peers. That means there is no parent and childservers. Each peer could be connected to several other peers. In this case, the peer is responsible of:

  - Maintaining connections with other peers.
  - Running a routing protocol, such as Optimized Link State Routing Protocol (OLSR), Ad hoc On-demand Distance Vector(AODV), etc.
  - Maintaining the servers' topology.
  - Forwarding requests and responses to their respective destinations.

- **Clients:** Any IoTaaS should provide an interface for the users so they can test and evaluate their systems. An Authentication, authorization, and accounting (AAA) system should be provided in order to give the clients the needed access only to the reserved components and things. The system should be able to serve several clients at the same time without affecting each other. The client interface could be provided using a desktop application or web application.
- **Inter-IoTaaSs Communication:** It is expected to have specialized IoTaaS architectures. For example, an IoTaaS for medical applications, another one for transportation, etc. The IoTaaS should be able to cooperate with other IoTaaSs in order to provide the clients with one environment in order to test and evaluate their systems. Each IoTaaS should implement a unified interface to communicate with the other IoTaaSs.

4.9/ SERVERS

The term *Server* refers to the system (software) that runs on the server, and not to the hardware, nor to the operating system. The operating system could be any that is capable of running network-based applications. The IoTaaS server is the central component of the IoTaaS architecture, it is the system which manages all other components of an IoTaaS architecture. The server could be developed using any programming language which supports multithreading and distributed techniques. Any server implementation should have three main components:

- **Daemon:** It is the subcomponent which manages the server, provides the interfaces which should be implemented by the other subcomponents and bridges all the other subcomponents.
- **User Interface (UI):** The user interface should simplify managing the subcomponents. The UI could be a Graphical User Interface (GUI) or a Command Line Interface (CLI). The interface which is mentioned here isn't the same as clients' one. This interface should be accessed only by the IoTaaS managers.
- **Reporter:** This subcomponent manages the log generated by other subcomponents. Reporter should be able to analyze the logging data and give the results. This subcomponent should work depending on dynamic rules in order to be customized according to the given environment and the testing scenarios. The data should be organized in exportable form in order to give the possibility to analyze it using external tools.

For the other components, each IoTaaS implementation can decide which of these managers would be integrated in the server. For example, whether the NEP would be necessary in this environment or not, whether the gateways would be integrated in the same server or they will be managed by another service, etc.

4.10/ USER

The end user is the ultimate judge of any technology. Any testing and evaluation architecture which would ignore this fact would fail shortly. Unfortunately, measuring users' satisfaction can be difficult but it isn't impossible. An IoTaaS should provide the interfaces to measure the users' satisfaction by allowing human testers to give their opinion after interacting with the IoT system and devices. Users' satisfaction could be predicted but it might not be fully automated.

4.11/ CONCLUSION

IoTaaS which is an architecture for testing IoT has been provided. It describes the main components which could be existed in any IoT testing environment. The architecture could be used in smart grid, smart city, WSN. The main idea of this architecture is to enable communication and service exchange between testing environments of different IoT sub-domains which lead to the possibility to have one distributed environment for testing several IoT applications. The architecture consists of seven components which cover managing physical/virtual *things* and objects, communicating with entities directly and indirectly, emulating certain types of networks, generating and managing testing scenarios, integrating human experiences and managing connections with other IoTaaS-based testing environment.
In chapter 5, we present Cloud Environment for Mobile Application Testing (CEMAT) which is our pilot IoTaaS-based project to test mobile application.

5

IoTaaS Pilot Implementation

5.1/ Introduction

IoTaaS main architecture enables developer and tester to deal with all types of things (devices). IoTaaS is an abstract architecture which could be implemented for different environments. In this chapter, we are presenting the pilot implementation of IoTaaS under the code name **CEMAT (Cloud Environment for Mobile Application Testing)** which focuses on mobile devices and their connected objects. The goal of this implementation is to prove the possibility of realizing IoTaaS architecture. CEMAT provides a distributed environment for testing and evaluating mobile applications. It organizes the servers in a tree structure. Each server might have several mobile phones and several emulators. CEMAT enables developers and testers to test and evaluate different scenarios of their applications on many mobile devices and emulators at the same time. The rest of this chapter is organized as follows: Section 5.2 discusses the selection criteria of a mobile operating system to be the first to be supported in CEMAT. Section 5.3 describes the structure of CEMAT server. Section 5.4 presents the experiments which have been carried out to test CEMAT functions.

5.2/ Mobile Operating System

The predecessors of today's smartphones are yesterday's personal digital assistants (PDA) and mobile phones. Mobile phones gave consumers the convenience of having a phone wherever they went, while PDA's gave consumers the ability to easily carry around all of their personal information (address book, calendar, note pad, etc.) and have access to their email or other data. The smartphone began as a combination of these two devices, giving consumers the convenience of one device that performed both functions [Cromar, 2010]. In 1992, IBM started developing the first smartphone known as "Simon". Then in 1996, Nokia came out with the Nokia 9000 which included all the functionality of a PDA and a phone. The first iPhone was released on June 29, 2007 with Apple's iOS mobile operating system. On November 5, 2007, the Open Handset Alliance, a consortium of technology companies including Google, device manufacturers such as HTC, Sony and Samsung, wireless carriers such as Sprint Nextel and T-Mobile, and chipset makers such as Qualcomm and Texas Instruments, unveiled itself, with a goal to develop open standards for mobile devices. Android was unveiled as its first product, a mobile device platform built on the Linux kernel version 2.6.25 [Alliance, 2015b]. Windows Phone

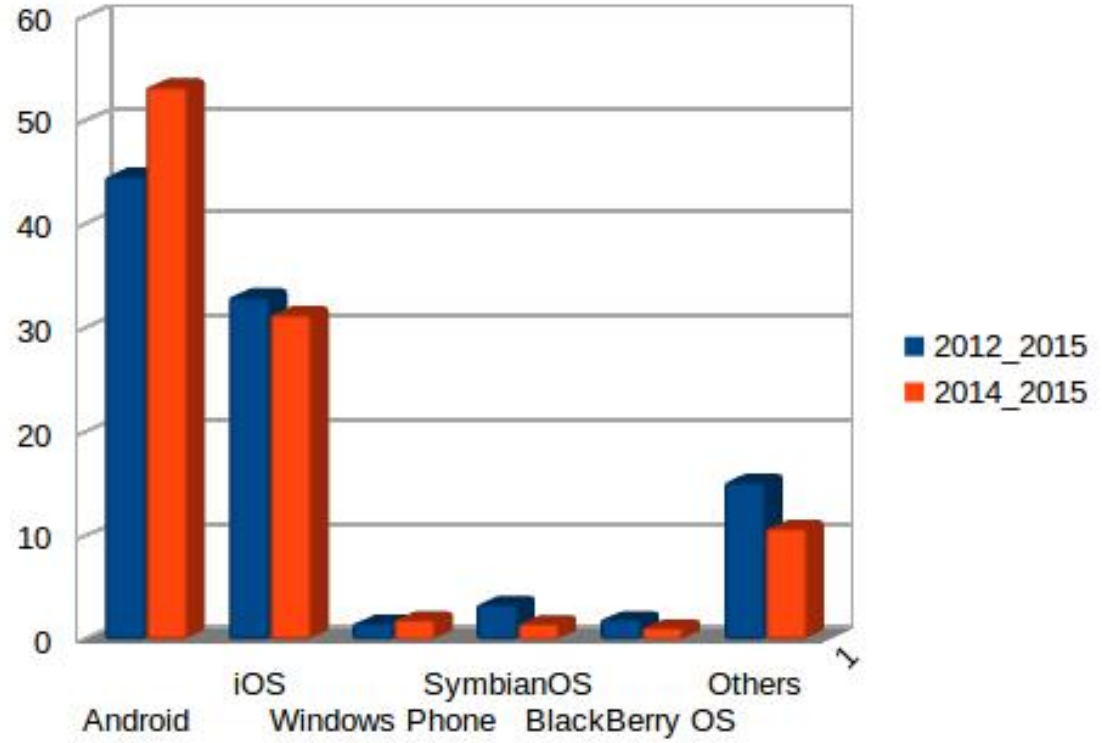

Figure 5.1: Top 5 Mobile & Tablet Operating Systems

operating system has been developed by Microsoft for smartphones as the replacement successor to Windows Mobile. BlackBerry OS is a mobile operating system developed by BlackBerry Ltd for its BlackBerry smartphones. Symbian is an open-source platform developed by Symbian Foundation in 2009. Symbian was used by many major mobile phone brands, mainly by Nokia. The last major update for Symbian was in October 2012 by Nokia. Many other mobile operating systems have been developed such as: Bada, Fire OS by Amazon, MIUI by Xiaomi Tech, Flyme OS is develop by Meizu, Firefox OS is from Mozilla, Sailfish OS is from Jolla, Tizen which is hosted by the Linux Foundation and guided by a Technical Steering Group composed of Intel and Samsung and Ubuntu Touch OS is from Canonical Ltd.
Figure 5.1 shows market statistics for the top 5 mobile & tablet operating systems used in world. It compares mobile market statistics for the last three years (2012-2015) with statistics for the last year (2014-2015) [StatCounter, 2015]. Statistics show that Android dominates the market since 2012. During the last year, Android kept gaining more market share and it acquired about 53% of the market mobile operating systems. iOS acquired about 31% of the market which means that Android and iOS control about 84% of the market. In the following, we provide a brief list of the advantages of Android Android OS compared to iOS:

- Android's source code is released under open source licenses.
- It has specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear).
- It is supported by a wide range of mobile device manufacturers.

In addition to that, a specialized version of Android for IoT applications is under development. For all these reasons, Android was the best choice for CEMAT.
The Android platform has been deployed across a wide range of devices, such as mobile phones, tablets, watches, televisions and cars. Nowadays, there are hundreds of devices with Android operating system. Not all of these devices run the official versions which are developed and released by Google.
Emulator is one of the most important tools for testing and evaluation mobile applications. The emulator is a virtual mobile device that runs on a computer. The emulator enables developers to develop and test Android applications without using a physical device. Android provides an emulator as part of its Software Development Kit (SDK). Android emulators are customizable tools, developers can change the parameters in order to simulate certain device (e.g. screen sizes and resolutions, underlying hardware configurations, SD card size, RAM size etc.). Developers can simulate calling and SMS messaging between two emulator instances, in addition of running location based services (using "dummy" GPS coordinates). There are many limitations in android emulators, such as: the limited performance comparing to read devices, the inability to determine device state and network state, the lack of Wi-Fi and bluetooth support.
Many independent projects are working to develop emulation environment for android operating system. The most important projects are Genymotion, Andy, BlueStacks, ARC Welder, ARChon Custom Runtime and AMI DuOS. These tools can be categorized into two main categories. The first category contains the tools that run as applications on a desktop operating system (such as : Windows, Mac and linux), such as Genymotion and Andy. The tools in the second category run as a plug-in extension in an Internet browser, such as ARC Welder. Other projects are working to port android versions as complete operating system in order to be installed on real computers, such as: Android-x86 and Android-IA.
Android-x86 is an open source project licensed under Apache Public License 2.0 to port Android open source project to x86 platform [android x86, 2015]. Different android versions have been ported to several x86-based computers, such as: ASUS Eee PCs/Laptops, Viewsonic Viewpad 10, Dell Inspiron Mini Duo, Samsung Q1U, Viliv S5, Lenovo ThinkPad x61 Tablet. Android-x86 provides a complete, compilable and workable source tree which simplifies modifying and recompiling Android system files.

5.3/ SERVER ARCHITECTURE

CEMAT server is the main component of the cloud which refers to the application which will handle CEMAT functions. Each CEMAT environment should have at least one server. There is no special requirements for server's hardware. For the server's operating system, the three main choices of operating system (Microsoft, Linux and Mac) could be used. In our implementation, we used Linux Ubuntu Server 14.04.2 LTS - 64 bits. The programming language used is Java, which means that the same application could be deployed on the other operating systems (Windows, Mac). In order to eliminate the need to recompile the application with each change, all the parameters (paths, variables and commands) that could/should be changed according to the system or the environment have been stored in XML files. Figure 5.2 shows the architecture that has been used in the implementation. The components of the architecture will be described in the following subsections.

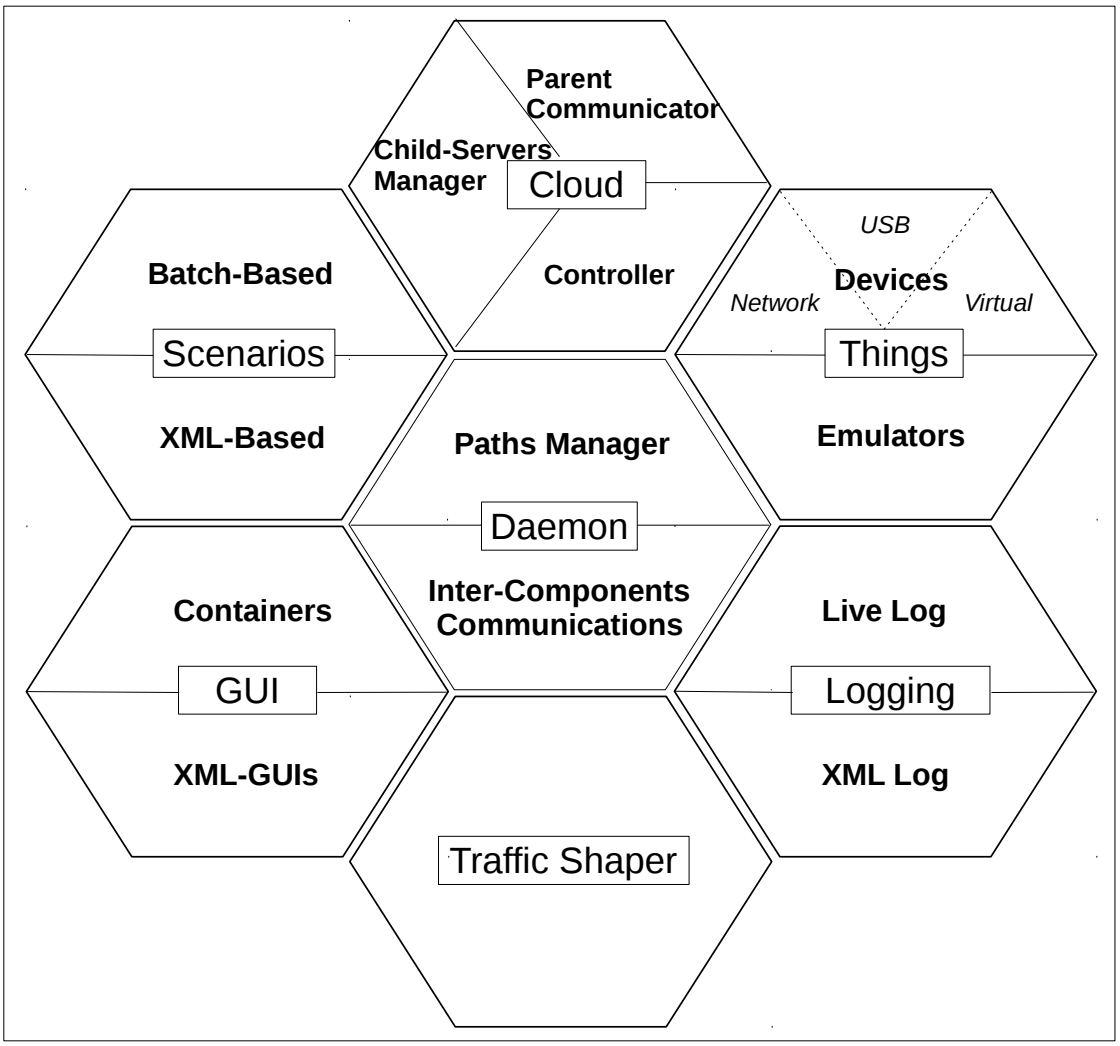

Figure 5.2: CEMAT Server Architecture

5.3.1/ DAEMON

CEMAT Daemon is the central component in CEMAT server architecture which has many responsibilities. Daemon is responsible of maintaining the link between the parameters which are used in the application and the ones in the XML files. For this reason, Daemon distinguish four main types of parameters (Figure 5.3):

- **Paths Parameters:** All paths used in CEMAT could be modified directly in a XML file. Paths could be Other XML file paths, Android SDK tools path or any other paths related to operating system. Paths.xml should be placed in the same directory with CEMAT jar file.
- **Main Parameters:** This file contains all default values that could be used by CEMAT. for example, TCP ports range, default values for integer and string parameters,etc.
- **Cloud Parameters:** They are formed out of three separated XML files. First file contains all parameters related to cloud connections, such as connection timeout, retrying delay, etc. The second file contains local host information, such as: IP

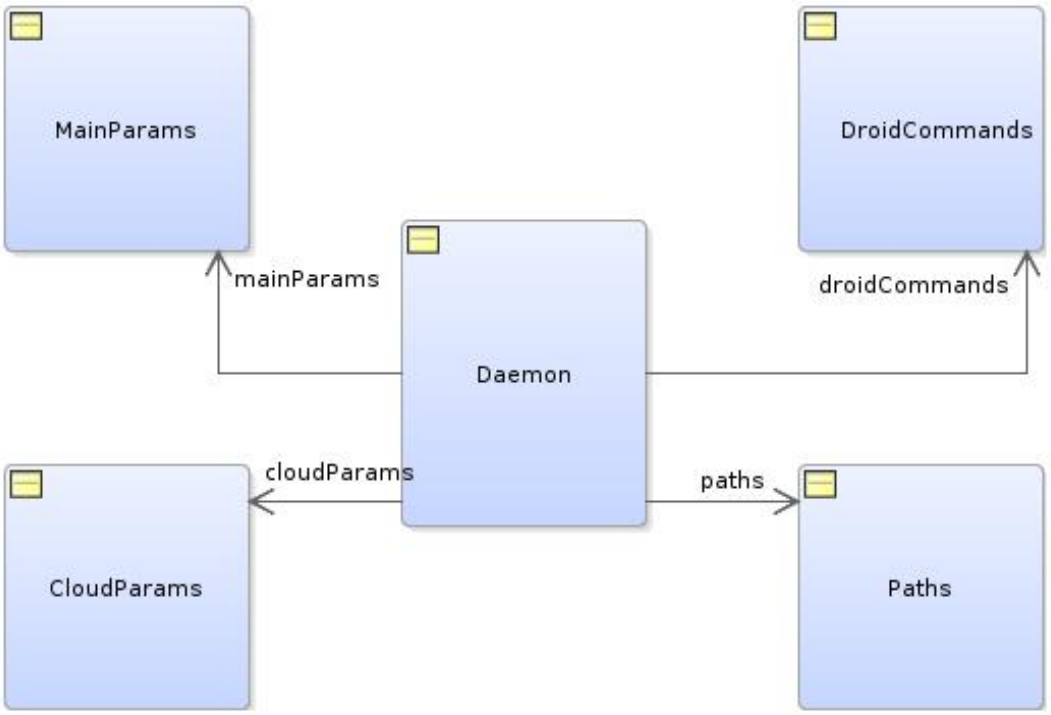

Figure 5.3: CEMAT Daemon - Parameter Types

address, name, etc. The third XML file has the same structure as the second file except that it contains the information related to the parent (Upper CEMAT server).

- **Droid Parameters:** Android depends on a changeable set of variables and commands. These parameters could be changed from a version to another. All these parameters are stored in many XML files in order to keep CEMAT flexible and easily extensible.

CEMAT Daemon provides inter-components communication. In other words, it is the bridge between all other components which provides different functions so one component could request a service from another component.

5.3.2/ THINGS MANAGER

This component has two main subcomponents, devices subcomponent which manages all real/virtual devices and emulators subcomponent which manages the Official Android SDK Emulators. One can communicate (from a computer) with Android Devices and Android Emulators using Android Debug Bridge (ADB) which is part of the official Android SDK. ADB is a client-server tool which consists of three applications (Figure 5.4):

- **ADB Server:** It is a background service which resides on the computer where Android SDK is running. It is the bridge between ADB daemons and ADB clients. It searches for android emulator and android devices by scanning odd-numbered

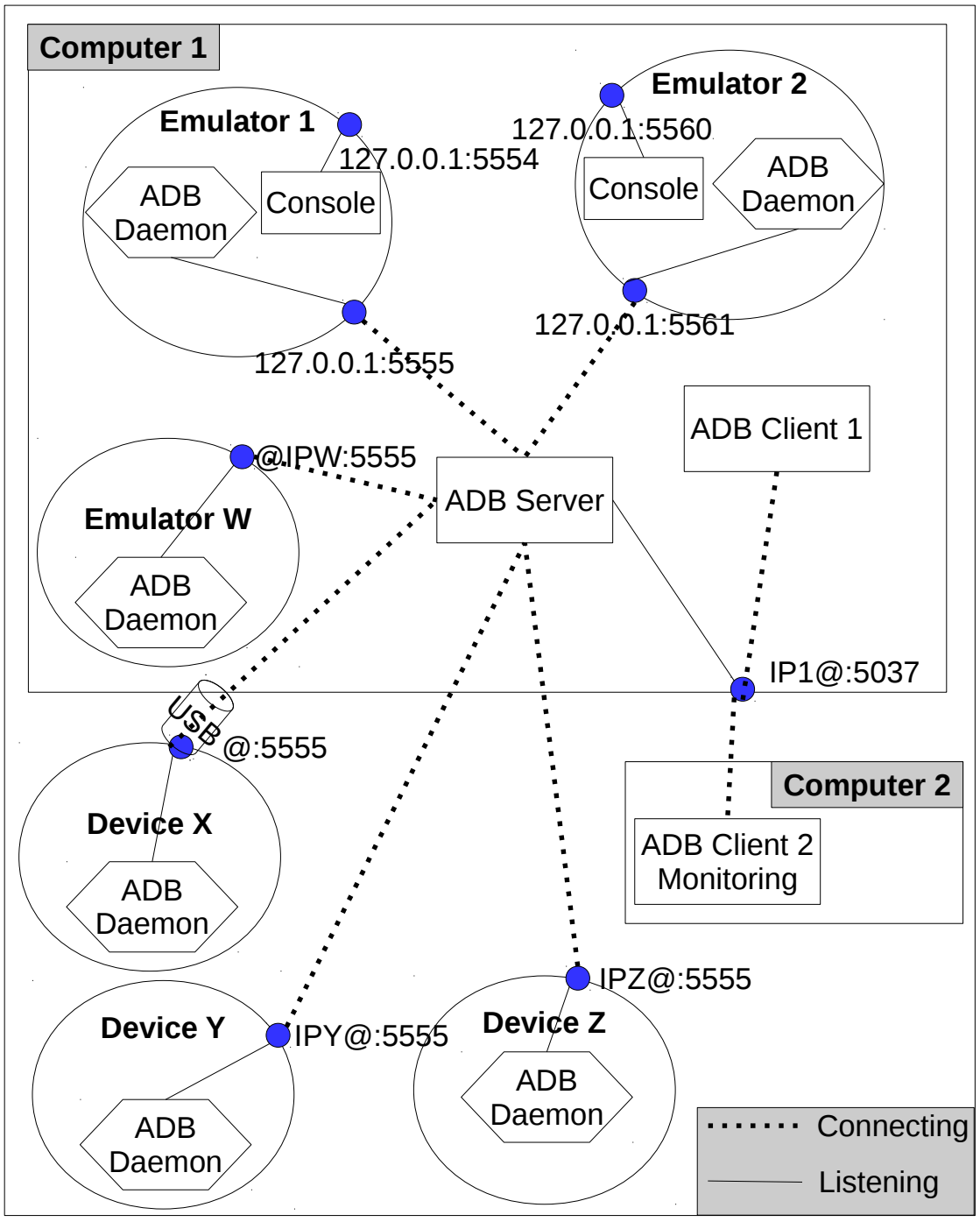

Figure 5.4: Android Debug Bridge (ADB)

ports in the range 5555 to 5585. ADB Server listens to port TCP 5037, ADB Clients can connect to it by connecting to this port.

- **ADB Client:** It is an application which runs on a computer (could be different than the one used for ADB Server) and it is responsible of sending the commands to the ADB server. These commands could be originated from a user or from another tool. The ADB client tries to find a server on its local machine by scanning the port TCP

5037. If no server is listening on this port, the client will launch the service of the ADB Server. The client could be configured to connect to an ADB Server running on another computer by using its IP address.

- **ADB Daemon:** It is a background service running on each android emulator or device. On an emulator, the ADB daemon listens on localhost (127.0.0.1) to odd-numbered ports in the range 5555 to 5585, and it is accessible only from the localhost. On a device, the ADB Daemon listens on the port 5555 and it is accessible from the network outside the device.

In the following, we discuss Android Devices and Android Emulators.

5.3.2.1/ DEVICES

In general, there are three types of entities which are considered as devices (Android's point of view):

- **USB-based Devices:** This category includes all devices which are connected directly using USB, *Emulator X* in Figure 5.4. These devices might be connected to a network (For example: Wi-Fi, 3/4G), but this network won't be used for ADB.
- **Network-based Devices:** The ADB connection in this case is established using an IP address, *Device Y, Device Z* Figure 5.4. The device could be on the same local network as the server or on any other WAN network, the only condition is to be accessible from the server. This type of devices simplifies testing and evaluating network-based and GNSS-based applications in a real environment outside testing laboratory. This category also includes devices which run Android-x86, Android-IA or any other architecture.
- **Virtual Devices:** Usually, 3rd party emulators are built on virtual machines, such as Oracle VM VirtualBox, *Emulator W* in Figure 5.4. These emulators are treated as network-based devices, not as emulators, even if they were running on the same server.

5.3.2.2/ EMULATORS

The Android emulator is an application that mimics most of the hardware and the software functions. It is based on QEMU which is a generic and open source machine emulator and virtualizer. Each emulator listens on a pair of sequential ports (on localhost). The ports range between 5554 and 5585 which means that maximum 16 emulators can run on the same computer. The number of emulators which can run on the same computer is also related to: the hardware capabilities of the same computer and the Android OS versions used. The emulators are assigned these ports as follows:

- Emulator-1: 5554,5555.
- Emulator-2: 5556,5557.
- ...

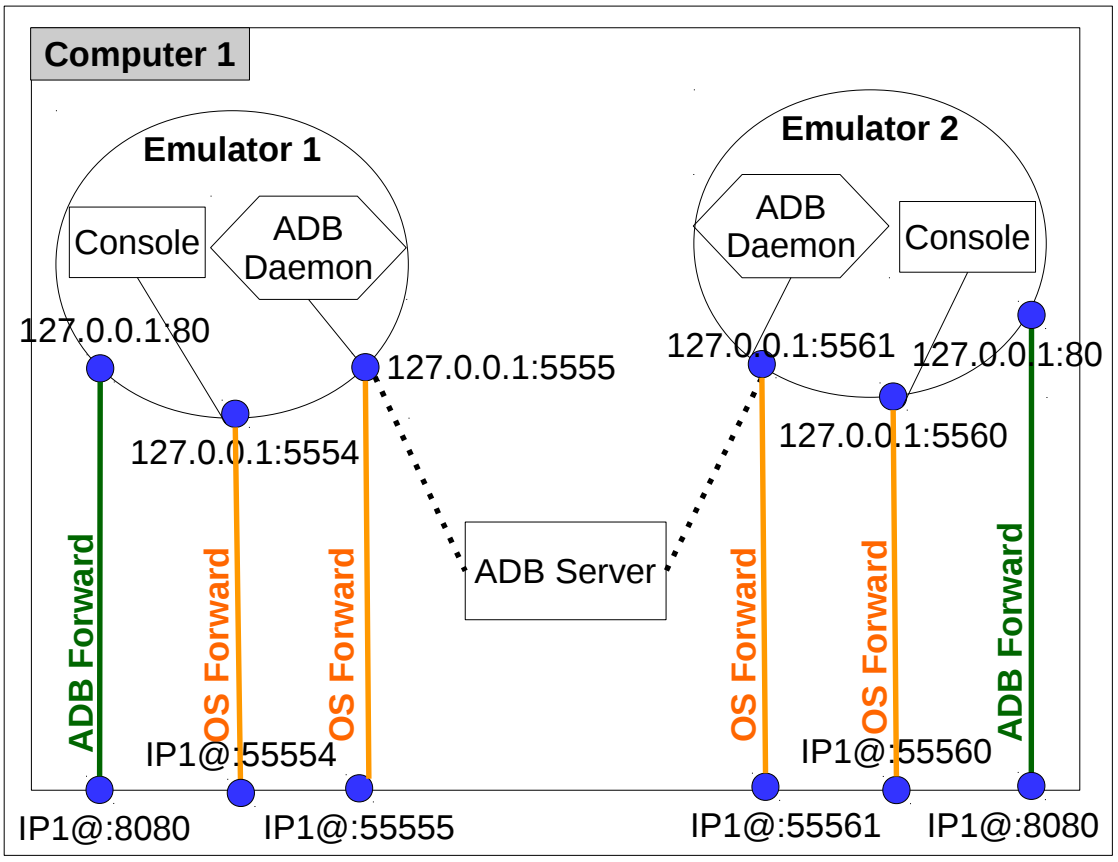

Figure 5.5: Android Emulators - Ports Forwarding

- Emulator-16: 5584,5585.

The two ports are described as follows:

- **Even-numbered port:** This port is used as a console. One can telnet the emulator using this port in order to send commands, such as: setting up redirection, sending a voice call or SMS to another emulator instance, changing the current GEO coordinates, etc.
- **Odd-numbered port:** This port is used by ADB Daemon. The ADB server scans the odd-numbered ports in order to build the list of emulators on the same computer.

The emulators are accessible only from the computer itself. This would cause a problem for many functions. The following list describes the most important affected functions and how the associated problems were solved in CEMAT (Figure 5.5):

- **Management:** The emulators could not be managed by any ADB Server but the one on the same computer. They couldn't be accessed using the console port from the outside. In order to eliminate this limitation, CEMAT bind the pair of ports of each emulator with another pair of ports. The second pair of ports listens on all IP addresses of the computer which means that they will be accessible from the outside. The ports forwarding has been applied on the OS level. For example, port 5555 will be bound with port 55555, port 5556 with port 55556, etc.

- **Network-based Applications:** The testing would be affected in two cases: if the mobile application is acting as a server or if it is a part of peer-to-peer applications. Fortunately, ADB provided a solution to redirect a given port to another operating system port using the command line. In CEMAT, this solution is applied automatically whenever an emulator is added to the cloud.

As we mentioned earlier, emulators can mimic most of the hardware and the software functions but not all of them. In the following, we list some of the most important functions which could not be applied using Android emulators:

- **Wi-Fi:** Emulators can't get direct access to Wi-Fi cards on the hosting computer. So they can't be used to test and evaluate applications which need to deal directly with Wi-Fi frames.
- **Voice Call:** It is a simulated call which doesn't include real data exchange.
- USB and Bluetooth are not supported.
- Determining battery state, network state isn't supported.

5.3.3/ GRAPHICAL USER INTERFACE (GUI)

There are two types of user interfaces, Graphical User Interface (GUI) and Command Line Interface (CLI). CEMAT has been designed to support both of them. As a pilot implementation, an easy and simplified GUI has been designed in order to facilitate managing the CLOUD. Any Java program can be classified as an *Application* or an *Applet*. In the following, we will discuss the differences between Java Applications and Java Applets:

- **Independency:** Java Application is an independent program which runs directly on the Java Runtime Environment (JRE)-enabled operating system. Java Applet is a small application can't run independently, but it requires a Java-enabled browser (such as: Chrome, Firefox, etc) in order to be executed in it.
- **Access Restrictions:** Java Applications have no restrictions to access any data or files available on the system. Applets are treated as untrusted softwares and the only resources they can access are the ones provided by the browser itself.
- **Native Methods:** Java Applications can use native methods (methods created in another programming language). Applets aren't allowed to use such methods.
- **Network Programming:** Applications can get direct access to the network using operating systems libraries, so they can create sockets, analyze packets, etc. Applets are isolated in the browsers and can't use socket-based applications.
- **Bytecode:** Applications need to be installed and configured prior to their usage. They should be upgraded or re-installed with every new version. Applets are loaded directly from the server, which means they don't need to be pre-installed. whenever a new version of the applet is released, it is sufficient to load it on the server in order to be sent to the clients when they request it.

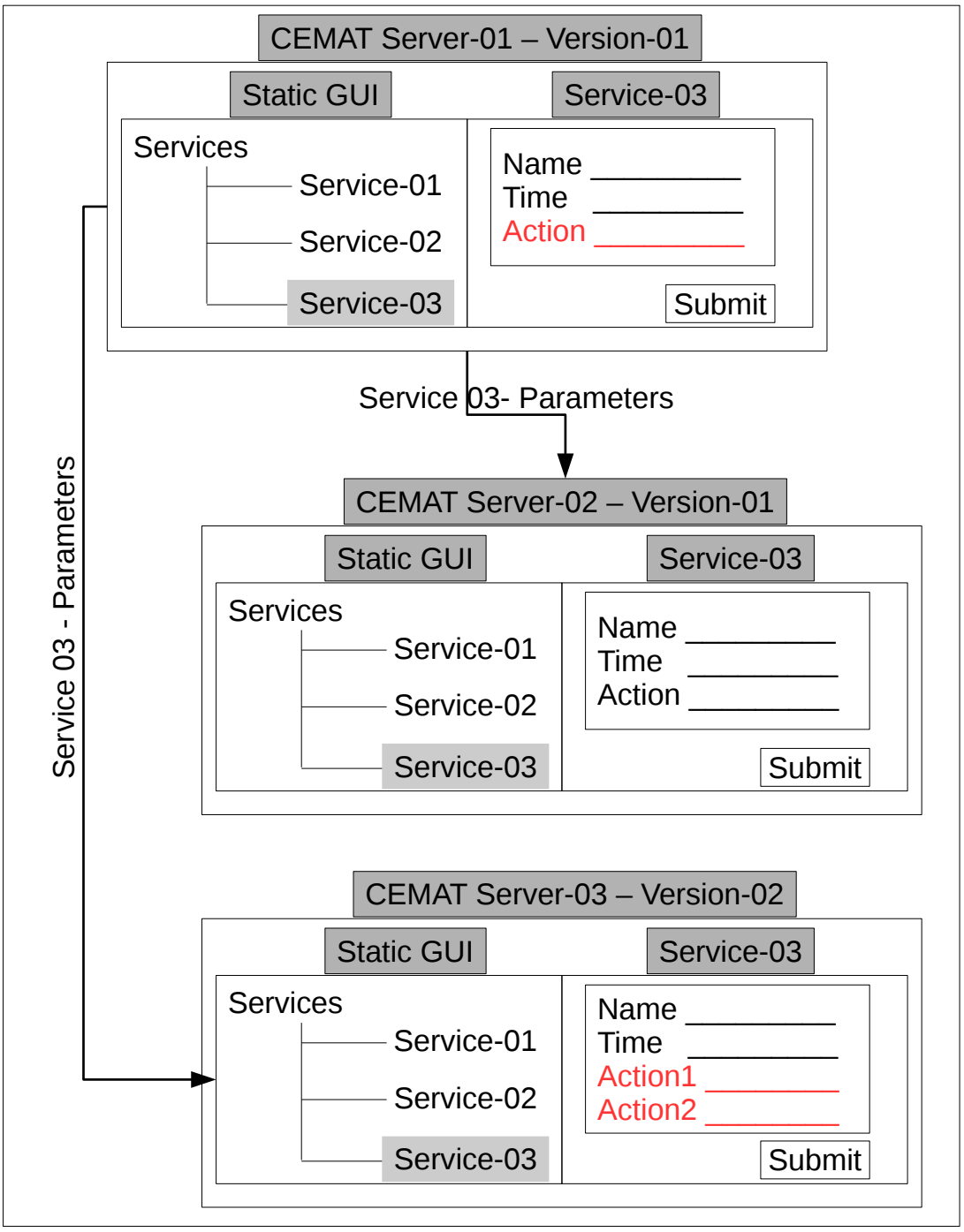

Figure 5.6: Java Application - Distributed Services Scenario

- **GUI:** It is easier to create dynamic and interactive programs using applets. Applications mostly have static GUIs in which all elements would be fixed during the development phase.

Building CEMAT as a web application would simplify exchanging services between CEMAT servers, but at the same time it would add many limitations for using native methods and creating sockets which are required in order to realize other components.

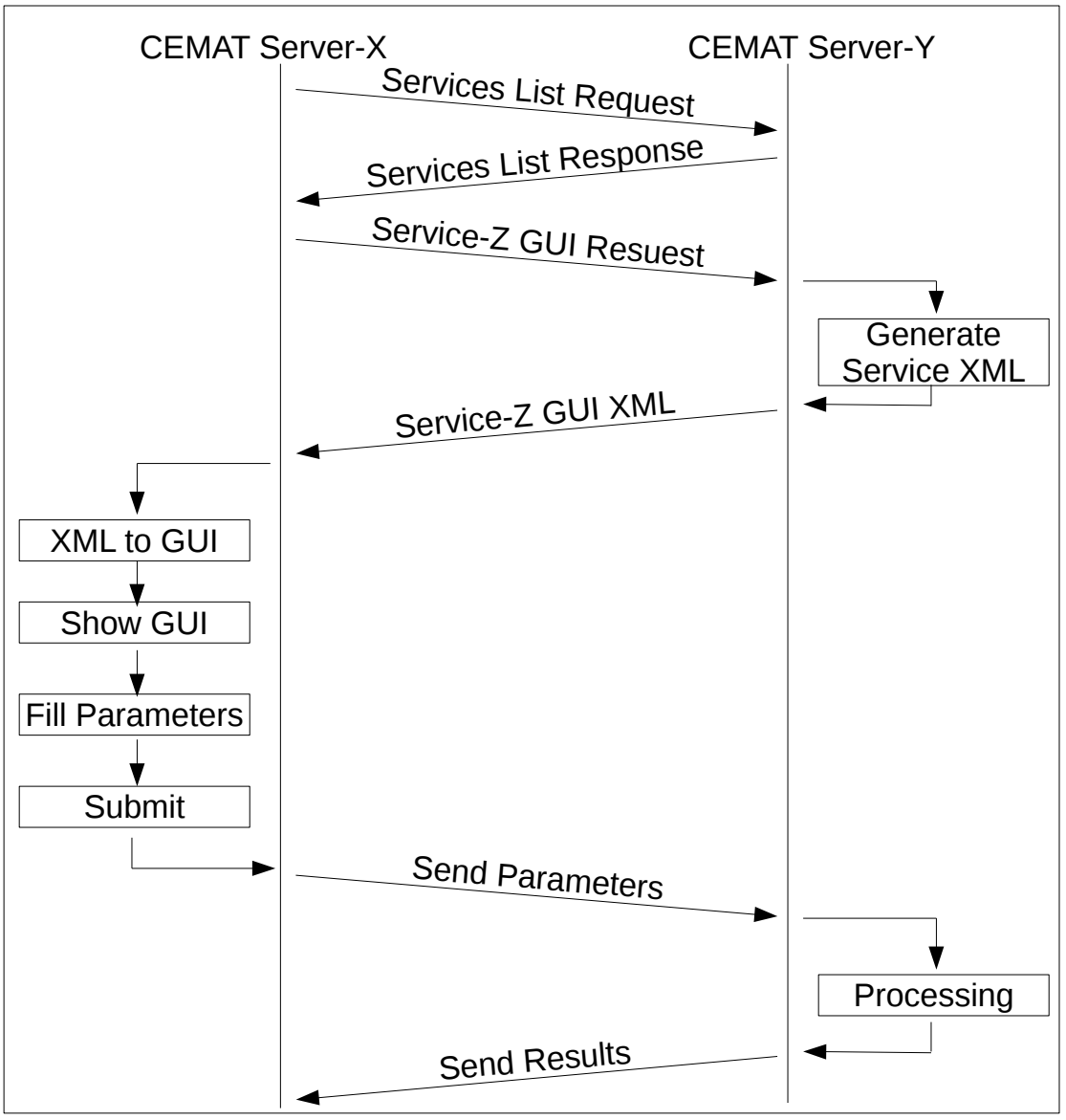

Figure 5.7: CEMAT - XML GUI

On the other hand, Building CEMAT as an application would complicate using some server services by another server which doesn't have the right GUI, either because it doesn't have the service or it has another version of the service.
Figure 5.6 presents an example for this case. The scenario suggests a cloud with three CEMAT servers built as Java applications with local static GUIs. *Server-01* and *Server-02* have software version-1, *Server-03* has been upgraded to version-02. Software version-03 provides the same services except for service-03, for which the fields of the GUI have been modified. In such scenario, the server that requests the service sends the parameters to the server that provides the service. In our case, *Server-01* will be able to use service-03 from *Server-02* because they have the same version, but it can't use service-03 from *Server-03* before getting the right upgrade.
Our solution for this problem is shown in Figure 5.7. The solution depends on having two different GUIs:

- **Containers:** It is the main GUI that will host service GUIs. This container is static and it contains only the general information about services. User entry fields shouldn't be added in this container.
- **XML-GUIs (Services):** Each service should have its own GUI described in an XML format. The XML will be processed using SwiXML library which is a generating engine can parse a XML file and render it into Swing (an API for providing a GUI in Java). The XML file which contains a description is written in a (human/machine)-readable format. SwiXML has a detailed format capable of describing any types of GUIs.

Using XML-based GUI enables Java applications to be similar to applets by loading GUIs from a remote server in real time. Using XML-GUIs simplifies designing and modifying GUIs in applications and applets. In the following, we describes the method used so that Server-X can request Service-Z from Server-Y:

1. Server-X sends a request to Server-Y to get its list of services.
2. Server-Y sends back to Server-X a XML file contains its list of the services.
3. Server-X selects the service which best matches the needs (Service-Z) and sends to Server-Y asking for its GUI.
4. Server-Y generates Service-Z's XML-GUI and sends it back to Server-X.
5. Server-X renders the XML file using SwiXML library and shows the GUI to the user. The user fills the GUI with the needed data and submits the service.
6. Server-X sends the parameters with the GUI to Server-Y.
7. Server-Y processes the parameters received from Server-X and sends the results back to Server-X.

By using the concept of XML-GUI, services can be developed independently, developers can provide the GUIs of their services as part of the code. For any modification, the service would be changed on the server directly. One of XML-GUI advantages is that we separate the real GUI description from the platform, which means that same GUI could be rendered differently on different platforms in order to match the context (screen size, fonts, etc).

5.3.4/ CLOUD MANAGER

Cloud Manager component maintains connections with other servers in the cloud. In a given CEMAT environment, CEMAT Servers are organized in a tree-based architecture, which could be formed of subtrees. Each CEMAT subtree has same attributes of a normal CEMAT tree. Cloud Manager has three subcomponents: Controller, Parent-Communicator and Child-servers Manager. CEMAT doesn't has a subcomponent for *Inter-CEMAT Communications*, the reason is that a CEMAT Server communicates with all other servers as roots for independent subtrees. A CEMAT Server can use all Things (Device and Emulators) in its tree (direct child-servers and their subtrees). A CEMAT Server can't communicate with its siblings (Other servers which shares the same parent-server). In the following subsections, we describe these subcomponents.

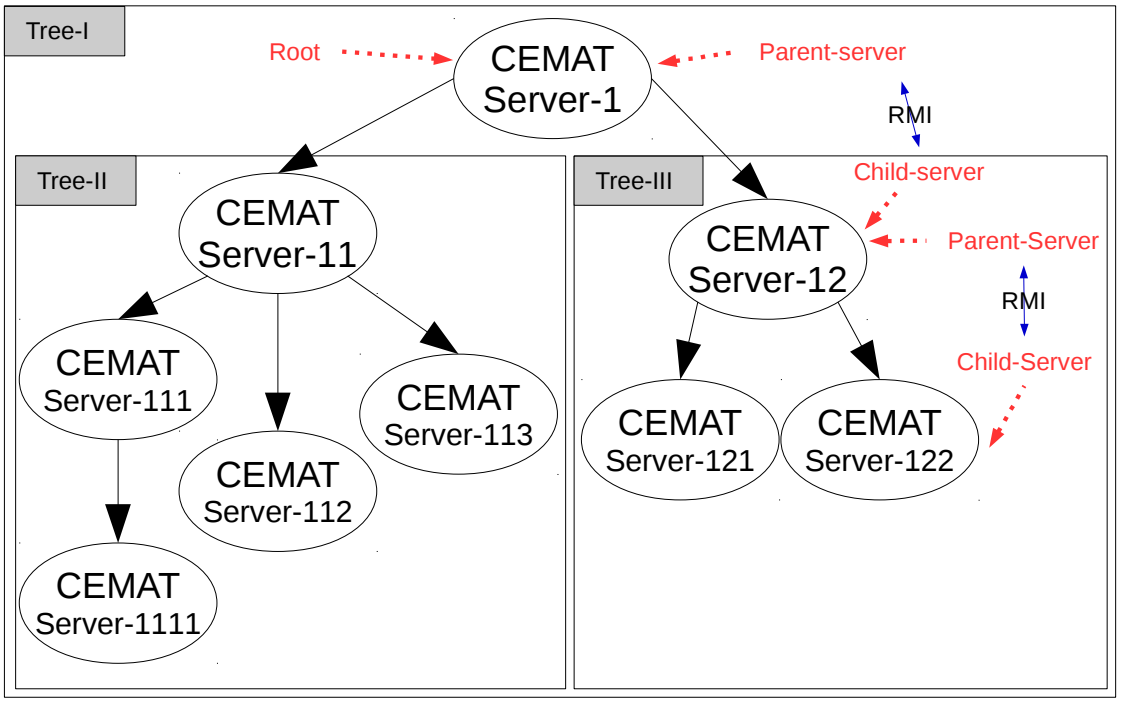

Figure 5.8: CEMAT - Servers' Tree

5.3.4.1/ CONTROLLER

Figure 5.8 shows an example of CEMAT environment composed of tree *Tree-I*, the root *Server-1* has two child-servers *Server-11* and *Server-12*. Both of servers are roots of subtrees *Tree-II* and *Tree-III* respectively. In order to exchange services and data between servers, they should use distributed system techniques. Since that CEMAT Server has been developed in Java, which means all servers are in Java, the best choice is Java Remote Method Invocation (RMI) system.
The Java RMI system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine [Oracle, 2015]. Using RMI in a tree-based architecture means that each pair (parent-server, child-server) will have a separated RMI connection. The number of RMI connections in a tree is equal to the number of (parent-child) relations in that tree.
A Java RMI application consists of 5 elements:

- **Interface:** It contains an abstract definition of the methods provided by the server. The interface should exist on both client and the server.
- **Remote Object:** It is an implementation of the methods which are described in the interface. The remote object exists only on the server.
- **Server Stub:** An object resides on the client and represents the remote object, it communicates with the Server Skeleton.
- **Server Skeleton:** An object on the server. It bridges between the Server stub on the client and the Remote Object on the server.

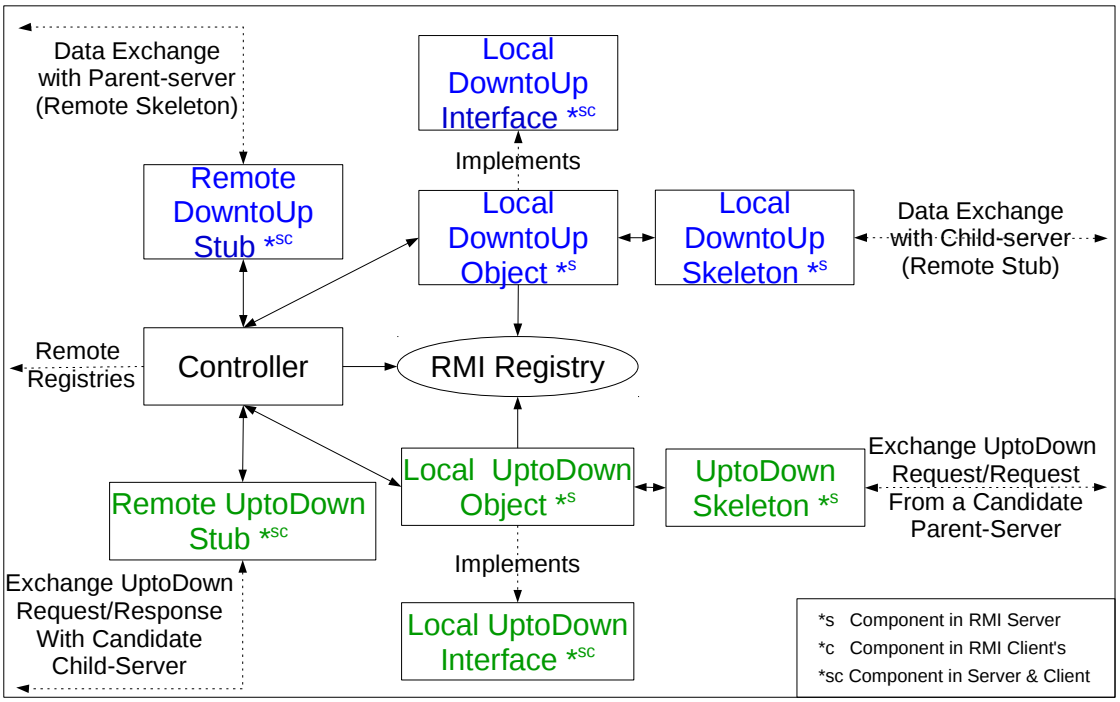

Figure 5.9: CEMAT Server - Controller Architecture

- **Registry:** It provides naming service for locating remote objects. It could be on the server itself or on an independent server.

CEMAT can build its tree using one or both of the following methods (Figure 5.9):

- **Down-to-Up:** In this method, the tree would be built starting from the child-servers (leafs). In other words, the child-servers connect to a parent-server. The parent-server could be the root or it could also be a child-server.
- **Up-to-Down:** A CEMAT Server could propose adoption to another CEMAT server using UptoDown RMI. In this case, building the tree could be started from the root by adopting child-servers and their subtrees (if exist).

The controller is responsible of the following tasks:

- Detecting and preventing loops (by using servers' names).
- Preparing tree updates in order to be sent to child-servers. Tree updates consist of all changes which took place between the server itself and the root.
- Forwarding service requests/responses from Daemon to Parent Communicator and Child-servers Manager, and vice versa.
- Forwarding service requests/responses received from other servers to Parent Communicator and Child-servers Manager, and vice versa.

Each CEMAT Server should have a local name which should be unique from siblings, ancestors and descendants. CEMAT Server utilizes the same methods as those used

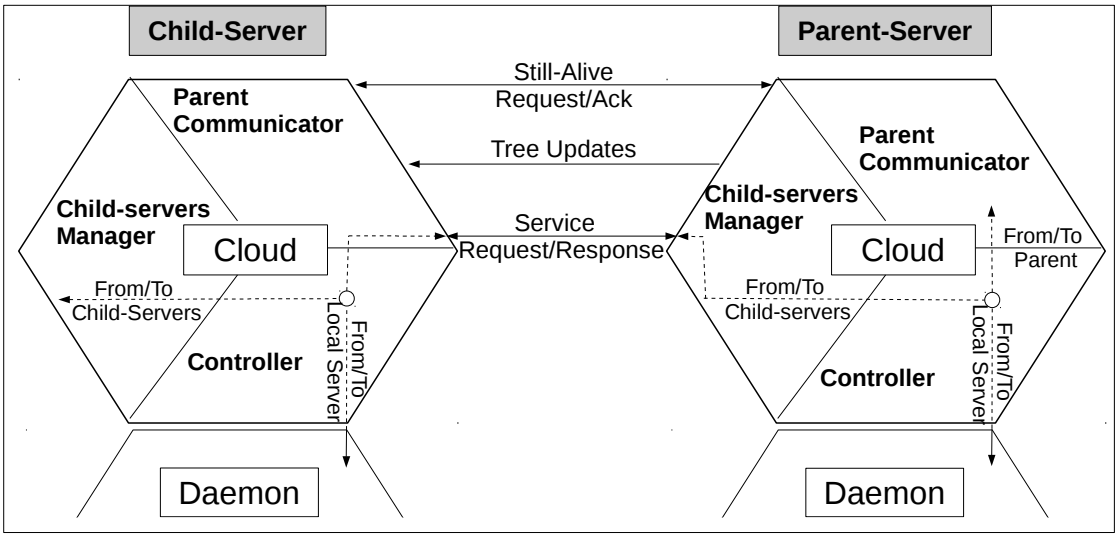

Figure 5.10: CEMAT Server - Services Exchange

by Domain Name System (DNS) servers to create URLs. Each CEMAT Server could be identified using its URL (path from the server to the root) which starts with the local server's name and ends with the root's name separated by "::". For example, in (Figure 5.8) the servers would have the following URLs: Server-112::Server-11::Server-1, Server-122::Server-12::Server-1.

5.3.4.2/ PARENT-COMMUNICATOR

This subcomponent maintains the connection with the parent (if exists). Parent-Communicator is responsible of the following tasks:

- Sending still-alive requests periodically and waiting for responses from the parent. If no acknowledgement is received, the request will be resent. This procedure will be repeated for a number of times (default value is 10 times) before considering the Parent-server is dead.
- In case of losing the connection with the Parent-server, a message will be sent to the Controller in order to send the updates to the child-servers and to start reconnection procedure.
- Forwarding service requests which are received from the Controller to the Parent-server.
- Forwarding service responses which are received from the Parent-server to the Controller.

Parent-Communicator doesn't interfere in connecting/reconnecting procedure. All its tasks are performed while the connection with the parent-server is active.

5.3.4.3/ CHILD-SERVERS MANAGER

It manages the child-servers which are connected to this parent-server. As soon as a child-server is connected, it will be added to child-servers list. The Child-server manager is responsible of the following tasks:

- Processing still-alive requests which are received from the child-servers and send back the responses.
- Checking its list periodically to find out dead child-servers (child-servers which stopped sending still-alive requests). The child-server is considered as dead server after a predefined time (default value is 60 seconds). The dead child-servers will be removed from the list.
- Forwarding service requests which are received from the Controller to the corresponding Child-server.
- Forwarding service responses received from the Child-server to the Controller.
- Multicasting tree updates which are received from the Controller to all its child-servers.

Child-servers Manager doesn't interfere in connecting/reconnecting procedure. All its tasks are performed with child-servers which are already connected to the parent-server.

5.3.5/ SCENARIOS

CEMAT defines two types of scenarios, Batch-based scenarios and XML-based scenarios. In the following we describe both of these scenarios and the differences between them.
In Batch-based scenario, command are sent directly without using an agent on the emulators and the devices. Figure 5.11 presents an example of this type of scenarios:

- A CEMAT Server *Server1* prepares a list of all emulators and devices (things) which will be part of the testing session. The list contains the emulators and devices of its descendants. The list contains: emulator11::server1, emulator111::server11::server1, device113::server11::server1 and emulator121::server12::server1.
- *Server1* prepares a list of commands (scenario) which should be applied on these things. The list consists of 4 commands: command01, command02, command03 and command04.
- *Server1* sends to each child-server the scenario and a list of its devices and emulators which should apply this scenario. Sending is performed using Threads which means that *Server1* will create a thread for each child-server. Therefore, all child-servers would receive the testing parameters at the same time.
- *Server11* receives a list which consists of emulator111 and device113.
- *Server12* receives emulator121, in addition to the scenario which is a list of commands.

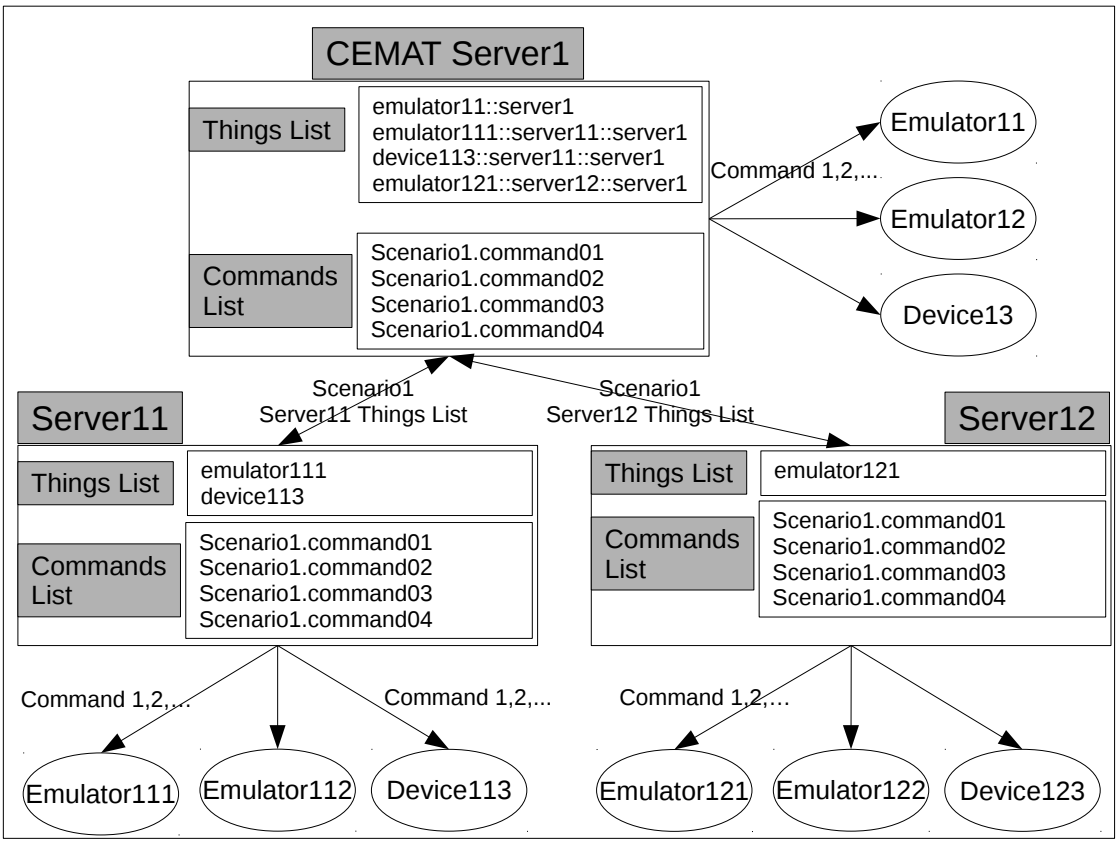

Figure 5.11: CEMAT - Batch-based Scenario

- As soon as *Server11* and *Server12* receive the data, they create a thread for each emulator or device and start sending commands one by one using ADB commands.

This type of scenarios is convenient when testing same scenario on all emulators and devices. In case of having different scenario for each emulator or device, XML-based scenarios should be used.
XML-based scenario is a pilot implementation of Scenarios component that has been described in subsection 4.7. XML-based Scenarios have the following advantages over Batch-based Scenarios:

- The ability to send different scenarios for different emulators and devices.
- Scenarios are sent using Constrained Application Protocol (CoAP) which a specialized web transfer protocol for use with constrained nodes. This is a light protocol for exchange data.
- Emulators and Devices receive the whole scenario. They don't have to be connected to a CEMAT server during the execution (offline mode).

Figure 5.12 presents the main architecture for XML-based Scenarios which consists of two subcomponent:

- **Scenario Manager:** It resides on one (or on many) CEMAT Server(s). Scenario Manager uses a local database to map scenarios which are stored on local/remote

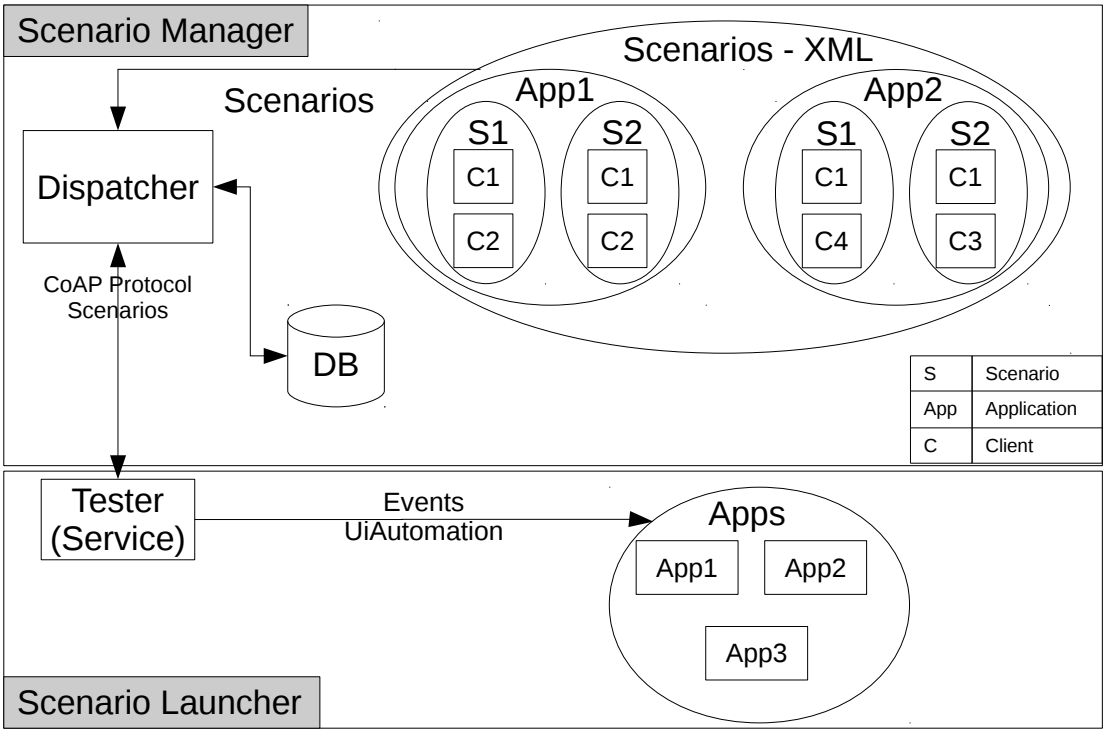

Figure 5.12: CEMAT - XML-based Scenario

file system, with the devices and the emulators which are connected to CEMAT server or to its descendants. The dispatcher sends to each emulator/device the corresponding scenario.

- **Scenario Launcher:** It resides on all emulators and devices. It is an android service which runs in the background. It maintains a connection with its CEMAT Server in order to exchange scenarios as soon as they are available on the server. The launcher uses *UiAutomation* which is a class for interacting with the device's UI by simulation user actions and introspection of the screen content [Android, 2015].

Batch-based and XML-based scenarios (as a pilot implementation) provide testers and developers with a method to execute certain scenarios. Collecting results depend on log and results files which should be provided by the applications (under test) themselves.

5.3.6/ TRAFFIC SHAPER

CEMAT Servers could be connected to the same network (for example, same LAN), they could be connected using the Internet. Network performance is different from a case or another, and it is different for the same case but in different times. Traffic shaper is a pilot implementation for NEP concept which is described in subsection 4.6.
Traffic shaper needs two types of method in order to perform its function:

- **Measurement Method:** It is a method to calculate packet delay, packet loss and other network performance attributes between every two CEMAT servers. That

means each CEMAT server should calculate these attributes between its own machine and its parent (if exists), and between its machine and each child-server. Instead of developing methods dedicated for performing these calculations, CEMAT utilizes the data which is provided by the *Still-Alive* request/ack messages which are used by *Cloud* component in order to maintain the connections with the connected servers. CEMAT utilizes the timestamps in *Still-Alive* messages in order to calculate the time needed to transmit a message from one server to another. It is important to mention that all CEMAT servers should be synchronized using a Network Time Protocol (NTP) Server.

- **Shaping Method:** Linux offers a set of tools for managing and manipulating network activities. Traffic Control (TC) is one of these tools which is a command line used to do the following tasks: deciding which packets to accept, rate of input, rate of output, packet loss, latency, etc.

These two methods are used by the Traffic shaper in order to control data exchange between CEMAT servers. In the following list, we describe the steps used by the Traffic shaper to manage the traffic on a CEMAT server:

- Each CEMAT server has default values for network performance attributes.
- A CEMAT server could maintain a local database which contains customized values for other servers in CEMAT tree.
- The Traffic Shaper should check periodically *Cloud* component in order to get the new measurement values for the connected server.
- Based on the received data and given network performance attributes, the traffic shaper could issue the needed parameters for the *tc* command.
- This procedure is repeated periodically based on a predefined time interval.

Instead of having a static database, it is possible to use a propagation model in order to simulate certain network scenarios.

5.3.7/ LOGGING

Logging is the act of recoding events, actions and messages so they can be used later to diagnose problems. CEMAT is a cloud environment which means that log data is distributed. Each CEMAT Server has its own log data. CEMAT uses java.util.logging package which is part of official Java packages and it contains the classes and interfaces needed for logging in Java. Other CEMAT components use logging package in order to send their logs. Log data is shown using two methods: Live log and XML log. Live log is a graphical interface in CEMAT which organizes log data according to its importance. A CEMAT Server could send a request to one of its descendants asking it to send its log data instantly. In other words, a CEMAT Server could use live log in order to monitor its own server and to monitor any of its descendant servers.
XML log is an XML file that contains all log data in form of records. Each record represents a detailed entry that contains all data about the log event. Code 5.3.1 shows an example of the record's structure. XML files could be processed using a log analyzer in order to

generate statistics and to compare files from different servers in order to have a full idea about the session or the system in general.

Code 5.3.1 - XML Log

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date></date>
  <millis></millis>
  <sequence></sequence>
  <logger></logger>
  <level></level>
  <class></class>
  <method></method>
  <thread></thread>
  <message></message>
</record>
</log>
```

5.4/ Experiments and Results

5.4.1/ Environment Design

Our experiments were done by preparing a real testing environment. The environment consists of (Figure 5.13 shows the logical structure):

- **Servers:** Three servers were used in the environment as follows:
  - **Server-01:** It is the root of CEMAT tree. It has the IP address 192.168.1.99/24.
  - **Server-02:** A child-server of the parent Server-01 which has the IP address 192.168.1.97/24.
  - **Server-03:** A child-server of the parent Server-01 which has the IP address 192.168.1.98/24.

  The servers run under *Linux Ubuntu Server 14.04.2 LTS - 64 bits*. They are connected to a Wi-Fi network.
- **Emulators:** The environment has 6 emulators (2 emulators on each server). The emulators used are Genymotion emulators which run as a virtual machine on the server (Oracle VM Virtualbox). Genymotion gives IP addresses 192.168.56.101, 192.168.56.102 to emulator-01, emulator-02 respectively. The emulators run on *Android 5.0 (Lollipop)* operating system.
- **Mobile Phone:** The mobile phone runs on *Android 4.2.1 (Jelly bean)* operating system and it is connected to a USB port of Server-01.
- **Applications for Testing:** The testing covers many android applications as follows:

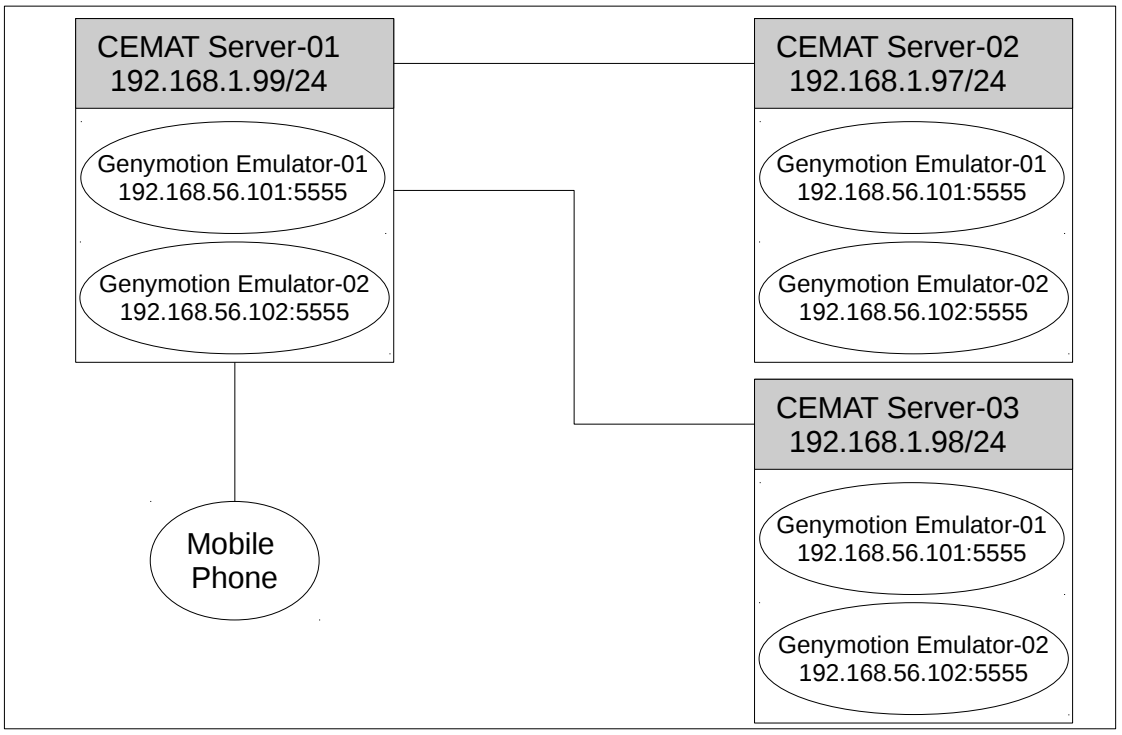

Figure 5.13: CEMAT Experiment Design

- **Settings:** It is an Android system application which allows users to modify device parameters, app features and behaviors. It has been tested in order to assure that CEMAT is able to send events to systems applications.
- **Google Chrome - Internet Explorer:** It is a pre-installed application. The test focuses on sending different browsing commands.
- **F-Droid:** It is an installable catalog of FOSS (Free and Open Source Software) applications for the Android platform. The test focuses on controlling the application by sending user events.
- **NotePad:** It is an open source application which isn't part of the official operating system. The test was registered using *Robotium Recorder*.
- **TestApp:** It is a testing application which was developed in order to test different GUI events. The test was performed using *CEMAT Scenarios*.

The main idea of testing different applications is to cover many categories, such as: system applications, closed applications, open source applications and in-house applications.

5.4.2/ Environment Installation

In the following, we present the steps which were applied in order to prepare the environment to test and evaluate the applications.

- Configuring IP addresses for CEMAT servers and connecting them to the Wi-Fi network.

- Deploying CEMAT framework as a Jar file and launch the application on the servers.
- Configuring CEMAT servers with their own parameters (name, paths, ADB, etc).
- activating DowntoUp RMI on Server-01 and starts listening to accept connections from Child-servers.
- Launching the emulators on Server-01 and connecting the mobile phone to USB port of Server-01.
- Launching the emulators on Server-02 and Server-03.
- connecting Server-02 to Server-01 as a Child-server.
- Activating Server-03 UptoDown RMI so Server-01 can send a request to Server-03 to become a Child-server (adoption).
- On Server-01: Adding all the emulators and the mobile phone to one list.
- Sending the Scenario Launcher to all testing entities.

5.4.3/ Scenario and Results

The testing scenario started with launching and manipulating *Settings* application by sending multiple events which they were sent using ADB commands. The mobile phone and the emulators received the events and apply them. Then, Google Chrome was launched using ADB and a request was sent to launch different web sites. F-Droid was sent to the mobile phone and the emulators and then it was installed, launched and upgraded on all of them. For testing NotePad, we used Robotium Recorder to generate NotePadTest (Unit Test) which contains the scenario for testing NotePad. NotePad and NotePadTest were sent, installed on all testing entities. Then, NotePadTest was launched and it applied the scenario on NotePad application. TestApp was sent and installed on the testing entities. Scenario Launcher on every testing entity launched its own scenario from the XML file and applied the events.
The test showed that using customized XML-based test is more accurate than sending same events to all testing entities. For example, after sending the events in the first scenario (Settings application), the mobile phone performed same touch events differently because it hadn't same screen dimensions. This problem could be solved using Robotium and XML-based testing because they can recognize the graphical components regardless of their positions on the screen.

5.5/ Conclusion

We presented the first IoTaaS-based pilot project CEMAT. CEMAT is an implementation to test Android applications. CEMAT integrates mobile devices, Android emulators and emulators developed by a 3-rd party. CEMAT is developed in Java in which there are two types of application: desktop application and Applet. In CEMAT, we used XML-based GUIs which mix the advantages of using desktop applications with the dynamic GUIs of applets. Two types of scenario have been developed: batch-based scenarios and XML-based scenarios. Experiments in a real environment have been described along with their results.

III

CONTRIBUTION - COLDE

6

CONNECTIONLESS DATA EXCHANGE (COLDE)

6.1/ INTRODUCTION

This is a suggestion to extend the IEEE 802.11 protocol by adding the functionality that allows exchanging data between two Wi-Fi entities without the need to have an alliance or establish a connection between them. The entities could be normal access points (infrastructure mode), ad hoc devices, Wi-Fi Direct or even normal Wi-Fi clients [Abu Oun et al., 2015]. The proposed extension is designed to simplify exchanging public and non-confidential data in IoT.

This extension allows broadcasting information to all Wi-Fi devices in certain areas, even if they are connected to different networks, or even if they are not connected to any network. COLDE allows the Wi-Fi devices (i.e, mobile phones and laptops) to benefit from the new services with the help of the other devices that include these services. For example, some mobile phones do not include certain localization systems, so they can get the current position from the other devices (which include that localization system), if these devices exist in the same geographical area and most specifically in the Wi-Fi coverage area of the first device.
Examples and situations vary with localization functions, emergency evacuation, integration between Wi-Fi devices and VANET, exchanging data with access points in the same area without the need to be connected to them, or to use a service from another device.
Nowadays, the majority of Wi-Fi networks are private networks. Such networks exist in large enterprises, small companies, shops, houses and even mobile Wi-Fi as in the case of a mobile phone running in an ad hoc mode or in a Wi-Fi direct mode. These Wi-Fi networks are mostly connected to the Internet using broadband connection. Private Wi-Fi owners do not open it to the public to avoid many threats. These threats could be classified into three main points:

- **Local network security:** to protect the internal network.
- **Public network security:** to prevent others from using the network in illegal actions like hacking other networks, sending spam,... or any other action which could be considered a cyber crime.

- **Service level:** to assure that nobody will use their Wi-Fi in a way that could degrade the whole performance of the network.

In order to realize this solution we need two components. The first one is an extension of the IEEE 802.11 in order to allow exchanging data without connection. We propose COLDE (Connectionless Data Exchange) which extends the IEEE 802.11 to exchange non-confidential and small amounts of data between Wi-Fi devices without a connection between them and without requesting them to disconnect from their original networks. Exchanged data could be a broadcast message from the server or one of the clients or it could be a service request from one of the clients. COLDE also organizes the hierarchy (if it exists) and the roles of entities, and defines the rules to send, receive and (re)broadcast the data. The second component is Lightweight Services, this component defines and the data that could be transferred using COLDE extension between the entities.

6.2/ IEEE 802.11 (Wi-Fi)

6.2.1/ Network Architecture Models

There are two main network architecture models. The first model is Open Systems Interconnection (OSI), which was presented by International Organization for Standardization (ISO) and which has seven layers. The second is DoD (U.S. Department of Defense) mode (also known as TCP/IP model or simply the Internet model). This model is widely used on the Internet, it organizes networks into four layers. (Figure 6.1) clarifies the relations between the two models. There is no perfect matching between the OSI Model and the DoD model because of the variance between them.

6.2.2/ IEEE 802.11 Key Concepts

A wireless Local Area Network (WLAN) is a data transmission system designed to provide location-independent network access between devices by replacing cable infrastructure by radio waves. The IEEE presented the original 802.11 specification in 1997 as the standard for wireless LANs. IEEE 802.11 standards focus on the bottom two levels of the ISO model (Figure 6.1), the physical layer and data link layer. In literature, Wireless Fidelity (Wi-Fi) and IEEE 802.11 are used interchangeably.

6.2.2.1/ IEEE 802.11 Architecture Model

Basically, there are two types of devices in WLAN networks [IEEE, 2012]:

- **Station (STA):** A logical entity that is a singly addressable instance of a medium access control (MAC) and physical layer (PHY) interface to the wireless medium (WM).
- **Access Point (AP):** An entity that contains one STA and provides access to the distribution services, via the WM for associated STAs.

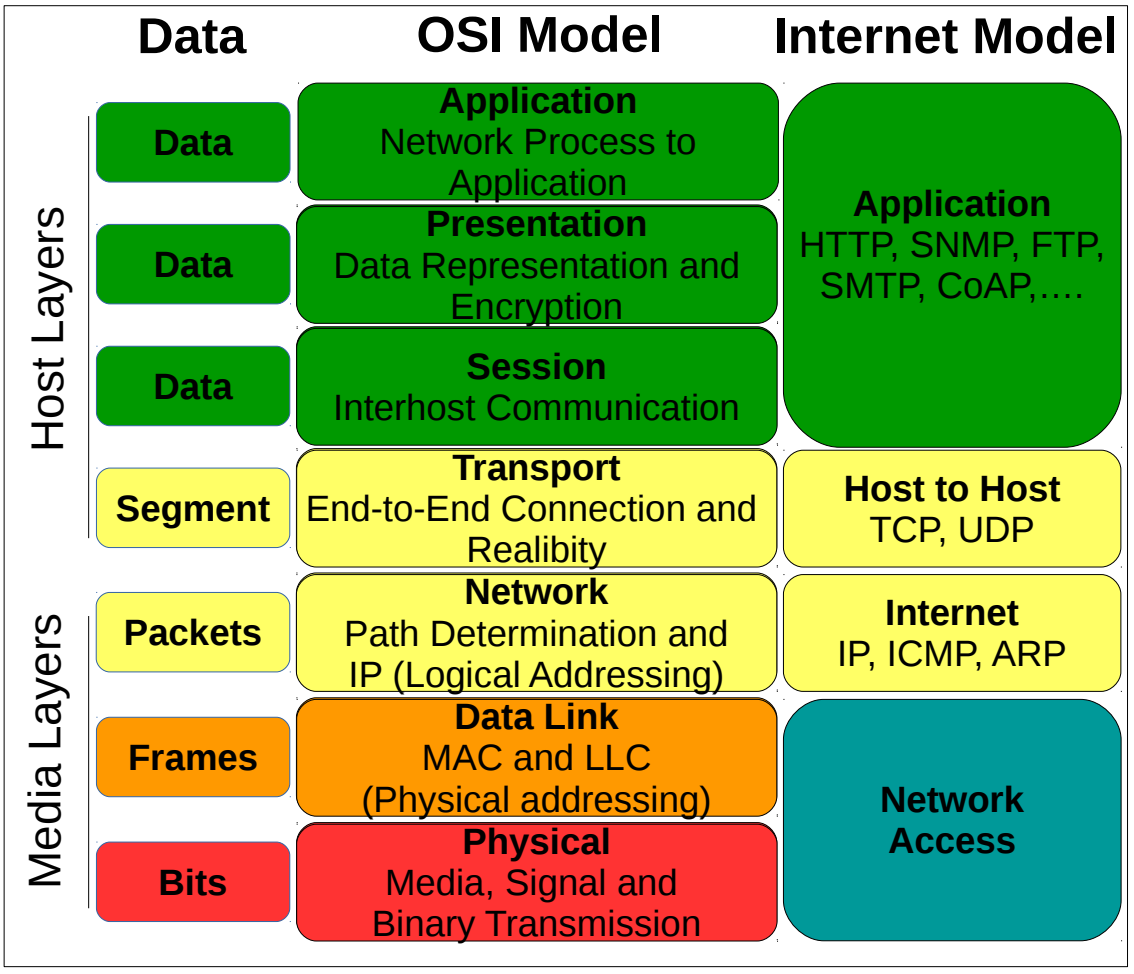

Figure 6.1: Network Models

Mainly, there are two modes in which communication can take place, in addition to a third mode has been developed recently:

1. **Infrastructure Mode:** Through an AP, a set of STAs are able to communicate with each other or with any other networks (Wired/Wireless) accessible by the AP.

2. **Ad hoc Mode:** A group of STAs need no AP in order to communicate with each other, they can communicate directly on a peer-to-peer basis.

3. **Wi-Fi Direct Mode (Soft-AP Mode):** It's a more feature-rich, secure, and smarter version of existing Wi-Fi ad-hoc networking. Wi-Fi Direct aims at enabling Device-to-Device communications between STAs (referred to as peers or nodes). Wi-Fi direct defines the term *Group*, in which it consists of two STAs or more. The group works as an infrastructure Wi-Fi on a single channel. One peer in the group acts as Group Owner (GO) and the other devices, called clients , associate to the GO. GO role is not predefined, but it is negotiated upon group creation/recreation [Casetti et al., 2014].

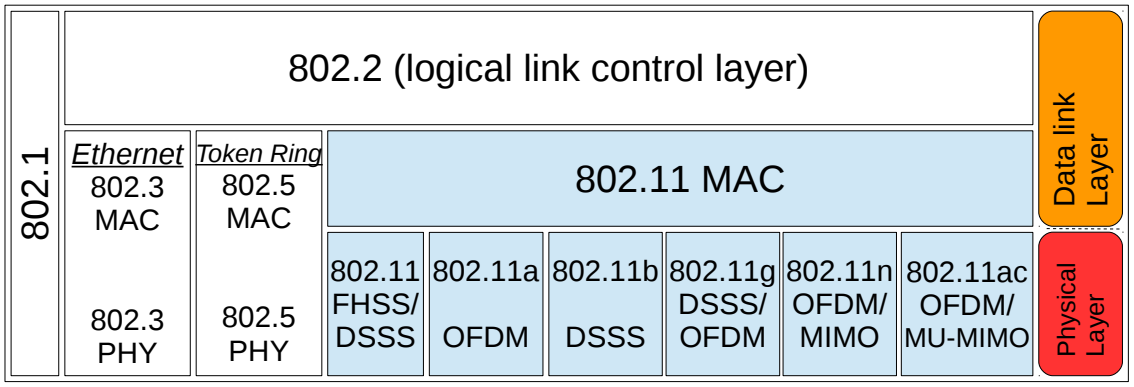

Figure 6.2: 802.11 standards in the IEEE 802 standard Suit

In infrastructure mode, a single AP together with all associated STAs is called a Basic Service Set (BSS). The informal (human) name of the BSS is called Server Set Identifier (SSID). The Basic Service Set Identifier (BSSID) is the formal name of the BSS and is always associated with only one BSS and it is the MAC address of the AP. Two or more interconnected wireless BSSs that share the same SSID, security credentials are called Extended Service Set (ESS) and in this case the SSID is called Extended Service Set Identifier (ESSID).
In ad hoc mode, client devices in ad hoc network without an AP form an IBSS (independent BSS).

The IEEE 802.11 protocol covers the MAC and Physical Layers of the OSI model (Figure 6.2):

- **Physical Layer (PHY):** It includes a specification of the transmission medium and the topology. In addition that it includes functions as: Encoding/decoding of signals, Preamble generation/removal (for synchronization), Bit transmission/reception. There are 5 standards available: IEEE 802.11 (a,b,g,n and ac), Table 6.1 provides a brief comparison between these standards [Khanduri et al., 2013].
- **Medium Access Control Layer (MAC):** The MAC Layer defines two different access methods. The Basic Access Method is the Distributed Coordination Function (DCF), which is a Carrier Sense Multiple Access with Collision Avoidance mechanism (usually known as CSMA/CA). Another optional protocol that is part of the IEEE 802.11 standard is the Point Coordination Function (PCF). The main functions of MAC layer are the following [Stallings, 2002]:
  - On transmission, assemble data into a frame with address and error detection fields.
  - On reception, disassemble frame, and perform address recognition and error detection.
  - Govern access to the transmission medium.

Standard	Maximum physical rate	T_x	Spectrum	Compatible with 802.11
802.11	2.4 GHz	2 Mbps	DSSS/ FHSS	None
802.11a	5.0 GHz	54 Mbps	OFDM	None
802.11b	2.4 GHz	11 Mbps	DSSS	802.11
802.11g	2.4 GHz	54 Mbps	OFDM/ DSSS	b
802.11n	5 or 2.4 GHz	600 Mbps	OFDM / MIMO	a/b/g
802.11ac	5.0 GHz	6.93 Gbps	OFDM / MU-MIMO	a/n

Table 6.1: IEEE 802.11 Standards

6.2.2.2/ IEEE 802.11 MAC FRAMES

The MAC frame format comprises a set of fields that occur in a fixed order in all frames. Figure 6.3 depicts the general MAC frame format. The first three fields (Frame Control, Duration/ID, and Address 1) and the last field (FCS) constitute the minimal frame format and are present in all frames, including reserved types and subtypes. The fields Address 2, Address 3, Sequence Control, Address 4, QoS Control, HT Control, and Frame Body are present only in certain frame types and subtypes.
The Frame Body field is of variable size. The maximum frame body size is determined by the maximum MAC Service Data Unit (MSDU) size (2304 octets), plus the length of the Mesh Control field (6, 12, or 18 octets) if present, the maximum unencrypted MAC Management Protocol Data Unit (MMPDU) size excluding the MAC header and FCS (2304 octets) or the maximum Aggregate-MSDU (A-MSDU) size (3839 or 7935 octets, depending upon the STA's capability), plus any overhead from security encapsulation [IEEE, 2012].

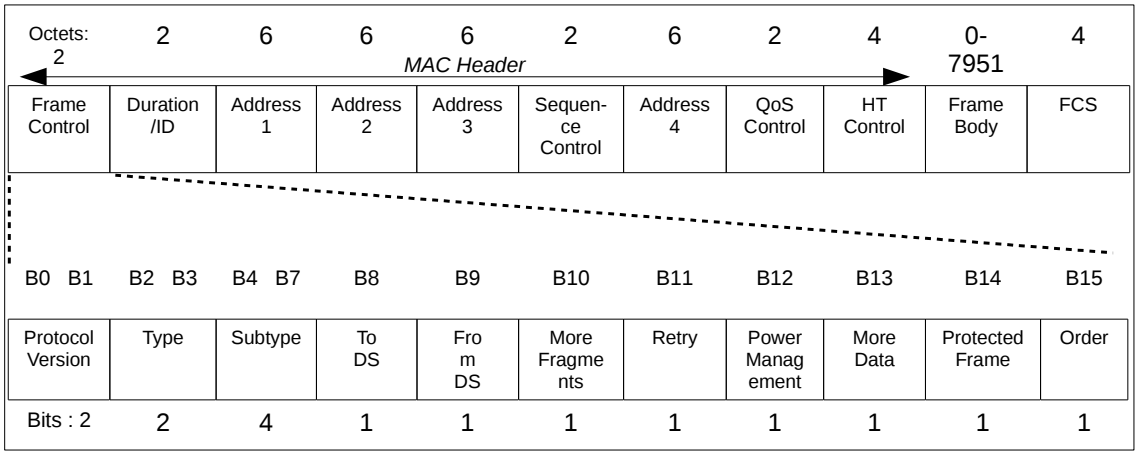

Figure 6.3: MAC frame format

There are three main types of frames (one can differ between them based on *Type* field value B2-B3 (Figure 6.3)) [IEEE, 2012] [Westcott et al., 2011]:

- **Management frames (Type = 00):** They are used by STAs to join and leave the

BSS. Because of their importance in our research, they are described in detail in subsection 6.2.2.3

- **Control frames (Type = 01):** Control frames have only header information. They assist with the delivery of the data frames, clear the channel, acquire the channel and provide the unicast frame acknowledgements. They are transmitted at one of the basic rates (mandatory), so they would be heard by all the stations. There are 9 subtypes control frames: Control Wrapper, Block Ack Request (BlockAckReq), Block Ack (BlockAck), PS-Poll, RTS, CTS, ACK, CF-End and CF-End + CF-Ack.
- **Data frames (Type = 10):** Data frames that carry MSDU data that is passed down from the higher-layer protocols are normally normally encrypted. The data frames that carry no MSDU payload are not encrypted. There are 15 data frame subtypes grouped as following: Data + [, CF-Ack, CF-Poll, CF-Ack + CF-Poll], no-data [Null, CF-Ack, CF-Poll, CF-Ack + CF-Poll], QoS Data + [,CF-Ack, CF-Poll, CF-Ack + CF-Poll] and QoS [Null, CF-Poll, CF-Ack + CF-Poll].

6.2.2.3/ IEEE 802.11 MAC MANAGEMENT FRAMES

Management frames are also known as MAC Management Protocol Data Unit (MMPDU). They are used by STAs to join and leave the BSS. The following is a list of 14 subtypes management frames as defined by the 802.11-2012 Standard: Association (request/response), reassociation (request/response), Probe (request/response), Timing Advertisement, Beacon, Announcement Traffic Indication Message (ATIM), Disassociation, Authentication, Deauthentication, Action and Action No Ack. Management frames are the only ones that are sent/received all the time, regardless of STA's status, whether it was already associated to an AP or still searching for one. Figure 6.4 presents the management frame format. Frame Body field consists of two different set of fields:

- **Information Elements (IE) Fields:** These fields are used to send additional information. These fields have numerous potential application areas which require embedding additional information in the management frames. The general format of IE fields has three fields:
  - **Element ID:** It is a field of one byte with an unsigned integer, in case of using "Independent RIE", it has a value between 32-255. In case of using "Vendor-specific Information Element", this fields is set to 221.
  - **Length:** It is a field of one byte with an unsigned integer. The Length field specifies the total number of octets of the next field.
  - **Variable:** It is a variable-length element-specific Information field.
- **Non-IE Fields:** In the 802.11-2012 standard, there are 42 non-IE fields, such as Beacon Interval field, Capability Information field, etc.

We are going to focus on only three subtypes management frames (based on subtype value b7 b6 b5 b4 (Figure 6.3)). These three frames play an essential role in scanning phase :

- **Beacon Frame (subtype = 1000):** It is used by the APs (and the STA's in an IBSS) to communicate throughout the serviced area the characteristics of the connection

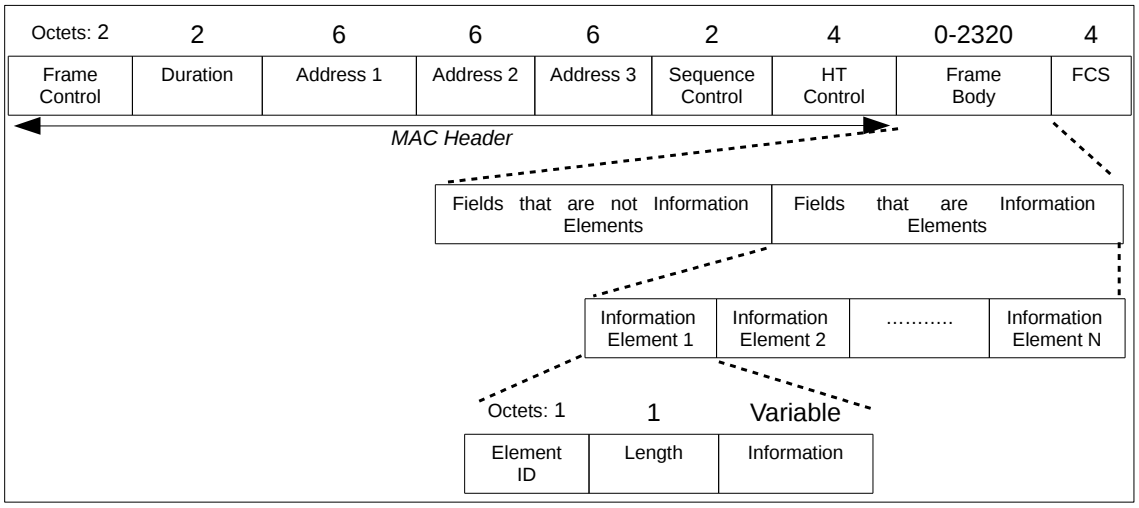

Figure 6.4: MAC Management frame format

offered to the cell members. This information is received by potential clients (in passive mode) and by clients that are already associated to a BSS. They are sent periodically at a rate predefined.

- **Probe Request Frame (subtype = 0100):** It is used by a STA to obtain information from another STA or AP. This frame is used by a STA to locate any, or a particular IEEE 802.11 BSS.

- **Probe Response Frame (subtype = 0101):** It is used to reply to a probe request.

6.2.3/ IEEE 802.11 Station Access Phases

A Wi-Fi client access process involves three phases (Figure 6.5):

- **Scanning (or Discovery):** The IEEE 802.11 standard defines both types of scanning techniques [Gupta et al., 2007].

  – Passive scanning mode, the WNIC (Wireless Network Interface Card) listens on one channel at a time for Beacon Frames from APs. It records the corresponding signal strength and other relevant information about the AP. Using this information, the WNIC then chooses which AP to associate with.

  – Active scanning, Probe request frames are transmitted on all the channels. The responses received from APs in the form of Probe Response Frames are then subsequently processed by the WNIC. Active scanning is the default scanning technique for a WNIC, which enables it to implore an immediate response from an AP, without waiting for the beacon frames to be sent by it

- **Authentication:** The IEEE 802.11 authentication is the first step in network attachment. IEEE 802.11 authentication requires a Wi-Fi client to establish its identity with an AP. There is no data encryption or security at this stage.

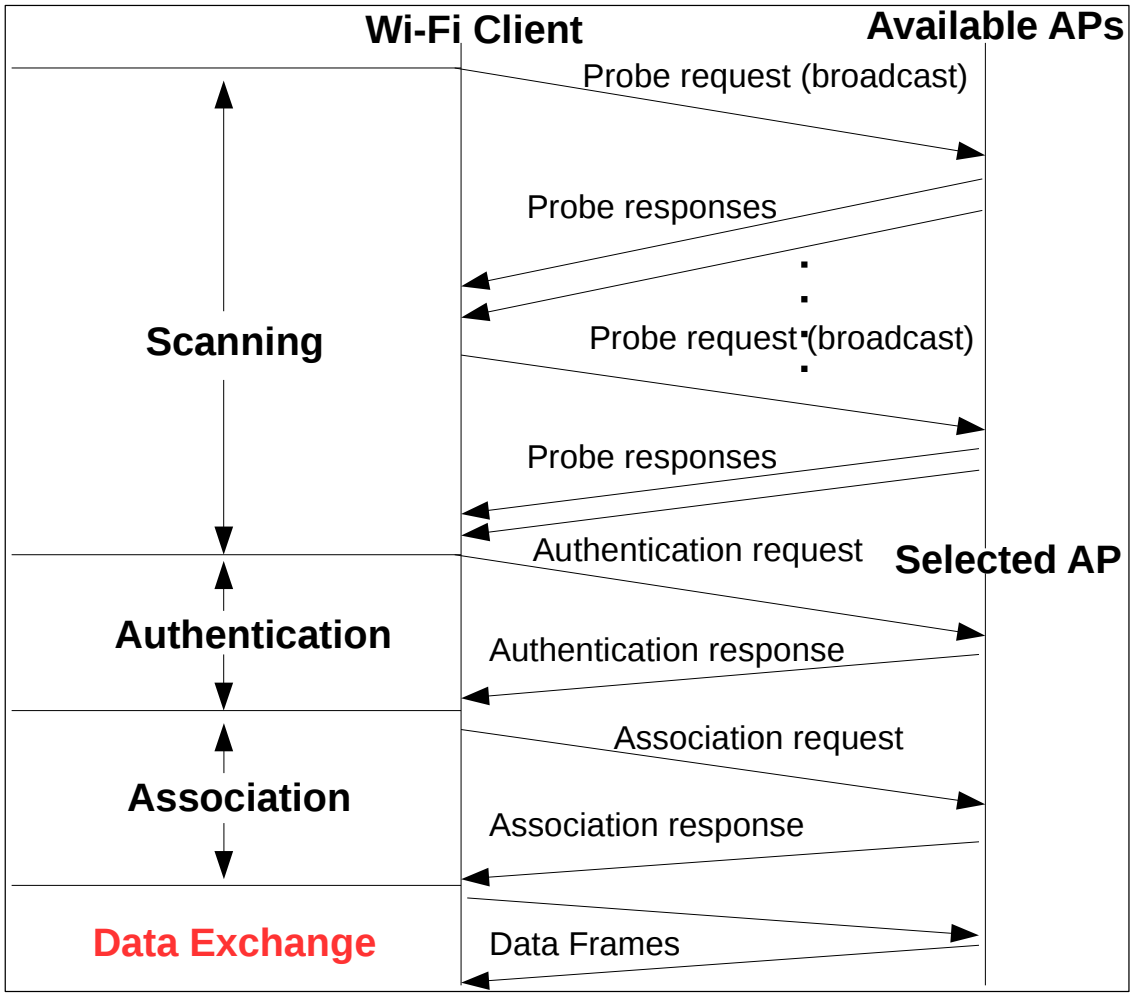

Figure 6.5: Wi-Fi STA Access Phases

- **Association:** Once authentication is complete, Wi-Fi clients can associate (register) with an AP to gain full access to the network. Association allows the AP to record each Wi-Fi client so that frames may be properly delivered. Association only occurs on wireless infrastructure networks, not in ad hoc (peer-peer) mode. A station can only associate with one AP at a time[Intel, 2014].

It is important to mention here that a client keeps scanning for Wi-Fi network (Active/Passive Scan), even after being associated with a network. In other words, scanning happens as long as the Wireless NIC is enabled.

6.3/ COLDE PROTOCOL STACK

COLDE concept depends on simplifying non-confidential-data exchange. COLDE defines two components:

- **Component I (COLDE Extension) :** An extension of the IEEE 802.11 protocol

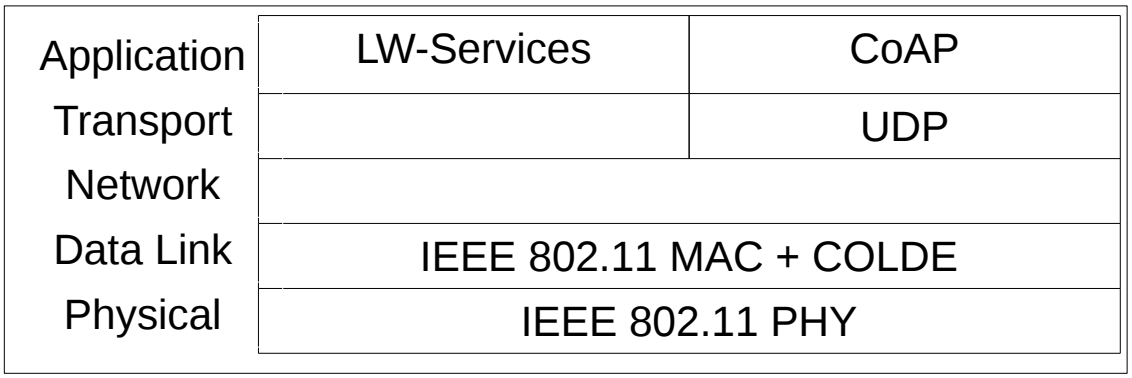

Figure 6.6: COLDE Protocol Stack

(Data-Link layer), we refer to it here as *COLDE* or *COLDE Extension*.

- **Component II (LightWeight Services):** An application architecture that will run on the application layer. It helps to organize the data that will be sent using COLDE extension.

This stack has been designed to offer smaller packet overhead, thus resulting in a faster data exchange. The layers that have been excluded are:

- **Network Layer:** COLDE piggybacks the management frames of the IEEE 802.11. These frames are independent from the upper layers. Network layer addressing and routing are not needed. The MAC layer frames depend only on MAC addresses. Excluding network layer allows data exchange with other networks (in case that the Wi-Fi client is connected to a network). At the same time, the Wi-Fi client that isn't connected to any network has no IP address. So in both cases, the network layer can be excluded.
- **Transport Layer:** COLDE doesn't rely on session. The concept of COLDE is that there is an application trying either to get/broadcast information, it will prepare the service PDU and send it. The PDU will not be fragmented. No session will be held on the level of the operating system. No special packets of frames will be sent. The application is responsible of sending another request in case of not receiving a response.

On the other hand, COLDE can play an important role as the first protocol to enable connectionless IoT. COLDE can be used to offer communication Data Link layer for CoAP (discussed in section B.2.4.2). CoAP depends on UDP as a transport protocol. In this scenario, the service PDU in COLDE frame will be the UDP datagram. Access points can play the role of relay agents or CoAP-proxies.

6.4/ COLDE DESIGN AND STRUCTURE

Creating a connection or an alliance is an essential phase for data exchange between a Wi-Fi access point and a Wi-Fi client, or between clients themselves on the Wi-Fi network.

Having a connection between the access point and the Wi-Fi client means that the client is a part of the network and can have access to the rest of the network. In public Wi-Fi networks that should not be a problem, usually there is no sensitive or confidential data in it, it just offers connection to the public Internet and it is available to anybody. But it is not the same situation for the private Wi-Fi, only authorized persons with the required credentials can connect to that Wi-Fi and use its services. In case two clients need to exchange some data, they should be connected either using an AP or one of them is in ad hoc mode or a Wi-Fi direct mode, which leads to the same problem between a Wi-Fi client and a Wi-Fi access point.
Actually, the proposed extension is only software and needs no special hardware. Any Wi-Fi device, whatever its role in the network, can be provided with an extension, which means it can be provided by any access point, mobile in an ad hoc mode, mobile in a direct Wi-Fi mode or even a normal mobile running in a pure client mode.
COLDE allows distinguishing devices based on their MAC addresses regardless of the network that relayed its request.

6.4.1/ COLDE - WORKING METHOD

In the normal mode, one can't exchange data before going through the three phases and being part of the network. In COLDE we exchange the data during the first phase (Scanning). The data will be carried into beacon frames for broadcasting the information, and into probe Request/Response Management Frames (Figure 6.4) to request a service. This approach more specifically uses the Request Information Element (RIE) part of the management frame (in case of the probe Request/Response frames), which is a variable length part, which the client usually uses it to ask the access point for some extra information like the SSID, the supported rate, etc.
The frame exchange comprises the following steps (Client side):

1. When an application needs certain data, it prepares the Packet Data Unit (PDU) of the corresponding Service.
2. The application sends the PDU to the Operating System (OS)'s library that manages the Wi-Fi NIC (WNIC).
3. The OS's library adds the PDU as an IE.
4. The OS's library sends the IE with the next Probe Request (Active scanning).
5. When the OS's library receives the Probe Responses, it sends the received IEs to the application.
6. The application searches the IEs for the demanded service. If the response isn't received, the application will try to send another PDU after certain timeout.

The frame exchange comprises the following steps (AP side):

1. An AP receives the Probe Request and extracts the IEs.
2. If the AP has COLDE extension, it can understand the service request, otherwise it will treat the probe request as any other requests.

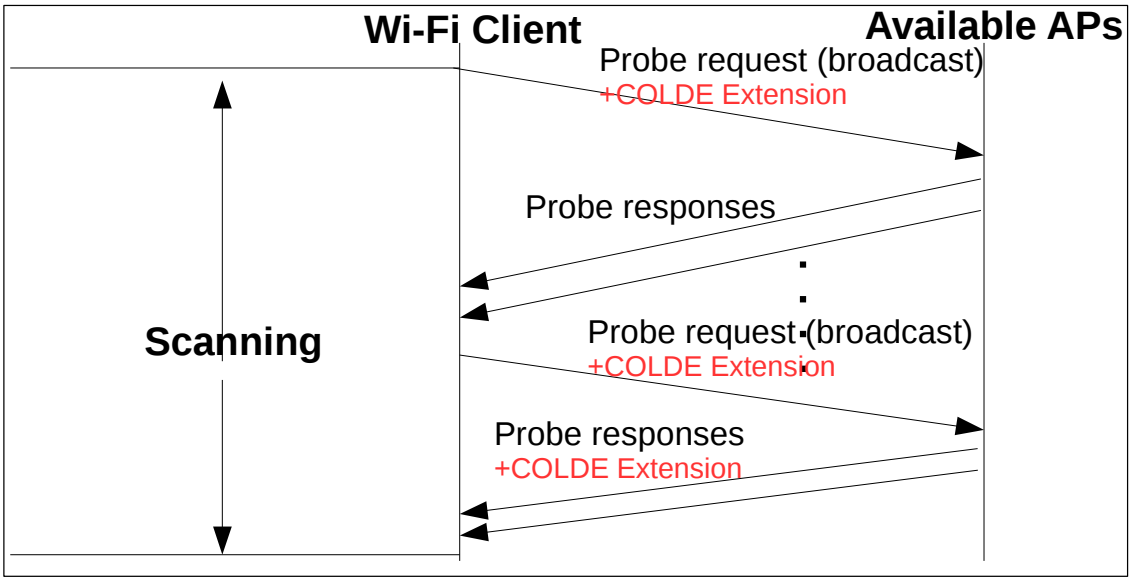

Figure 6.7: COLDE Extenstion - Working Method

3. If the AP can reply to COLDE request, it adds the response as an IE. The IE will be sent in the next probe response.

6.4.2/ COLDE FRAMES

COLDE frame can be added as an IEEE 802.11 Information Element in two different forms:

- **Independent RIE:** Each request information element has a unique ID, the ID numbers between 32-255 have been reserved for future use. One of these IDs could be used to define a new information element to send a special request from a Wi-Fi entity to another Wi-Fi entity (broadcast if the SSID is unknown, directed if the SSID is already known).
- **Vendor-specific Information Element:** Because of the extensive importance and to allow some flexibility to the vendors, the IEEE 802.11 standard itself has a provision to carry non-standard, vendor-specific information in the "Vendor Specific Information Element" (IE) field of management frame (Figure 6.8). This IE (with ELEMENT ID 221) is provisioned to be always present as a last IE in the frame body of beacon. Using it, up to 251 bytes of information can be embedded in each management frame[Gupta et al., 2012a].

COLDE extension is defined to have a common general format consisting of (Figure 6.9):

- **Options - B0 (SOS):** This bit is used to indicate the urgency of the request. COLDE will give this request the priority over the rest of the requests. At the client, COLDE will not wait for the periodic scanning and it will request an immediate network scanning in order to broadcast the request directly. At the AP, the AP will give this request the priority over the other requests from all clients.

Octets: 1	1	3	1	X
Element ID	Length	OUI	Sub-OUI	Data
221	L=3+1+X (X<=251)			

Figure 6.8: Vendor-Specific Information Element

- **Options - B1 (Authenticated):** This bit indicate whether this request is coming from an authenticated client. Service Authentication is discussed in details in section 6.6 (COLDE Security).
- **Options - B2 (Server):** The client can manually add a server address port. The AP has the choice of forwarding the request to the mentioned server, forwarding the request to the default server or ignoring the request.
- **Options - B3 (Reserved):** This bit is reserved for future use.
- **Service Category (B4 - B7):** These four bits can help to categorize the services. Based on these bits, the APs can filter the requests to allow/reject certain types of services. The service categories can be any type of public services in which no private or classified data is included. For example, localizations services, emergency request, evacuation services, warnings, evacuation directions, weather forecast,etc. In case of fixed Wi-Fi access points, there could be many data channels in which the Wi-Fi client can check from time to time in order to get the latest updates. Table 6.2 lists the main service categories.
- **Service ID:** It is a one octet field with an unsigned integer. It gives the possibility to define 256 different services for each category.
- **Server's Address:** A field of 16 octets, it specifies the server's address. It can be an IP-v4, IP-v6 or an URL (Uniform Resource Locator). This field exists only if B2 is set to 1.
- **Server's Port:** It is a field of two octets which specifies the port address of COLDE service on the server. This field exists only if B2 is set to 1.
- **Service PDU:** It is variable-length field, each service can define its own data structure. The length of this field depends on the maximum allowed by Information element in IEEE 802.11 standard.

Category	Name	Description
0000 (0)	SOS	Emergency and evacuation services
0001 (1)	Essential Info	Time, Weather, etc.
0010 (2)	Telecommunication	Services related to mobile telecommunication (ex. LTE D2D (Device-to-Device))
0011 (3)	Localization	GPS, Indoor positioning and Indoor navigation services

Continued on next page

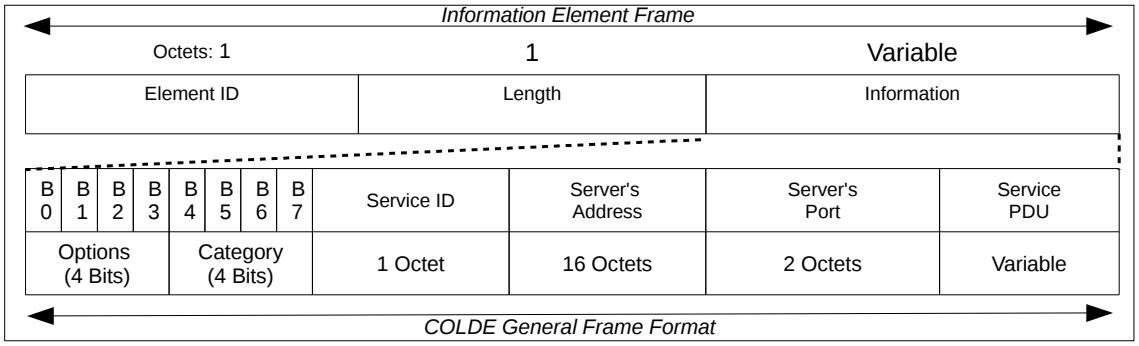

Figure 6.9: COLDE General Frame Format

Continued from previous page

Category	Name	Description
0100 (4)	Transportation	Traffic information, Public transport broadcast (Bus, metro and trains information), flights updates (Airports), etc.
0101 (5)	Commercial	Commercial offers, products information (ex. Supermarkets, Restaurants, Hotel, etc.)
0110 (6)	Parking	Parking services (Available spots, spots location, etc.)
0111 - 1111 (7-15)	Reserved	For future use

Table 6.2: COLDE - Service Categories

6.4.3/ COLDE HIERARCHY

COLDE is a hybrid system in which it can operate depending on a centralized and non-centralized hierarchy at the same time. There are two main characteristics in this design:

1. A centralized system in which there is a server control forming the broadcast tree on the first level only, sending broadcast messages and responding to the LightWeight Services requested by the nodes in the tree.

2. A Non-centralized system in the way that the nodes work without the need to know the parent-node or the children-nodes, the nodes can send broadcast messages, but the receivers can distinguish the messages according to their sources, among which the messages from the server should be more credible than the others.

Two main components could exist in the environment (Figure 6.10):

- **Server:** It is the root of the broadcasting tree which maintains the connections with the direct children-nodes; the connections between the server and its child-nodes are established on the Internet/Intranet. The server can push notifications/requests

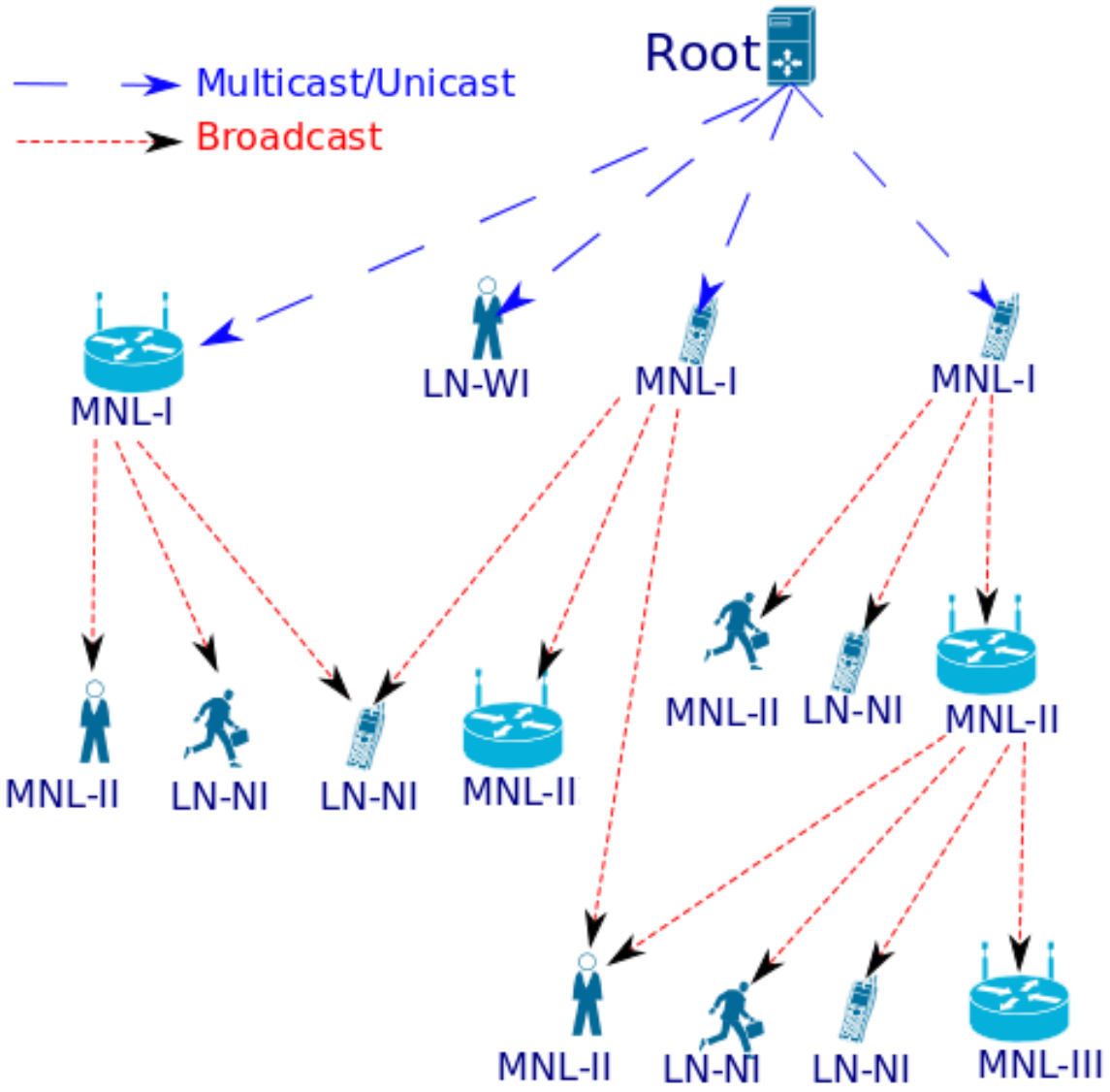

Figure 6.10: COLDE - Broadcasting Structure

to a specific child-node(s) directly, and they will broadcast these messages in the sub-tree(s).

- **Node:** The node is any device which has a wireless card that can be used to receive/send the data from/to its environment or directly from the server. Each node has many attributes, i.e. GPS info (position, number of satellites, number of child nodes, battery Level, node speed, bandwidth, traffic cost, neighbors,...).

In the following, we discuss node types, their function in the hierarchy and node transformation between types.

6.4.3.1/ Node Types

The nodes are categorized according to their location and their capabilities (Table 6.3):

- **Main-Node Level-I (Type-I or MNL-I)**
  The main node level-I is a wireless device (a mobile or an access point) the parent of which is the root, it receives messages from the root as unicast on the IP network (3G, Broadband). It broadcasts the received messages (notifications, requests) in its Wi-Fi coverage. The main node level-I can send requests to the root directly as unicast, besides periodically sending a neighborhood list to the server. The delay between two updates can be either pre-defined by the server or even customized by the node.

- **Main-Node Level-N (Type-II or MNL-N)**
  The main node level-N is a wireless device (a mobile or an access point) the parent of which is one of the main nodes (N is the depth of the node in the tree), but it can communicate directly with the root on the IP network, it receives messages from the parent as broadcast on the Wi-Fi. It rebroadcasts the received messages (notifications, requests) in its Wi-Fi coverage. The main node level-N can send the requests to the root directly as unicast, it periodically sends a neighborhood list to the server. The delay between two updates can be either pre-defined by the server or even customized by the node.

- **Leaf-Node with Internet (Type-III or LN-WI)**
  The leaf-node with Internet is a wireless device (a mobile or an access point) the parent of which is the root, it receives the messages from the parent as unicast. It cannot rebroadcast the received messages and it cannot receive any broadcast from any other main node. So, the leaf-node with the Internet could either be main node level-I or main node level-N, but without the ability to receive/(re)broadcast the messages from/in its Wi-Fi coverage. This type can work in case a mobile does not include the protocol but it has Internet/Intranet connection with the server.

- **Leaf-Node - No Internet (Type-IV or LN-NI)**
  The leaf-node without Internet is a mobile the parent of which is another main node different from the root. It receives messages from the parent as broadcast on the Wi-Fi. It can not rebroadcast the received messages. Any request should be relayed by another main node. This type can work in case a mobile does not include the protocol and at the same time has no Internet/Intranet connection with the server.

	MNL-I	**MNL-N**	**LN-WI**	**LN-NI**
Device	Mobile or AP	Mobile or AP	Mobile or AP	Mobile
Parent	Root	MNL-(N-1)	Root	MNL-N
Connection with Parent	3G/Broadband	Wi-Fi	3G/Broadband	Wi-Fi
(Re)broadcast	Yes	Yes	No	No
Can communicate with root	Yes	Yes	Yes	No
Receive broadcast	Yes	Yes	Yes	No
Demand Service	Unicast to root	Unicast to root	Unicast to root	Wi-Fi Broadcast
Neighborhood List	Yes	Yes	No	No

Continued on next page

Continued from previous page

	MNL-I	MNL-N	LN-WI	LN-NI

Table 6.3: COLDE - Node Types

6.4.3.2/ MAIN-NODES SELECTION CRITERIA

As soon as the node (Type-I, Type-II or Type-III) connects to the root, the parent will be the root itself, the node will be either Main-Node Level-I or Leaf-Node with the Internet (Type-III), depending on its capability to broadcast the messages in its Wi-Fi coverage. Periodically, the root evaluates the tree structure starting from its own child nodes, the evaluation process depends on the following rules (Figure 6.11):

- The node stays as a 1st level main node as long as there is no other main node in its Wi-Fi coverage area. The server can decide whether there is another node in the area either:
  - by using the geographical position of the node.
  - or because the node receives no broadcast from other main nodes.
  - or because it is moving with the highest speed than a pre-defined node speed.
- otherwise, the node will be moved to one of the other main nodes to be a child node in the same area according to the following priorities:
  - Age of the main node.
  - Speed of the main node.
  - The battery level of the main nodes.
  - The greatest number of satellites in view of the main nodes.
  - The lowest number of neighbors of the main nodes.

6.4.4/ MULTI-TIER BROADCAST

Broadcast is referred to when transmitting a message that will be received by every device on the network, while multicast is the delivery of a message or information to a group of destination computers simultaneously in a single transmission from the source[Tanenbaum, 2002].
If one wants to broadcast a message to all the devices in selected areas, regardless of which wireless network they are connected to, neither the multicast solution nor the broadcast solution can achieve this goal separately, but a solution consisting of both of them can.

We present a solution “Multi-Tier Broadcast” which depends on the multicast and the broadcast at the same time, and on many levels. The destined message should be sent using multicast to some devices in these areas, so they can broadcast it to the rest of the devices. Each device receiving it will rebroadcast it again till the message expires. The message expires either when the TTL is 0 or when its validity time runs out. If the devices

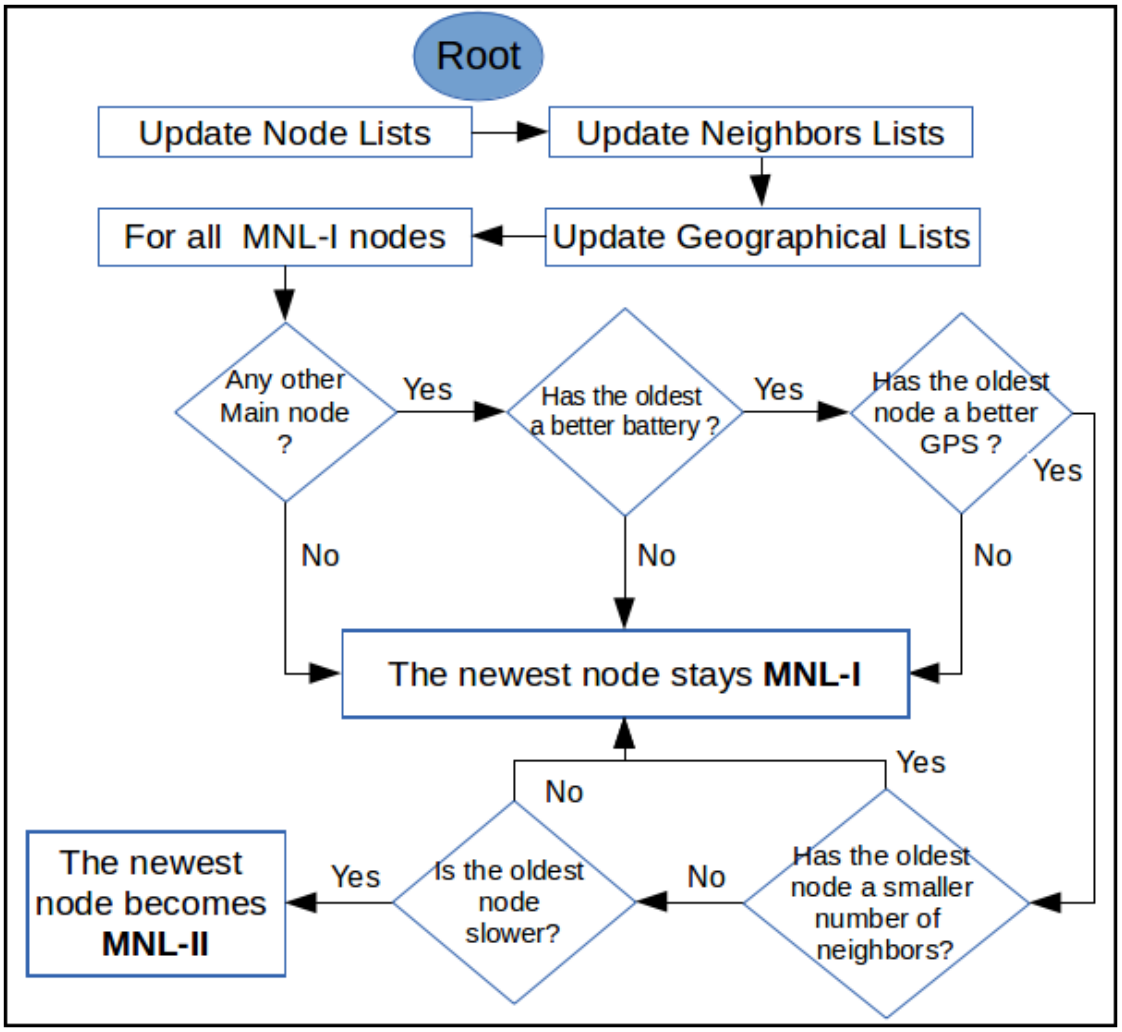

Figure 6.11: COLDE - Main Node Selection

in these areas do not include the multicast, the first phase can be done using the unicast. But even with a Multi-Tier solution, a lot of devices will not receive the message, these devices are either not connected to any network or the devices receive no messages from the source. We can solve this problem by using Multi-Tier Broadcast with COLDE.
In this mode, the server sends a message to its children nodes (Nodes Type-I (MNL-I) and Type-II (LN-WI)), each request has a TTL, the TTL is combined with two factors:

- **Number of Rebroadcasts:** this defines how many times the message will be rebroadcast, each node decreases this value by 1.

- **Expiry time:** this is a timestamp to define when the message will expire, this factor has priority over the first factor.

6.5/ LIGHTWEIGHT SERVICES EXCHANGE SYSTEM

Lightweight Services (LW-Services) are the services that depend on a non-confidential small amount of data. This data could be transferred using only one frame, so that it will not consume the resources of the providing entity. These services could be anything, for example asking for localization information, sending an SOS signal in emergency cases, requesting evacuation instructions,...
Most of the traditional services could be used to compromise the network, but this is not the case with the suggested Lightweight Services. If we go through the list of the most dangerous threat to Wi-Fi networks (mentioned in the COLDE concept), it will be as follows:

- **Local network security:** there is no need to establish a connection between the client and the access point, so the client will never be part of the network.
- **Public network security:** the access point can maintain a list of secure services and their servers, so the client will not be able to participate in any illegal action.
- **Service level:** this category of services consists of two packets only, the request and the response, so the client can never abuse the service by downloading large files, watching live broadcasts,...

The combination between the Lightweight Services and the Connectionless Data Exchange can provide the owners of the private Wi-Fi with a secure method to run their access points as a Lightweight Services Provider.

6.5.1/ SYSTEM ENTITIES

There are four entities in the service model. They cover the service PDU format, LightWeight Server, LightWeight Services Helper and the service beneficiaries.

- **Lightweight Service (LW-Service) PDU** is defined by each service individually.
- **LightWeight Server (LWS)** is the server that provides the lightweight service.
- **LightWeight Services Helper (LWSH)** is the entity that provides the service, or it relays the requests to the LWS and sends back the responses.
- **LightWeight Service Beneficiary (LWSB)** is the entity that needs a piece of data from the LWS but it is not able to get it directly without the help of a LWSH.

6.5.2/ SYSTEM DESIGN

The main points which have been taken into consideration in the LightWeight Services design are the following:

- The LWSH and the LWSB should not need to establish a connection between them.
- The Wi-Fi access point can refuse requests to certain servers or for certain services.

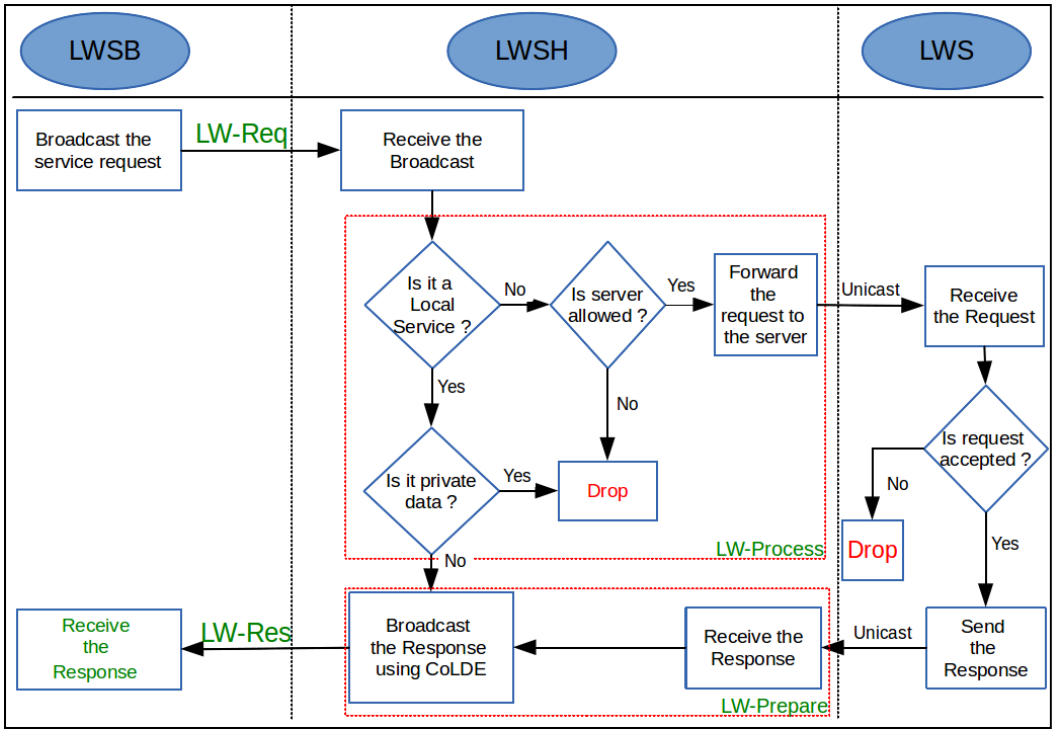

Figure 6.12: LightWeight Service Mechanism

- The client does not have to be connected to the Wi-Fi in order to ask for this service.
- The data could be gathered locally on the LWSH or could be relayed to a LWS.

6.5.3/ SERVICE MECHANISM

The service consists of two or four phases, according to the type of service:

- **LW-Req (LightWeight Service Request):** the LSWB broadcasts this request and waits for a response from a LWSH. In case it receives no response, it re-broadcasts the request after waiting for a pre-defined delay. This delay should be pre-defined in the protocol in order to prevent service abuse.
- **LW-Process (LightWeight Service Processing):** the LWSH processes the request as follows (Figure 6.12):
  - Verifies that the request is allowed according to its own list.
  - If the service should be provided locally, it verifies that it is allowed to send such data.
  - If the request should be relayed to a LWS, it verifies that the LWS is allowed on its own list, and then it relays the request. The UDP is used for the communication between the LWSH and the LWS.

- **LW-Ready (LightWeight Service Ready):** as soon as the LWSH receives the response (either locally or from a LWS), it prepares the response in order to send it to the LWSB.
- **LW-Res (LightWeight Service Response):** the LWSH sends back the result to the LWSB (Unicast).

6.6/ COLDE SECURITY

Despite that COLDE is designed to exchange non-confidential data, some rules should be fixed to prevent abusing the services by users and modifying the SOS information by an intermediate device/person. The rules should be able to:

- prove that a request (ex. SOS request) sender is who is claiming to be.
- proves to a user that the received response hasn't been modified by any intermediate device/person (ex. Evacuation directions).

It is important here to mention that confidentiality of information isn't required because the exchanged data should be public, non-sensitive or personal data. For example, no commercial transactions should be involved.
The main problems in providing a security model for such system is the limited size. Digital signature algorithms such as RSA (Rivest Shamir Adleman), Diffie-Hellman and Digital Signature Algorithm (DSA) need key sizes longer than most of the data that could be exchanged using COLDE. In addition to the limited size, using digital signature algorithms on some mobile devices could not be the best scenario because of hardware limitation.
Our solution for a security model depends on a One-Time Password authentication system (OTP) and hashing functions. Following, we will discuss briefly these two techniques and the security model provided for COLDE.

A one-time password is a password that is valid for only one login session or transaction. A secret pass-phrase is used to generate a sequence of one-time (single use) passwords. With this system, the user's secret pass-phrase never needs to cross the network at any time such as during authentication or during pass-phrase changes. Thus, it is not vulnerable to replay attacks [Haller et al., 1998]. There is two main standard for generating One-Time Passwords: HOTP and TOTP.

- **Hmac-based One-Time Password algorithm (HOTP):** HOTP depends on Hashed Message Authentication Code – HMAC. it uses cryptographic hash function SHA-1 in combination with a secret key. HOTP uses counter value which encrypts with the HMAC [M'Raihi et al., 2005]. HOTP passwords can be valid for an unknown amount of time.
- **Time-based One-Time Password algorithm (TOTP):** TOTP is the time-based variant of this algorithm, where a value T, derived from a time reference and a time step, replaces the counter C in the HOTP computation. TOTP passwords keep on changing and are only valid for a short window in time [D. M'Raihi Verisign, 2011].

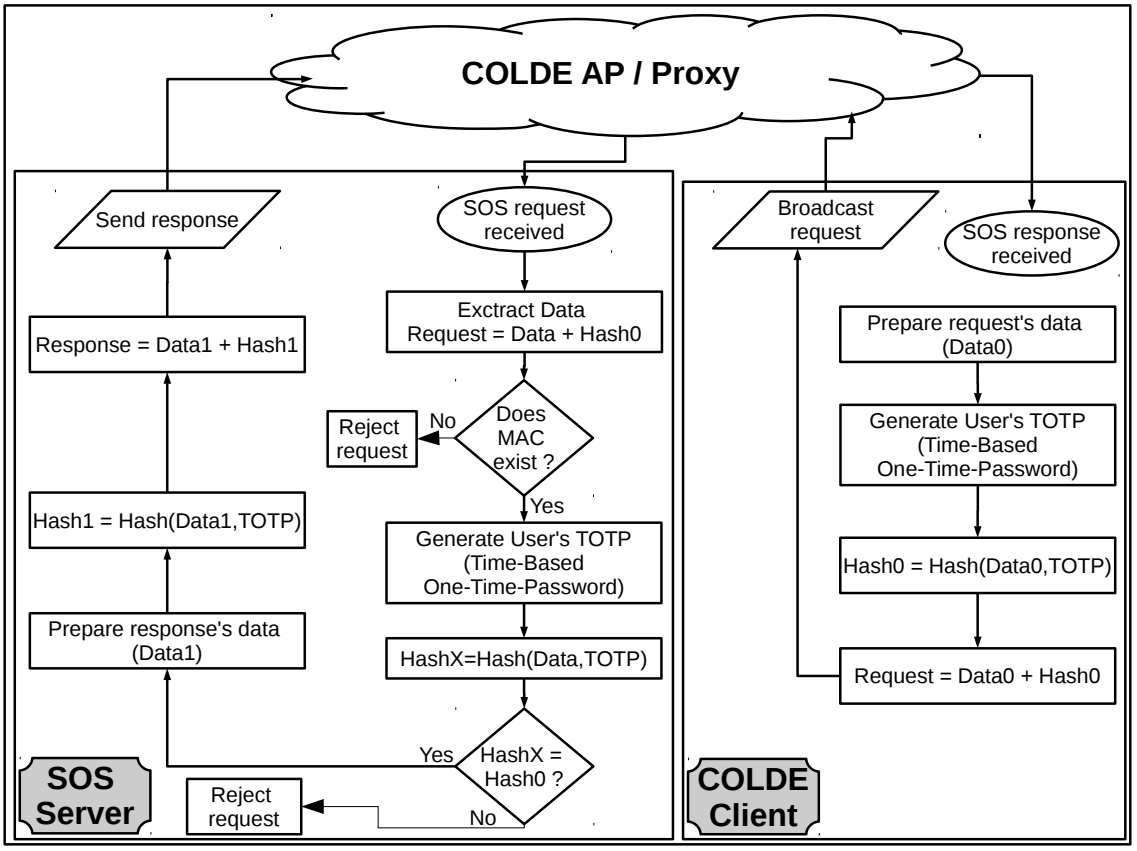

Figure 6.13: SOS Service Security

The TOTP is considered as a more secure One-Time Password solution because of the possibility to control the validity period.

A Hash Function is used to map a message of any length to a data of fixed size. There are many algorithms used nowadays for applying hashing, such as message-digest algorithm (MD5), Secure Hash Algorithm family SHA-0, SHA-1, SHA-2, etc.

The proposed security model depends on having a security mechanism for the critical services, such as SOS and evacuation services. The mechanism consists of two phases, Registration Phase and Verification Phase. Following, the mechanism is discussed by an example of SOS and rescue services. In registration phase, each device should be registered in the SOS database. The information would include the following: Owner's personal data, MAC address, Shared secret (Password), Time-Based One-Time-password function, Hash function, etc. The procedure of creating the database and collecting verifying the data is out of scope of this research. The client prepares the SOS request as follows (Figure 6.13):

1. It prepares the SOS data (Data0).
2. It generates a Time-Based One-Time-password (TOTP).
3. Using the hash function, it generates the hash string (Hash0) for the SOS data

(Data0) and the TOTP from the previous step.

4. The client broadcasts the request (Data0,Hash0) and waits for the response.

5. The client keeps sending SOS messages till it gets a valid response.

When a client broadcasts a SOS using COLDE, the COLDE-AP(s) in the same area (that received the request) forward the request to the registered SOS-Server directly, or indirectly by sending the request to a COLDE-Proxy server. The SOS-Server processes the request as follows:

1. It extracts the data (Data0) and the hash string (Hash0) from the request.

2. SOS-Server verifies that the MAC address is registered in its database. The request will be rejected in case there is no corresponding record.

3. Using the MAC address, it can generate a Time-Based One-Time-password (TOTP).

4. Using the hash function, it generates the hash string (HashX) for the original data and the TOTP from the previous step.

5. (Hash0 $\neq$ HashX) means that the request isn't originated from the user, or there is a time synchronization problem, SOS-Server will reject the request and finish the procedure. Otherwise it will continue to the next step.

6. SOS-Server sends the needed information to the concerned authorities. It prepares the response's data (Data1).

7. It generate the hash (Hash1) for Data1 and TOTP.

8. It forms the response by adding Hash1 to Data1 and sends it back the sender.

Following, we list some points and remarks to be taken into consideration:

- COLDE-client and SOS-Server should synchronize their clocks in order to generate the same TOTPs.
- COLDE-client can synchronize its clock using *Essential Services (0001)*.
- The password validity time would be tuned so the client and the server can generate the same TOTP.
- Handling the non-registered MAC addresses could be processed differently following to the environment and the situation (ex. natural disaster).
- A black-list could be prepared for the MAC addresses that send many non-valid requests.

The proposed mode can be customized in order to use different algorithms and rules. This model can be used for all types of lightweight services.

6.7/ CONCLUSION

We presented a new extension to IEEE 802.11 which is called CoLDE. It provides a simple and efficient method to exchange non-confidential and small amounts of data (only one frame) between Wi-Fi clients and APs without having any association. CoLDE loads the data in the IEEE 802.11 management frames directly which eliminates the complexity and the overhead of the network and transport layers. In order to organize data, we presented LightWeight services which could be transfered directly using COLDE. Another possibility is to use CoAP which has been designed by IETF to enable IoT and it depends on UDP. A design to prevent service abuse has been presented along with an example of emergency service which is a public service but it needs certain level of security.
In chapter 7, we present three COLDE-based applications which have been tested either in a real environment or in a simulation. The results of these experiments are discussed in detail.

7

COLDE IMPLEMENTATION

7.1/ INTRODUCTION

Experimentations have been carried out to validate COLDE model. The first experiment is a simulation to broadcast information in variable dense environment using COLDE. The experiment has been realized using NS2. After the promising results of the simulation, COLDE has been implemented in real environment and it has been tested with indoor positioning applications. The third application is a simulation to evacuate building in emergency situations by sending plans of evacuation by using COLDE model.

7.2/ BROADCASTING INFORMATION IN VARIABLY DENSE ENVIRONMENT

7.2.1/ INTRODUCTION

Broadcasting is a widely used communication mode in ad hoc networks. It allows sending an information from one node to all the nodes that are within its coverage area. This feature makes broadcasting a suitable mode for exchanging routing information in Mobile Ad hoc Networks (MANETs), sending emergency messages in Vehicular Ad hoc Networks (VANETs) or sharing local measurements in Wireless Sensor Networks (WSNs). Many studies of these networks tackled the broadcasting issues. They tried to handle the adaptation to density, the reduction of useless redundant packets, the guarantee of confidentiality and authenticity of broadcast data. Currently, one of the research topics is the design of a flexible method to broadcast information in variable dense environments. These environments consist of hundred or even thousands of clients in the same geographical area, where they can be connected to different networks. Therefore, one of the main challenges is to find a way to send broadcast packets to all nodes, no matter the network they are connected to, or even if they are not connected to any network. Adapting the ad hoc broadcasting algorithms proposed in the literature to work in variable dense environments brings out some considerations that must be taken into account, especially the ongoing services or communications of the nodes and the wideness of the area. For instance almost all these algorithms rely on the assumption that the nodes are connected to one network. This implies that if a node N_i wants to send a packet to another node N_j connected to a different network, at least one of them should disconnect from its original network. Indeed, normal Wi-Fi clients can be connected only to one network at the same

time. This disconnection can be a problem to many users because they will have to stop using the service of their main network. In wide areas, it is usual to find users that are geographically close, but connected to different networks. But in case of emergency for example it will be beneficial if they cooperate and exchange/forward safety messages. Finding a method that allows this type of communication is the main target of this work.

In this section, we present a new solution to use a multi-tier broadcast model to deliver messages to all the devices in a selected area by using COLDE.

7.2.2/ SIMULATION

For our simulations we have used the network simulator NS2 (version 2.35). We have implemented COLDE extension using C++ language, then we modified the management frames (beacons, probe request/response) of the protocol IEEE 802.11.
The simulation environment had the following characteristics:

- **Simulation area**: a square area of dimensions X*X (different according to the different scenarios,).
- **Number of nodes**: different according to the simulation area.
- **Nodes' locations**: the nodes are distributed over the selected area in which each node is located at fixed distance from the other nodes.
- **Propagation**: Shadowing Model.
- **Mobility**: the nodes are fixed during the simulation.

We have applied the testing according to two modes:

- **Directed Mode**: in this mode we have the root (the broadcasting node) and the client nodes, all the client nodes have direct connection with the root.
- **COLDE mode**: in this mode we have the following nodes:
  - The root: This is the broadcasting node, it is located at the center of the simulation area (1 root in each simulation).
  - MNL-I: main nodes have direct connection with the broadcasting node (4 MNL-I in all the simulations).
  - MNL-II: main nodes have no connection with the root (number of MNL-II nodes varies depending on the testing area).

  The broadcasting node sends a direct message to the main nodes MNL-I, the MNL-I nodes broadcast the message into their coverage areas using COLDE. The message received by MNL-II, the MNL-II nodes decrease the TTL and rebroadcast the message into their coverage areas.

Both modes have been repeated using different areas and different number of nodes. In each scenario we calculated the time needed for the nodes to get the message. Table 7.1 and Figure 7.1 show the parameters of each scenario and their results.

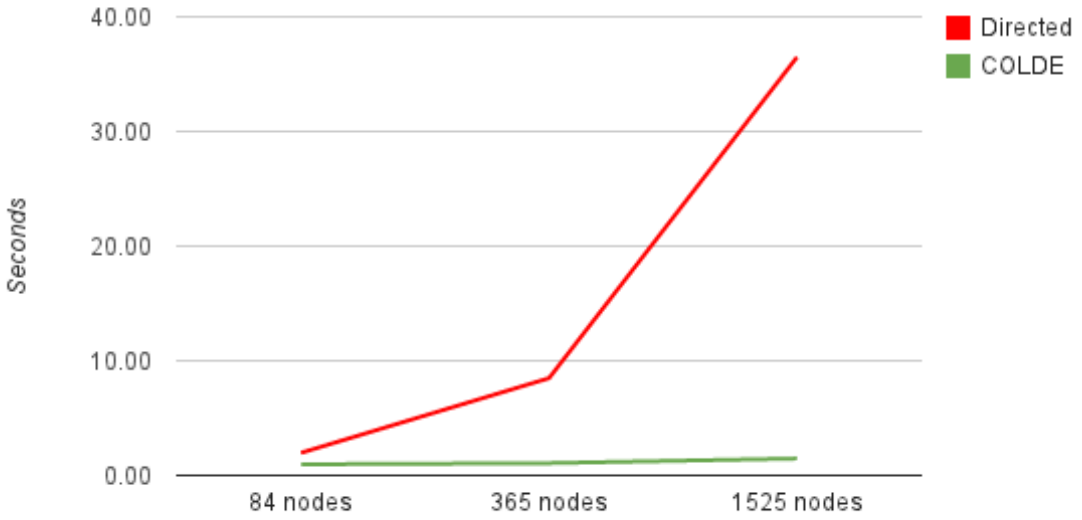

Figure 7.1: Simulation - Broadcast Duration Comparison

7.2.3/ Experiments and Results

The results show that COLDE can broadcast a message to about 84 nodes in half of the time needed by the Directed mode. In the second scenario, broadcasting the message using COLDE took 12% of the time needed by the Directed mode to broadcast the same message. In the third scenario, COLDE needed 4% of the time needed by the Directed mode. In other words, the Directed mode needed about 24 times the time needed by COLDE. Figure 6 shows the relation between the number of the nodes and the time needed to broadcast a message using the Directed mode and the COLDE mode.

7.2.4/ Conclusion

Exchanging data between the devices directly, without the need to have a connection and without being connected to the same network, provides a flexible method to broadcast important messages to all the devices in a selected area. Using a multi-tier broadcast model can extend the broadcast area which means that the message will reach a larger number of devices.

7.3/ Integration in Embedded System

COLDE proposes a general structure frame which can contain any type of data (Figure 6.9.). Implementation has been carried out as follows. Firstly, we use the vendor-specific information element (Figure 6.8.). Because of the extensive importance and to allow

Scenario			Directed		COLDE	
Area m*m	**Number of nodes**	**Broadcast time**	**Last Node**	**Duration**	**Last Node**	**Duration**
250*250	84	2.54	4.51	1.97	3.51	0.97
500*500	365	2.54	11.02	8.48	3.61	1.07
1000*1000	1525	2.54	39.03	36.49	4.01	1.47

Table 7.1: Simulation scenarios

some flexibility to the vendors, the 802.11 standard itself has a provision to carry non-standard, vendor-specific information in the "vendor specific" Information Element (IE) field of management frame. This IE (with ELEMENT ID 221) is provisioned to be always present as a last IE in the frame body of beacon. Using it, up to 251 bytes of information can be embedded in each management frame [Gupta et al., 2012a]. Utilizing the vendor specific information simplifies the implementation, many Wi-Fi cards' drivers identify and send this information element to the application for it to be processed. Each vendor has its own OUI (Organizationally Unique Identifier). OUI is a 24-bit number that uniquely identifies a vendor, manufacturer, or other organization globally or worldwide. The following byte (OUI sub-type) is used as a vendor-specific sub-type.

The time needed for processing a request is a serious bottleneck. While the probe request has a timeout measured by hundred of milliseconds, Request could need seconds to find the right response (such as, calculating the client position). In normal cases, the active scanning process takes 2 to 3 seconds [Adya et al., 2004a]. This time varies depending on the distance from the server's network, the link speed with this network and the server's capacity. In other words, when the client sends a probe request containing a request, the timeout can be hit even before receiving the probe response. The same problem will arise when trying to send another request.
We present our solution which utilizes two levels of caching. By using the cache, the client will get the result of its request in the following probe response frames.
We will cover the caching levels and the procedure followed in detail.
There are three main components in our solution. The Wi-Fi client device, the Wi-Fi access point and the COLDE Proxy Server (COLDE-Proxy).

7.3.1/ Integration Into the Wi-Fi Client Devices

The probe request is the main item needed to integrate client's COLDE in the Wi-Fi client's devices. The probe request is used to perform "Active Scanning".
In active scanning, probe request frames are transmitted on all the channels. The responses received from APs in the form of probe response frames are then subsequently processed by the WNIC (Wireless Network Interface Card). Active scanning is the default-scanning technique for a WNIC, which enables it to implore an immediate response from an AP, without waiting for the beacon frames to be sent by the AP [Gupta et al., 2007].
The client prepares the COLDE-Request frame. The frame will be added as an IE to be sent in the next probe requests (only one time). Every service can customize the COLDE-Request to include the needed data. The COLDE-Proxy can call the functions based on the service ID.

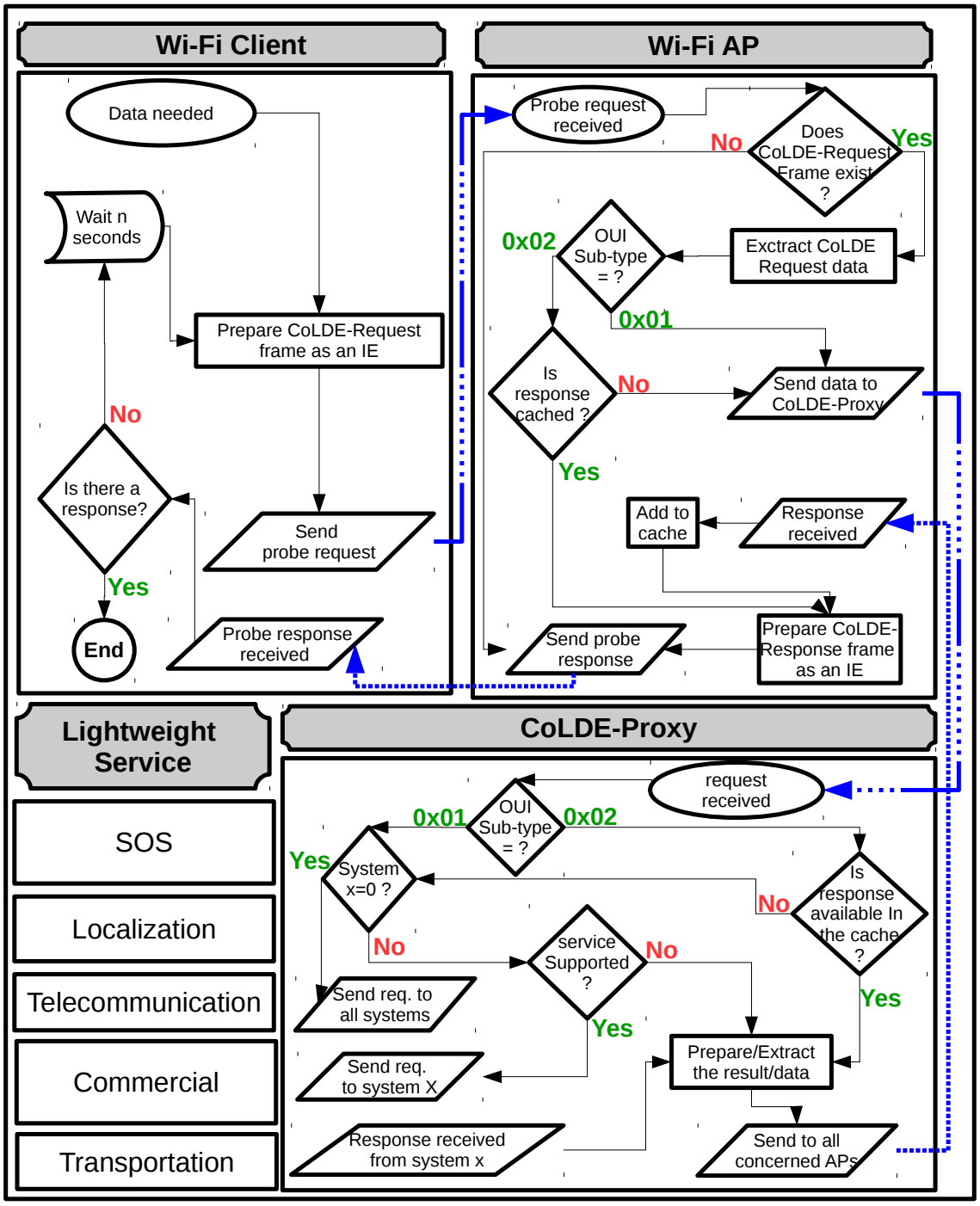

Figure 7.2: COLDE - Procedures and communications

The iw is a new nl80211 [Linux, 2015] based on Linux CLI (Command Line Interface) configuration utility for wireless devices. It supports all new drivers that have been added to the kernel recently. It is an open-source library. We have used iw to add the COLDE-Request as an IE in the probe requests, and to get the IE from the probe responses. As Wi-Fi clients, we are using an IPC (Internet Personal Computer) with Ubuntu and Arch linux operating systems, and a low-cost,compact Raspberry Pi with Arch linux.

7.3.2/ Integration Into the Wi-Fi Access Points

Access points play an important role in our solution. They have more tasks to do than just forwarding the frames from/to clients. We used Raspberry Pi as an access point. It has been equipped with Wi-Fi USB dongle. For the sake of testing we used three types of Wi-Fi dongles (Atheros, Ralink and Realtek). Arch linux (customized distribution for ARM architecture) has been used instead of Raspbian (based on Debian), the reason is related to the ability to control the Wi-Fi dongle driver by a 3rd-party application instead of the operating system itself.
The hostapd [Hostapd, 2015] has been selected as an access point application. Hostapd is a user space daemon for access point and authentication servers. It implements IEEE 802.11 access point management and it supports Linux (Host AP, madwifi, mac80211-based drivers) and FreeBSD (net80211). Hostapd is designed to be a "daemon" program that runs in the background. It can be distributed, used and modified under the terms of a BSD license.
The hostapd has been customized to process COLDE frame. We used the code 0x0c01de (it has not yet been assigned to any vendor) as a temporary OUI, we will refer to this ID as COLDE-OUI. Two sub-types have been added to further indicate the cache settings. The value 0x01 indicates that the client prefers to have instantaneous response (not from the cache). On the other hand, the value 0x02 shows that the client accepts cached response. Other values can be identified (254 values). These values can help to specify other settings. Whenever the hostapd receives a probe request with a COLDE OUI in the IE part, it extracts the COLDE-Request data. Then, it checks the OUI sub-type to decide whether it should check the local cache or not. In case of having the value 0x02, the AP searches the cache for any response stored for this client. If any response is found in the cache, the AP sends it directly in a probe response, otherwise it continues the procedure as if the sub-type were 0x01. If the OUI sub-type is 0x01, the AP forwards the data to COLDE-Proxy as a unicast. Hostapd prepares the COLDE-Response frame as soon as it receives the response from the COLDE-Proxy. It adds the COLDE-Response to a probe response to be sent to the client. The probe response has the same COLDE-OUI as in the IE. Each entry in the cache has a timestamp and a timeout. The timeout is measured in seconds and it can be customized to fit different networks and different lightweight services. It is possible to send the timeout from the COLDE-Proxy. The AP represents the 1st level of caching in our solution.

7.3.3/ Proxy Server (COLDE-Proxy)

COLDE-Proxy can be any linux or windows server, even it is possible to combine the access point and the COLDE-Proxy in the same device. Actually COLDE-Proxy is the interface between Wi-Fi clients and access points from one side, and the systems that provide the lightweight services from the other side. Such architecture facilitates the integration of COLDE in any system. COLDE-Proxy acts as a proxy for Wi-Fi clients. At the same time, it acts as a lightweight services cache (2nd level of caching) to speed up the process, to reduce the traffic with the system of the lightweight service (in case it is located on a different server) and to be a temporary backup in case of losing connection with the system. The possibility of using the caching varies depending on the system of the lightweight service itself.
COLDE-Proxy can work with different systems at the same time. COLDE-Proxy receives

the data from the AP(s), it puts them into the appropriate format (according to the selected lightweight service) and then it makes a call to the functions of that lightweight service. It sends the result back to the access point(s) that sent the data. This procedure is repeated every time a client sends a request. If the caching service is used, COLDE-Proxy checks the cache before starting the session with the system of the lightweight service.
(Figure 7.2.) summarizes the procedures of the three components in our solution and the communication between them.

7.4/ INDOOR POSITIONING USING COLDE

7.4.1/ INTRODUCTION

Indoor positioning using the IEEE 802.11 protocol has undergone considerable progress in the past decade. Indoor positioning became one of the essential technologies for many applications, such as disaster rescue, indoor navigation, and advertising. Indoor Positioning Systems (IPS) have been presented and implemented. These systems can be categorized into many groups according to their methods. One group is built on the use of the fingerprinting, which means a signature of environment features consistently and strongly depending on the physical location. This group has many categories according to the feature used. One category is time-based methods, these methods include Time-of-Arrival (ToA), Time Difference-of-Arrival (TDoA) and Round Trip Time (RTT). Another category is the angle-based method (i.e. AoA)[Zhang Da, 2010]. A third category uses (RSS) (Received Signal Strength) [Bahl et al., 2000]. These categories are the three most representative measurements for position estimation. Compared to ToA and AoA measurements, the RSS can be more easily measured without any additional special hardware devices in current open public WLAN networks [Tian et al., 2013]. The other group uses triangulation. It is virtually impossible to use this method without a significant error, because this method does not take into consideration the interference or the obstacles in the area, such as walls, furniture, and even other people in the building.
In general, indoor positioning needs a number of calculations which differs according to the methods used. There are two ways to perform these calculations. One way is to perform them on a mobile device, while the other way is to perform them on a server. Performing the calculations on a mobile device consumes the device's battery, and since mobile devices are normally battery-driven, energy efficiency is a very important consideration in Wi-Fi localization systems [Niu et al., 2013]. Methods such as Wi-Fi fingerprint-based localization solves part of this problem by sending the needed parameters to a server in order to perform the calculations.
Sending the parameters to a server requires the mobile device to have an active connection to this server, either by having an association with a Wi-Fi access point or using 3G/4G if the server is accessible from the Internet. Such a condition limits the usability of the positioning and localization services to the mobile devices that are connected to the right network where the server is accessible. On the other hand, GPS (Global Positioning System) is a system accessible by any person with a GPS receiver. Our goal is to provide the needed method to make indoor positioning accessible by any mobile device equipped with a Wi-Fi network card, without requiring it to be connected to any network, and regardless of the localization system used. For this purpose, we present our solution to exchange positioning and localization data using the Connectionless Exchange

	System*	Time	Last Position	Count	AP 1		....	AP N	
					MAC	RSSI		MAC	RSSI
				N					
Octet(s)	1	11	20	1	6	1	....	6	1
*System : it means the indoor positioning/localization system									

Figure 7.3: COLDE Indoor Positioning - Request Frame

Protocol (COLDE). COLDE uses the management frames in the IEEE 802.11 protocol to exchange small amounts of data [Abu Oun et al., 2014a].
The remainder of this section is organized as follows: *Section 7.4.2* presents the state of the art of this work and we survey related work in using connectionless protocols in positioning and localization. The positioning by methods and applications which can benefit by COLDE are discussed in *7.4.3*. Section *Section 7.4.4* provides the experiment scenarios and the results.

7.4.2/ Integrating COLDE in Indoor Positioning Systems

The customizable information element (COLDE-Request) (Figure 7.3.) provides the needed data structure for the methods that use RSSI fingerprinting, collaborative localization or time based methods. By using only one probe request frame, it is possible to send the last position information, the time (Figure 7.4.), the MAC address and the RSSI 31 of APs (the number of APs depends on the size of the *last position* field) [Abu Oun et al., 2014b].
The main fields of COLDE-Request are:

- **System** is a field of one byte with an unsigned integer, it specifies the ID of the positioning system that should process the followed data. System 0 means that there is no specific system, the data will be available for all systems. Subsequently, the user could receive no response, one response or many responses.
- **Time** is a field of 11 bytes (Figure 7.4.). The time is used in some positioning systems, such as OwlPS [Cypriani et al., 2009]
- **Last Position** contains the last position acquired by this client. It is a 20-byte length field. This field can be used to send the GNSS position (latitude: 8 bytes, longitude: 8 bytes and altitude: 4 bytes). Also, we can send a text or a code for relative location using all the bytes.
- **Count** is a field of one byte with an unsigned integer, it specifies the count of the data, in our case the count of APs gathered by the client, and the maximum number of APs is only 32.
- **AP MAC** is a field of 6 bytes containing the MAC address of the AP.
- **AP RSSI** is a field of one byte containing the RSSI, it indicates the power level being received by the AP.

Time, Length = 11 Octets									
	Year	Month	Day	Hour	Minutes	Seconds	Milli-Seconds	Hours ±UTC	Min ±UTC
Octet(s)	3	1	1	1	1	1	2	1	1

Figure 7.4: COLDE Indoor Positioning - Time Format

	System*	Time	Position	Message Size	Message
				Y	
Octet(s)	1	11	20	1	Y
*System : it means the indoor positioning/localization system					

Figure 7.5: COLDE Indoor Positioning - Response Frame

A different structure has been built for the response (COLDE-Response) (Figure 7.5.) The main fields of COLDE-Response are:

- **System** is a field of one byte with an unsigned integer, it specifies the ID of the positioning system that should process the followed data. System 0 means that there is no specific system, the data will be all available systems. Subsequently, the user could receive no response, one response or many responses.
- **Time** is a field of 11 bytes (Figure 7.4.). It specifies the sending time of the position.
- **Last Position** contains the position provided by the Wi-Fi access point or by the positioning system. It is a 20-byte length field. This field can be used to send the GNSS position (latitude: 8 bytes, longitude: 8 bytes and altitude: 4 bytes). Also, we can send a text or a code for relative location using all the bytes.
- **Msg Size** is a field of one byte, it contains the size of the Msg field.
- **Msg** is a variable length field (The *Msg Size* field specifies the size of this field). It is a free structure field, the positioning system or the AP can send a text message to the client.

COLDE-Proxy can work with different positioning systems at the same time (Figure 7.6.). Depending on the field *System* (COLDE-Request) (Figure 7.3.), COLDE-Proxy can decide the format of the data and to which system they should be forwarded to. COLDE-Proxy receives the data from the AP(s), it puts them into the appropriate format (according to the selected positioning system) and then it makes a call to the functions of that positioning system. It sends the result back to the access point(s) that sent the data. This procedure is repeated every time a client sends a positioning request. If the caching service is used, COLDE-Proxy checks the cache before starting the session with the positioning system.

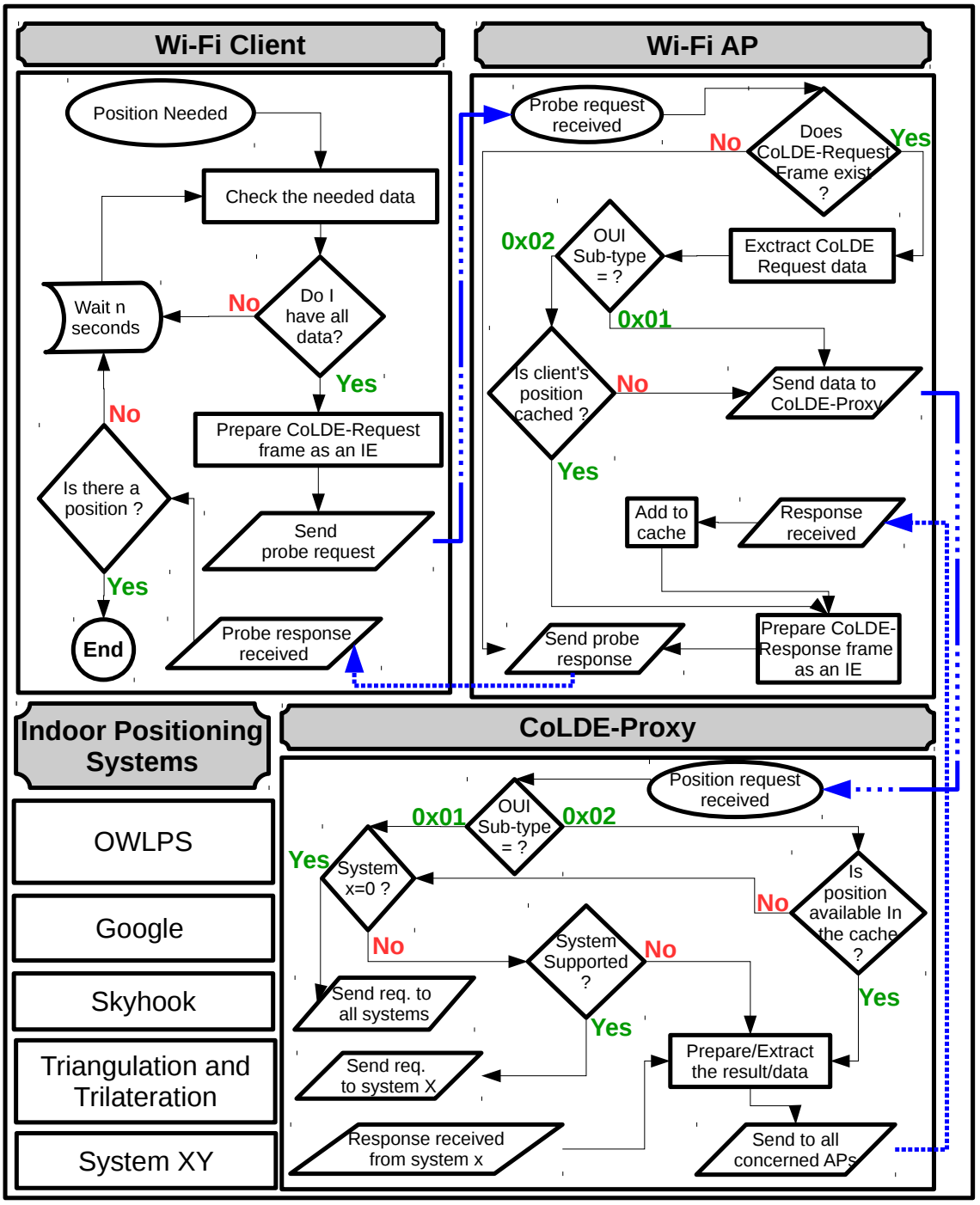

Figure 7.6: Indoor Positioning - Procedures and communications

7.4.3/ Related Positioning Methods And Applications

Below we discuss some algorithms and methods which have been developed for indoor positioning, at the same time we will mention how they could be improved using our solution for connectionless.

Wi-Fi Localization Using RSSI Fingerprinting
Wi-Fi Fingerprinting creates a radio map of a given area based on the RSSI data from several access points and generates a probability distribution of RSSI values for a given (x,y) location. Live RSSI values are then compared to the fingerprint to find the closest match and generate a predicted (x,y) location[Eduardo Navarro, 2010]. We take two examples of the applications that use fingerprinting methods.

Google Maps [Research, 2015] application can solve the problems of positioning navigation in open areas and indoors. It provides the possibility to calculate the route inside the building between several floors in addition to navigation [Ramani et al., 2014]. For orientation it uses Wi-Fi and cellular networks with positioning accuracy ranges from 5 to 10 meters [Alexey Kashevnik, 2012].

Skyhook Wireless [Wireless, 2015] is one of the main companies in the domain of localization and positioning. Skyhook is a multiple source location system that uses Wi-Fi, GPS and cell towers which should work better in cities where Wi-Fi and cell tower signals are highly present. Skyhook's Core Engine is a software-only location system. It uses a massive reference database comprised of the known locations of over 250 million Wi-Fi access points and cellular towers. Skyhook client software running on a Wi-Fi-enabled mobile device collects raw data from each of the location sources. The client needs to be connected on the Internet to send this data to the Location server.
It is possible to enhance these applications by using COLDE, the users will be able to broadcast the gathered information to APs, which will forward them to the location server of the company(s) that support(s) this area. The response will be sent to the access point and then back to the client in a probe response.

Owl Positioning System (OwlPS)
OwlPS implements several positioning techniques and algorithms (RADAR[Bahl et al., 2000], Interlink Networks [Networks, 2002], FBCM [Lassabe et al., 2005] and Basic FRBHM [Lassabe et al., 2006]), allowing to combine and compare them, even in a real-life experiment way [Cypriani et al., 2009]. The configuration where infrastructure executes all the processing needs several elements: mobile terminals equipped with Wi-Fi cards, access points or any capture device (listening for any positioning request transmitted by the mobiles), the aggregation server (which the APs forward the received positioning requests to) and the computation server (which computes the position of each mobile from information forwarded by the aggregation server) [Cypriani et al., 2010].
The system works as follows: the mobile airs a positioning request and capture devices (the infrastructure) capture it. This request consists of 10 to 20 UDP packets containing the local time. Each AP capturing the positioning request transmits it to the aggregation server along with additional data. The additional data consists of : the mobile MAC and IP addresses, the AP's MAC address, the time at which the packet was captured and the RSSI. The aggregation server gathers the data and forwards them to the computation server. The computation server analyzes the information received from the aggregation server and computes the mobile position. The computed position can be sent to the mobile by using direct connection between the computation server and the mobile.
OwlPS could use the COLDE Indoor Positioning Frame (Figure 7.3.), (Figure 7.4.) to exchange data.
By using COLDE, any Wi-Fi device can broadcast a probe request on a channel. The

APs capture the probe request, each will forward it to COLDE-Proxy along with the AP data. COLDE-Proxy will relay the data to the aggregation server. The latter will process the data and forward it to the computation server. The computation server computes the position and sends it back to the APs. The APs send back the position in the probe response.
Using COLDE would affect the mechanism of the system itself by adding the possibility to send the computed position to the mobile without having direct connection with it. Another advantage is the amount of data processed, while OwIPS depends on capturing and processing all frames (management and data frames), COLDE processes the management frames only.

Collaborative Indoor Positioning
This kind of approach is the opposite of the localization methods that depend on the infrastructure. The collaborative localization proposes a model where clients may act as reference points in addition to their role as clients. People-Centric Navigation (PCN) provides an indoor localization solution by using the clients themselves as reference points. Clients with the PCN (i.e., mobile phones) continuously obtain accelerometer and digital compass readings to estimate step counts and direction. They also estimate a vector of each step called step vector, using the direction information and stride length. Since the stride length varies between individuals, it is approximated from the body height. Clients also record RSS from neighboring clients, which is collected through the device discovery process of Bluetooth. Step vectors and RSS are transferred to a centralized server called a PCN server via 3G or Wi-Fi. Then the PCN server estimates relative positions among users and the results are sent back to the clients to give them estimated positions[Yamaguchi et al., 2012].
Using COLDE, PCN method can be enhanced to enable clients to broadcast their gathered data to the access point to be forwarded to the server without having any connection.

Emergency Evacuation
Emergency evacuation from buildings during catastrophic events need to be quick, efficient and distributed. Indoor localization with COLDE can be used to optimize this process. In some cases, people could be trapped inside the building because of some obstacles or because of being injured or having a certain disability. Indoor localization using COLDE can be used to have an updated database of mobile location tracking information. Indeed, they allow to determine the current location of the people present in a building. This permits rescue teams to find them by asking the approximate location.

7.4.4/ Experiments And Results

Our experiments and scenarios aim at demonstrating the improvement in indoor positioning systems by utilizing COLDE. We evaluate and quantify the performance of the protocol in a real and congested environment. In this environment, there are more than 20 APs on different frequencies, the APs handle hundred of requests, and the clients receive hundred of broadcasting frames.
The experiments have been conducted using the following parameters:

- **Indoor positioning system**: we used a centralized trilateration indoor positioning system. The system maintains a database of the APs MAC addresses and their coordinates. It computes the client position by utilizing the APs coordinates, and

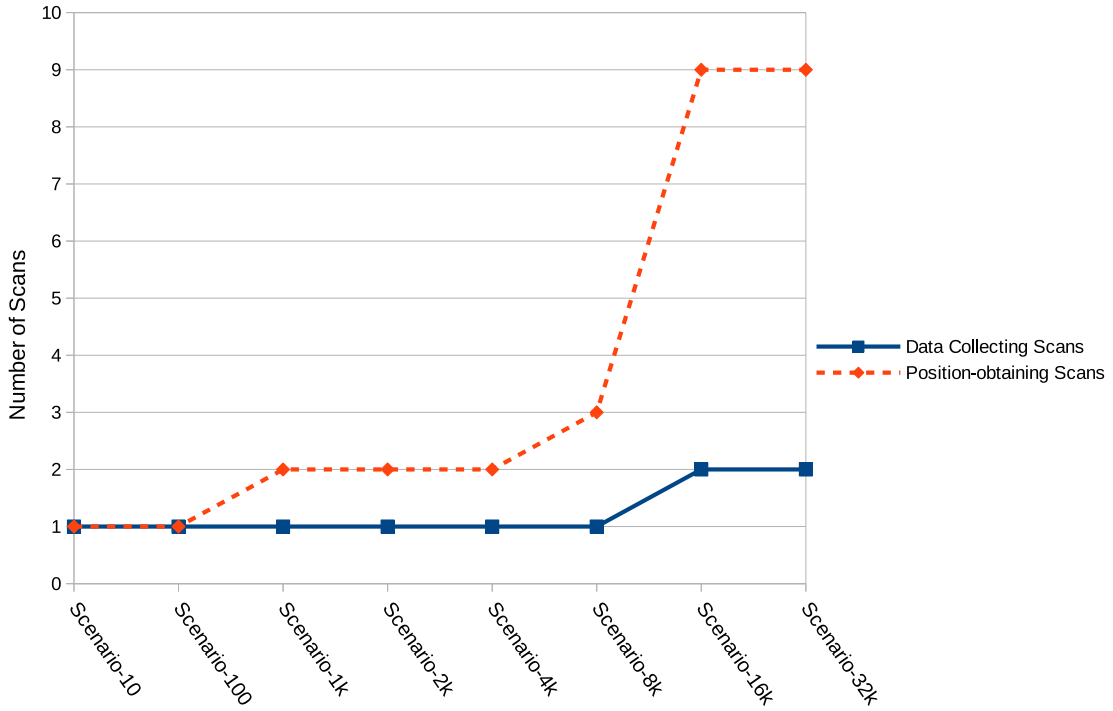

Figure 7.7: COLDE - Number of Scans

their RSSI. In our experiment, the trilateration indoor positioning system was running on the same server with COLDE-Proxy, so we eliminated the delay that could be caused by the network. The time needed to compute the position using the trilateration algorithm ranges between 5ms and 10ms. To simulate the delays in the other indoor positioning systems, we repeated the same trilateration algorithm with different delays. Each delay represents a different scenario, it simulates different indoor positioning system. We tested the following scenarios (the number after "-" is the delay in milliseconds):

  - **(Scenarios-Group-I)**: Scenario-10, Scenario-100, Scenario-1k, Scenario-2k, Scenario-4k, Scenario-8k.
  - **(Scenarios-Group-II)**: Scenario-16k, Scenario-32k.

- **COLDE-Client**: it is a Java application running on Linux. It utilizes the iw library (written in C). It uses the same architecture that we mention in Section IV. We customized it to collect the data needed for the trilateration indoor positioning system. A Wi-Fi client performs active scanning to collect the APs in its area and to send the data in a positioning request. The detailed functionality is described in (Figure 7.8.).
- **COLDE-Proxy and COLDE-AP**: we have implemented one COLDE-Proxy server and 3 COLDE-APs as mentioned in Section IV.

We installed the COLDE-APs in different location. We configured them with different SSIDs and on multiple frequencies. All APs are connected to COLDE-Proxy on the same

LAN. COLDE-Client is installed on an IPC.

COLDE	Probe	Size	Round-Trip Time
Without	Request	103 bytes	26 milliseconds
	Response	125 bytes	
With	Request	352 bytes	61 milliseconds
	Response	168 bytes	

Table 7.2: The effect of Adding COLDE into Probe Frames

In our experiment, we studied the following aspects:

- **Frame size**: we assessed the effect of adding COLDE data to the probe request/response frames. COLDE adds up to 255 bytes into the probe request/response frames. (Table 7.2) summarizes our results. It shows that, the round trip with COLDE needed about 53% more than the same trip without COLDE.
- **Data Collecting - Number of Scans**: number of active scans needed to collect the data. (Figure 7.7) shows the number of scans needed for each scenario. We noticed that in Scenarios-Group-I, a client needs to scan the network only once. For Scenarios-Group-II, a client needs to scan the network 2 times to collect the data.
- **Position-obtaining - Number of Scans**: number of active scans needed to obtain the position. (Figure 7.7) shows the number of scans needed for each scenario. We noticed that, a client needs to scan the network only once in Scenario-10 and Scenario-100, twice in Scenario-1k, Scenario-2k, Scenario-4k and 3 times in Scenario-8k. For Scenarios-Group-II, a client needs to scan the network about 9 times to obtain the position.
- **Data Collecting Time**: All infrastructure-based positioning systems require data sourced from various functions. For the data provided by the APs, We observed the time needed to collect this data using COLDE. This time includes: time needed to send probe request by the client, time to process the request by the AP and time to send the probe response to the client. In (Figure 7.9), we notice that data collecting time is about the same for Scenarios-Group-I. For Scenarios-Group-II, We found out that we needed double the time, because the client repeats the data collecting process after 5 failed tries, as described in (Figure 7.8.).
- **Position-obtaining time**: it is the time between sending the positioning request in a probe request, and receiving the position in a probe response. For both of the scenarios group, the position-obtaining time is the sum of two values: the time required to compute the position (on the server), and a value that ranges between 2000ms and 3000ms. This value represents the time needed to scan the network.
- **Position from the AP cache**: we observe the time when the client sends a second probe response, because it failed to obtain the position in the first one. In (Table 7.2), clients obtained their positions without caching in Scenario-10 and Scenario-100. For all other scenarios, the positions have been cached in APs, the clients obtained them in the next probe response frame.

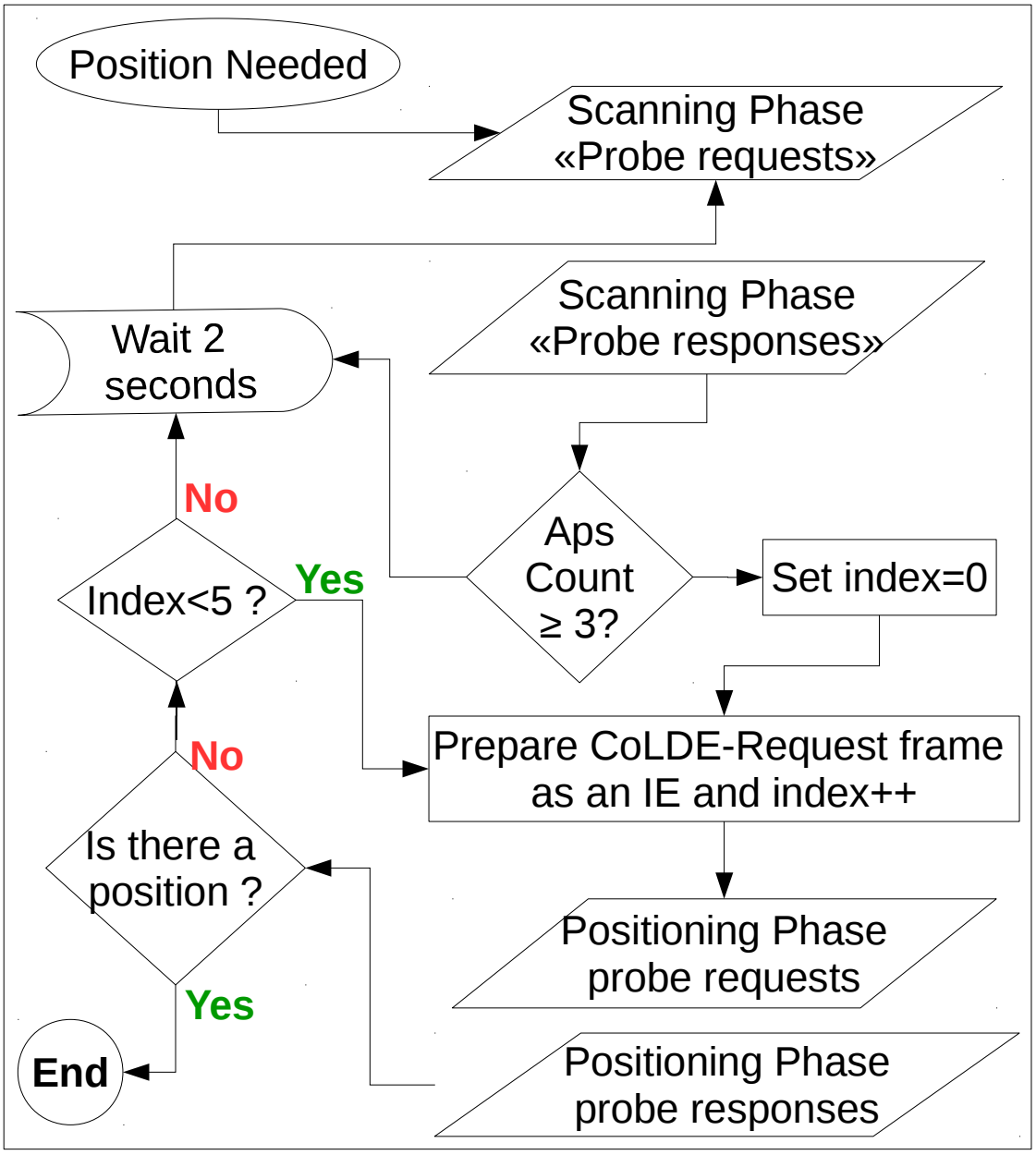

Figure 7.8: COLDE-Client - Trilateration

Experimental results (Table 7.2) prove that utilizing the management frames to send the data to the indoor positioning system does not change the protocol operation in any way. The delay resulted from adding data into the probe request/response frames is less than active scanning timeout. COLDE needs about 3 seconds to exchange the data between the clients and the indoor positioning system. The AP caches are used for all indoor positioning that need more than 100ms to compute the position. Using the AP's cache gives an effective solution for the problems could be caused by the timeout of management frames.

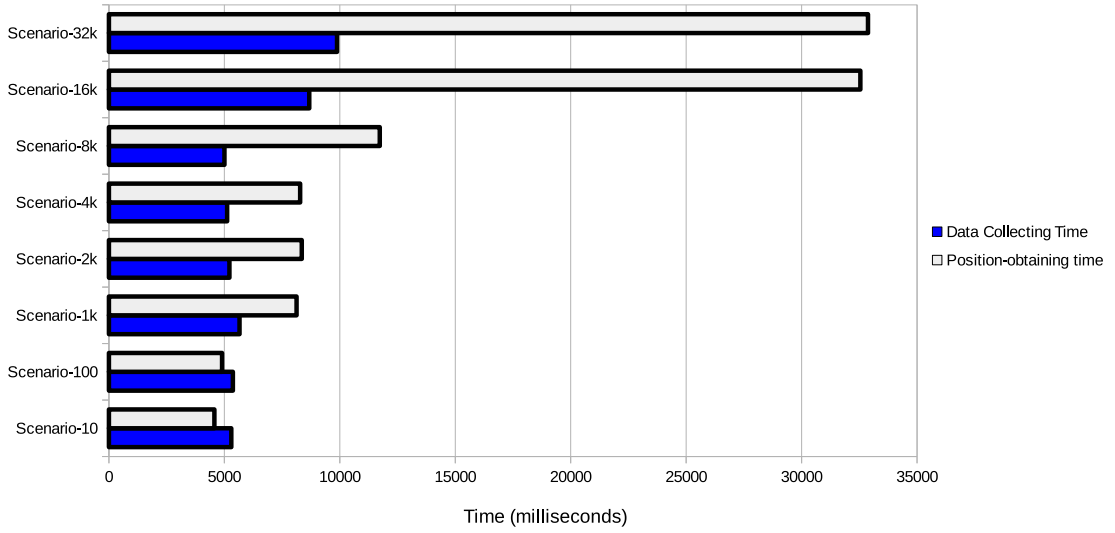

Figure 7.9: Scan and Positioning Duration

7.4.5/ Conclusion

We presented our solution to make indoor positioning functions available for public use. We proved that it is possible to change the way the service is provided by using CoLDE. We showed that CoLDE offers the needed mechanism for APs to be the bridge between Wi-Fi clients in their coverage areas, and the positioning systems that could be located on network different from the APs ones. We discussed the case where CoLDE can even improve some algorithms by providing them with more amount in data, where it would be impossible to have them using the traditional ways. We presented our component CoLDE-Proxy which is the interface between infrastructure devices and positioning systems. We showed that by using CoLDE-Proxy, clients in the same area can use different positioning systems, even without being connected directly to any of them. Integrating CoLDE into any positioning system does not interfere with the main functionality of the system, and it could even be transparent for the positioning system itself.
The experiments showed the ability to utilize CoLDE with a centralized trilateration positioning system. They proved the ability to have CoLDE working with indoor positioning systems that need more time to compute the positions, due to two levels of caching.

7.5/ Emergency Evacuation

7.5.1/ Introduction

The mobile phone inside the building could be located in any Wi-Fi zone. Selection criteria and optimization process should be applied to choose the most appropriate evacuation directions according to current position of each mobile phone [Abu Oun et al., 2013]. COLDE is an essential part of this solution. In the large building especially the public

ones, most people do not connect to the Wi-Fi access points in these buildings. Either because the access points are not public (limited to a certain group, or simply some persons do not need to use the network (ex. to not drain the battery). Using the COLDE to broadcast the alerts and evacuation messages can help in dealing with such scenarios without the need to deploy new public Wi-Fi networks inside the buildings. In some cases, people could be trapped inside the building because of some obstacles or because of being injured or having a certain disability. Thus, rescue teams need their exact locations inside the building in order to evacuate them.

7.5.1.1/ Using Wi-Fi to Broadcast Evacuation Directions

Broadcasting the evacuation directions using the Wi-Fi network can solve major problems which exist in the traditional ways, some of these problems are related to the people in the building in which some people could have certain disabilities preventing them from receiving the directions. Other problems are related to the state of building during the evacuation. For instance, having some blocked exits or dangerous corridors because of the fire. There are three different levels suggested for broadcasting the evacuation directions in a building using Wi-Fi:

- All the Wi-Fi access points in the building broadcast the emergency exits accompanied with their exact positions to all the mobiles. In this case the evacuation management system (if there is any) has no information about the persons who exist in the building and their approximate locations. Thus, each mobile is going to decide which exit is the closest according to the approximate distance between the mobile and the exit.
- Each Wi-Fi access point broadcasts the emergency exits which are located in the same range as the access point itself. As in the first level, there is no information available about the mobiles and the persons in the building and the mobile will decide which exit is the closest.
- Broadcasting customized directions to each mobile according to the position of the mobile and the building situation. The directions should be generated by the evacuation management system in the building, this solution works between three entities: the mobile phone, the access points, and the management server. The protocol depends on COLDE to exchange the data between the mobile phone and the access points without any association between them. This gives the ability to the mobile phone to stay connected to another network, while it is using the positioning and evacuation services of the building internal network. The access points relay the positions of the mobile phones to the server in order to keep updated snapshot of the mobile phones inside the building. Thus in evacuation time, the server will send the best evacuation plan for each mobile phone through the access point.

7.5.2/ Simulation Experiments and Evaluation

7.5.2.1/ Experiment Design

During this study, multiple scenarios have been simulated so as to measure the time needed to evacuate a building by following the evacuation directions which have been

sent using the Wi-Fi broadcast after the emergency alarm is activated. The simulation is done using NS2 equipped with the "Shadowing Patterns" model, many variables are taken into consideration, these variables could be categorized into three main groups:

- **Building structure:** Building dimensions, positions of emergency exits, capacity of emergency exits.
- **Network structure:** Wi-Fi access points and their positions.
- **Population:** Number of persons in a building, their initial positions, the initial target coordinates and speed.

7.5.2.2/ EXPERIMENT POLICIES

Following we discuss the policies used during the simulation:

- **Person movement policy:** A person moves from its initial position to its target in straight line. When it receives the evacuation broadcast along with the available evacuation plans, it stops moving and evaluates all the plans according to the distance between its position and the emergency exit of each plan. Then it starts moving in a straight line toward the closest exit. At the exit it joins the waiting queue to exit the building.
- **Initial person position, Initial person target:** Random functions could cover the whole area of the building or a certain side of it.
- **Person speed, Evacuation speed:** Random functions give different values for each person.
- **Exits positions:** All the emergency exits are located in the external walls of the buildings.
- **Access points:** Distributed to cover the whole area.
- **Exits Capacity:** Statistics consist of time of first and last persons arrived, time of first and last persons evacuated.
- **Evacuation performance:** Statistics contain the summary of all the exits and the evaluation of the evacuation process.

7.5.2.3/ EXPERIMENT SCENARIOS

Utilizing aforementioned policies, we test and analyze three different scenarios, each scenario has been applied ten times according to the following criteria:

Scenario	Area	APs	Exits	Persons	Distributions
1	20m * 20m	3	2	25	100%, 75%, 50%
2	40m * 40m	3	2	50	100%, 75%, 50%
3	60m * 60m	3	3	150	100%, 75%, 50%

Table 7.3: COLDE Evacuation - Experiment Criteria

7.5.2.4/ Experiment Results

In all scenarios, when the persons have been distributed over the whole building (Distribution over 100%), exits occupation was almost the same. In fact, when we monitored the time evacuation of the last person, we got similar times. It is not the same case when we tested the same scenarios with the same parameters with a distribution over 75% of the building area. The result in this case have been changed completely. In the scenarios with three exits, 50% of the persons have been evacuated using one exit and the other two exits evacuated the rest, whereas in the two exits scenarios, 75% of the persons have been evacuated through one exit. Therefore the total evacuation time has been increased about 15% comparing to the same scenarios when we used the distribution 100%. The worst evacuation time was noted when we applied the same scenarios with distributing the persons over 50%. The total evacuation time has been increased by 150% comparing to the first test. In the scenarios with three exits, more than 90% of the persons have been evacuated using one exit. Considering that the reason of the evacuation is an earthquake and we have only few minutes to evacuate the building. If we consider the time needed to evacuate the persons from the building is equal to the evacuation time that we got using the ideal distribution "Distribution over 100%". That means only about 33% of the persons in the building will be able to leave it in the right time when they are distributed over 50% of its area. We conclude that the reason is a bad load balancing because of choosing the evacuation plan by people individually. They picked their decisions depending on the distance between their current positions and each exit, and they ignored current situation of the building. By analyzing these results, we found that it was possible to evacuate about 25-30% of the persons who couldn't leave in case they used one of the other exits, especially for the persons who where nearly at the center of the building between all the exits.

7.5.3/ Real-World Experiments

This experiment depends on testing COLDE-based evacuation by using CEMAT architecture. In the following, we describe this experiment in details.

7.5.3.1/ Evacuation System Design

We designed a client-server evacuation system. The system works as follows:

- Each evacuation server should have a fingerprints database for the building which is covered by evacuation system.
- Each COLDE-AP broadcasts continuously the map of its floor either by using COLDE or beacon-stuffing. The maps on the APs could be loaded from the server.
- The Wi-Fi clients save the received maps so they will have them ready in case of an emergency.
- Each Wi-Fi client scans for the Wi-Fi access points, GSM towers and GPS coordinates (fingerprint), it sends the scan results to the evacuation server using COLDE.
- The evacuation server collects these results and localize the Wi-Fi clients inside the building and sends back the to the clients the directions for the closest exit.

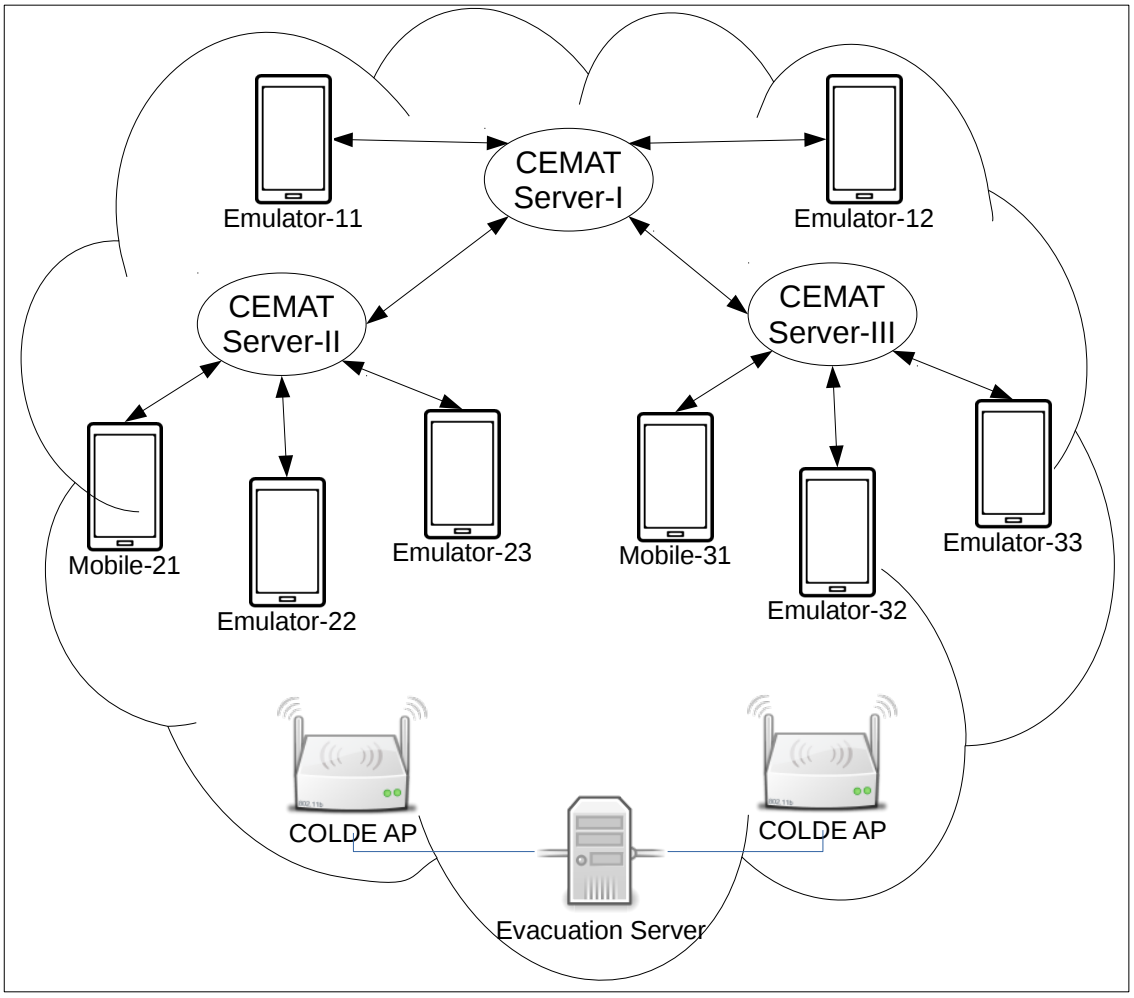

Figure 7.10: Real-world Experiment Architecture

- The evacuation server traces the Wi-Fi clients and keeps an updated view of the building.
- Whenever an emergency event takes place, the evacuation server generates evacuation plans depending on the distribution of the persons and the current situation of the building.
- At worst case scenario, if the whole network went down for any reason, each Wi-Fi client would have the map of the floor in addition to the directions which lead to the closest exit.
- Each evacuation server might have a mirroring server installed in a data center accessible by the authorities. So in case of losing the connection with the principal server at the building, rescue teams can access the mirroring server in order to evaluate the situation and put plans to evacuate the persons from the building.

The proposed design provides a fully redundant evacuation system depending on COLDE.

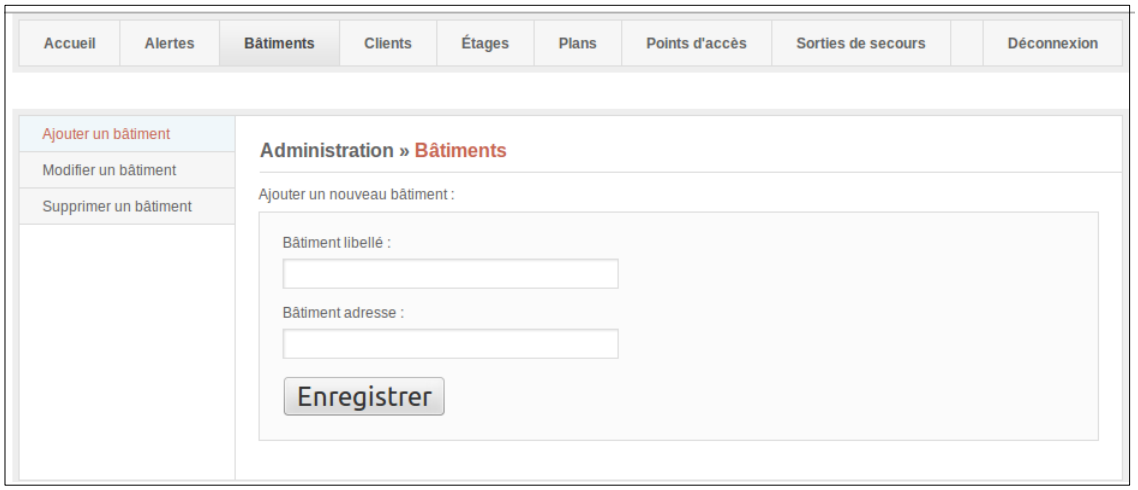

Figure 7.11: Evacuation Server

7.5.3.2/ System Implementation

In order to implement the proposed design, we developed the following items (Figure 7.10):

- **COLDE-AP:** The implementation has been presented in subsection 7.3.2. COLDE-APs will work as proxies in order to bridge between the clients and the server.
- **Evacuation Server:** It consists of three components which are as follows:
  - **Processing Component:** is a Java application which is responsible of receiving and processing data, generating and sending evacuation plans.
  - **Manager Interface:** is PHP interface which could be accessed remotely. This interface is used to configure the building, to load fingerprints and to customize evacuation plans (Figure 7.11).
  - **Data Storage Manager:** is the XML-based storage which includes all data received from the clients, the fingerprints, evacuation plans. Using XML-based database provides a high level of portability between the systems of evacuation.

  The three components could run on the same server or they could be distributed over several servers.
- **Evacuation Client:** It consists of the following components:
  - **Environment Scanner:** It is responsible of collecting data from the environment, such as: scanning for Wi-Fi access points, retrieving GSM tower data, checking GPS coordinates. The scanner organize data collected in a XML file in order to be sent to the evacuation server.
  - **User Interface:** It shows data alerts received from the server, maps received from the access points and guides the client to the closest exit (Figure 7.12).

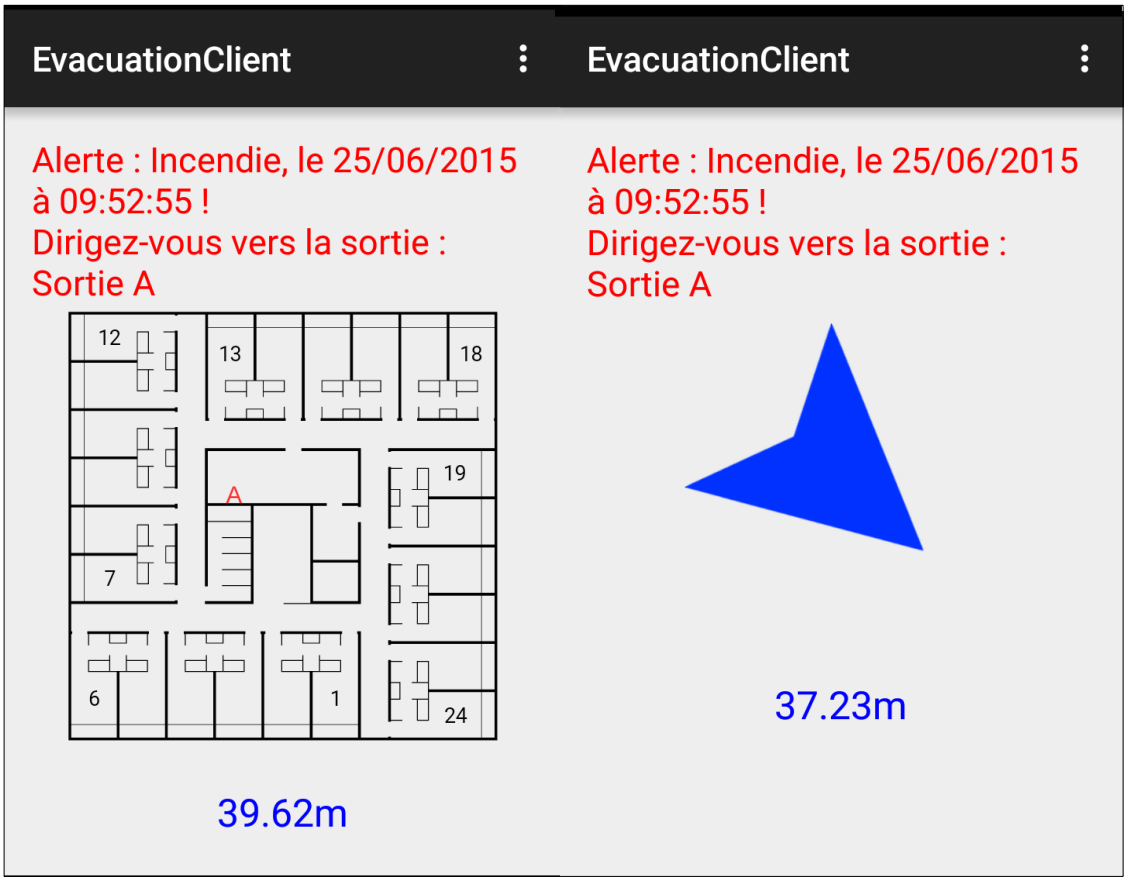

Figure 7.12: Evacuation Client

– **COLDE component:** It is the component which is capable of sending/receiving data to/from the access points by using Wi-Fi management frames. This component requires integrating COLDE code into the Wi-Fi driver of the mobile phone used for testing. Here we should consider the following facts: 1) Android Emulators don't support Wi-Fi signals because they depend on virtual machines, which means that it is impossible to manipulate Wi-Fi management frames by an emulator. 2) In order to integrate COLDE code into an Android mobile phone, the customized version of Android operating system for this phone should be modified, then it should be recompiled and reinstalled on the mobile phone. This procedure required access to the customized operating system source code which isn't available for all phones, in addition to *root* access to the mobile phone itself which isn't given by default.

Google Enterprise is the responsible of Android OS source code. Because of these difficulties and since it is possible to send a change request to Google in order to add this code in their next version, we developed the communication between clients and the server by using UDP which could be a proof of concept for the proposed evacuation system. The advantage of using UDP that it is possible to test the application on emulators since it doesn't need to manipulate Wi-Fi management frames.

7.5.3.3/ Experiment Scenario and Results

The experiment uses the architecture shown in Figure 7.10. The goal of this experiment is to test the feasibility of our system for evacuation using on several mobile phones and emulators at the same time. The experiment has been carried out as follows:

- Preparing three CEMAT servers and connecting them in a tree architecture.
- Launching 6 GenyMotion emulators (2 emulators on each *CEMAT server*).
- Adding 2 mobile phones (one mobile phone to each of the servers *CEMAT Server-II* and *CEMAT Server-III*).
- Installing *CEMAT Scenario Launcher* on all emulators and phones.
- Installing *Evacuation Client* on mobiles.
- Installing *Evacuation Client - Simulation* on emulators. Mobile phones can collect the information from the environment which is not the case for the emulators. This version depends on reading environment information from an XML file instead of collecting it. This version has the floor maps pre-installed.
- Preparing the scenarios files for emulators and mobile phones. Scenario files for emulators include simulated environment information which should be sent by emulators.
- Tracing the clients in the building and sending the closest emergency exit for each of them.
- Launching the experiment by triggering an emergency event and requesting an immediate evacuation.

The evacuation server starts tracing the clients by using the received fingerprints. It replies with the closest emergency exit. As soon as the emergency event is triggered, the server generates evacuation plans for each client. For the clients which have the floor map, they could navigate in the building depending on this map. It is important to notice that clients should be able to get their positions from the server all the time (Figure 7.12 - left side image). In case of losing the connection with the server or not having the floor map, the client will have an indicator to show the direction from its location to the emergency exit. In this case, the direction will be a straight line in which the client should find the exact route (Figure 7.12 - right side image).
This experiment proves the possibility to evacuate a building using person-based and customized-based evacuation directions. It also proves the possibility to use CEMAT in testing applications in heterogeneous environments by generating customized scenarios. The performance of the proposed evacuation system would be measured as soon as we received the Android version with COLDE extension.
It is possible to enhance this system by using auto-calibration [Spies et al., 2015]. By using auto-calibration, the access points could intercommunicate in order to update the fingerprints so as to reflect any change in the environment. This feature would simplify installing the system in changeable buildings.

7.5.4/ CONCLUSION

These experiments proved that broadcasting the alert messages and evacuation directions inside the building, in which each person/mobile can choose the best plan according to the distance to the exit, can be useful only in the ideal situation where the positions of the persons cover the whole building, and there is no diversity in the density in the building, which is not the case most of the time. In most of the cases, the person positions will be concentrated in certain places inside the building. Therefore, in case of the evacuation, they will line up in front of the closest exit waiting for their turn to leave the building, whilst the other exits are empty. Note that the other exits are far but the persons would leave faster using them. In these experiment, we supposed that each mobile phone represents just one person, in the real situations and in most of the case we will find that one group of two or three persons or even more will use the same evacuation directions of one mobile phone. The solution is to use COLDE to monitor the situation of the building and the persons inside it in order to generate the possible evacuation plans for each mobile phone as soon as the evacuation process is triggered. Evacuation plans should be generated depending on the current situation of the building and it could generate new plans in case of any update as it was explained in the real-world experiment.

IV

Conclusions and Perspectives

8

IoTaaS

8.1/ Conclusion

We presented IoTaaS which is an abstract architecture of a testing environment for IoT applications and systems. IoTaaS is a distributed environment which can work in a centralized/decentralized manner. Several IoTaaSs could be integrated in one cloud in two forms: peer-to-peer or tree-based. These IoTaaSs could be owned and managed by different enterprises. Users can use cloud services and resources transparently. Each manufacturer or testing provider can implement its own IoTaaS which, in a way, can fit to the services provided. IoTaaS is component-based architecture which gives manufacturers and testing providers the possibility to (re)use components from other enterprises. Communications between IoTaaSs are managed by Cloud component in each IoTaaS. IoTaaS is a heterogeneous environment which can include real objects and things and their emulators at the same time. Several XML-based platform-independent scenarios have been explained in details. A design for Network Emulation Protocol (NEP) has been proposed and described in order to emulate various types of network. Gateways component helps to connect things and objects with testing environments. Gateways could be also integrated in order to be tested with various types of thing. Users can be a part of tests in order to give human feedback. Server component is the main component which bridges and manages all other components.
As a pilot implementation, we presented Cloud Environment for Mobile Application Testing (CEMAT). It is a distributed tree-based environment for testing Android applications. CEMAT is an heterogeneous Java-based environment which includes real/virtual mobile devices and emulators. Real mobile devices could be connected through a USB cable (direct connection) or Wi-Fi/3G-4G (indirect). CEMAT depends on XML-based configurations which means that Android commands and system paths could be changed easily in order to move CEMAT from a machine to another. CEMAT provides two types of scenarios: batch-based scenarios which depend on sending events and command using ADB connection and XML-based scenarios which depend on a launcher service on the mobile device in order to execute events and command. Traffic shaper is the module which controls traffic performance between CEMAT entities. Logging is the module which manages all logs generated by CEMAT entities and organizes them XML format. XML-Based GUI is the solution between the strictness of Java applications and the limitedness of Java Applets. It solves the problem of having different versions of the same service on CEMAT servers. It enables CEMAT server to send its services' GUIs to the users, so users don't need to install a new version every time a GUI is changed. Daemon is the core module of CEMAT which bridges between all other modules.

8.2/ PERSPECTIVES

CEMAT functions could be extended to support iOS and Windows Phones. The steps are as follows:

- Deploying CEMAT to work on Windows Server and Apple Mac Systems.
- Developing a *Scenario Launcher* for each of them.
- Integrating communication methods with testing devices for both of mobile OSs.
- Adding support for their emulators.

In the following, we list some functions which could be added-values:

- Porting traffic shaper to Android, iOS and Windows phones can help to simulate certain wireless communication behavior on mobile devices and on emulators.
- Adding Optical character recognition (OCR) support to read printed texts in the screenshots of the results. This can help automatizing results and performing statistics automatically.
- Adding Command Line Interface (CLI), which could enable users with slow Internet connections to login using SSH or Telnet.
- Adding peer-to-peer communication, so several users can build their own CEMAT environments and they can collaborate to exchange services in decentralized mode.

Depending on the results of CEMAT, IoTaaS could be enhanced by adding the following functions:

- Extending Network Emulation to cover GNSS emulation and Wi-Fi emulation on virtual machines.
- Providing several implementations of Cloud component using different methods: RESTful, CORBA, RMI, etc. That would unify all cloud functions and the interfaces to exchange services between IoTaaSs.
- Studying the possibility to use services from cloud computing providers such as: Amazon, Google, Microsoft, etc.

Studying other IoT sub-domains can help to evolve the architecture proposed. The most interesting sub-domains could be smart cities, Intelligent Transportation System.

9

COLDE

9.1/ Conclusion

We presented COLDE which is software-based extension to IEEE 802.11 protocol. COLDE benefits from the management frames to exchange small amount of public data between clients and APs. Devices can also request some services by broadcasting the requests in their coverage area. A COLDE-enabled Wi-Fi client can exchange data with several COLDE-enabled APs simultaneously, even if it isn't associated with any of them or if it was associated with a different AP. We presented a 3-tier design which consists of: Wi-Fi clients, Wi-Fi APs and COLDE-Proxy server. The Wi-Fi AP relays data between the clients and server. We explained our design for LightWeight Services and how COLDE can be used to exchange data and services in many domains. We presented three applications for COLDE. *Broadcasting information in variably dense environment* is the first application which is a simulation using NS-2. The experiment proved that by using COLDE, we can lower the time needed to broadcast a message in a given area to 4% of the time needed to broadcast same message without COLDE. Using a multi-tier broadcast model can extend the broadcast area which means that the message will reach a larger number of devices. Then, we developed COLDE-Proxy and implemented COLDE in access points and Wi-Fi clients (Raspberry Pi and Linux), so we could proceed with the second application which is *Indoor Positioning*. In this experiment, we proved the possibility to use COLDE by clients to find out their positions indoor without being connected to any network. Experiment showed that it is possible to have a connectionless indoor positioning system (regardless the method used) similar to GNSS systems which could help users to benefit from Location-Based Services (LBS) indoor and outdoor. The third application is *Emergency Evacuation* which simulation using NS-2. In this experiment, we proved that in order to rescue the largest number of people, each building should have a COLDE-based system to monitor people position in it, so it will be ready to generate an evacuation plan for each person according to its current location and the possible emergency exits. Such system can help people to evacuate the building, in addition that it can provide rescue teams with a list of persons who are trapped in the building.

9.2/ PERSPECTIVES

COLDE extension could be a great help as an enabler for IoT. In order to put this extension in use, following aspects should be covered:

- An implementation of COLDE for a mobile operating system can help to design more applications. We believe that Android is the best choice because it has already released a version for many objects other than mobile one, such as: watches, cars, televisions, etc.
- Integrating COLDE functions in an AP firmware could simplify testing the performance and collecting statistics in congested environments. There are several choices for open source access points, such as DD-WRT, OpenWRT, FreeWRT, etc.
- More experiments should be conducted to provide a security design for critical services such as: emergency requests. It is important to mention that any security design should save the simplicity of the extension.
- A deep study of the services which could be provided using COLDE can help to organize the categories.
- Manufacturers can play a significant role by integrating COLDE in their IoT objects.

Depending on the experiments and results, a suggestion for an official request for comment (RFC) could be requested in cooperation with interested enterprises.

BIBLIOGRAPHY

[6lowpan WG, 2012] 6lowpan WG (2012). **Ipv6 over low power wpan (6lowpan)**. Technical Report, IETF. http://datatracker.ietf.org/wg/6lowpan/documents/.

[802.15.4, 2011] 802.15.4 (2011). **Constrained restful environments (core)**. Technical Report, IEEE. https://standards.ieee.org/getieee802/download/802.15.4-2011.pdf.

[A. Duraisamy, 2013] A. Duraisamy, M. S. (2013). **Mesh based peer to peer live video streaming using ant algorithm**. *International Journal of Engineering and Advanced Technology (IJEAT)*, 2(3):375–380.

[Abu Oun et al., 2013] Abu Oun, O., Abdou, W., Bloch, C., et Spies, F. (2013). **Broadcasting alert messages inside the building: Challenges & opportunities**. In *IPIN 2013, 4-th Int. Conf. on Indoor Positioning and Indoor Navigation*, pages 292–295, MontbÃ©liard, France.

[Abu Oun et al., 2014a] Abu Oun, O., Abdou, W., Bloch, C., et Spies, F. (2014a). **Broadcasting information in variably dense environment using connectionless data exchange (colde)**. In Mellouk, A., Fowler, S., Hoceini, S., et Daachi, B., editors, *Wired/Wireless Internet Communications*, volume 8458 of *Lecture Notes in Computer Science*, pages 283–296. Springer International Publishing.

[Abu Oun et al., 2014b] Abu Oun, O., Bloch, C., et Spies, F. (2014b). **Indoor positioning using colde: An ieee 802.11 connectionless extension**. In *IPIN 2014, 5th Int. Conf. on Indoor Positioning and Indoor Navigation*, pages 1–10, Busan, Korea. IEEE.

[Abu Oun et al., 2015] Abu Oun, O., Bloch, C., et Spies, F. (2015). **Connectionless wi-fi for internet of things (iot)**. In *CIoT 2015, Cloudification of the Internet of Things 2015*, Paris, France.

[Adachi, 2006] Adachi, S. (2006). **The strategic choice between" standardization" and" differentiation" in R&D**. PhD thesis, Massachusetts Institute of Technology.

[Adya et al., 2004a] Adya, A., Bahl, P., Chandra, R., et Qiu, L. (2004a). **Architecture and techniques for diagnosing faults in ieee 802.11 infrastructure networks**. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, MobiCom '04, pages 30–44, New York, NY, USA. ACM.

[Adya et al., 2004b] Adya, A., Bahl, P., Padhye, J., Wolman, A., et Zhou, L. (2004b). **A multi-radio unification protocol for ieee 802.11 wireless networks**. In *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pages 344–354. IEEE.

[Aho et al., 2013] Aho, P., Suarez, M., Kanstrén, T., et Memon, A. M. (2013). **Industrial adoption of automatically extracted gui models for testing.** In *EESSMOD@ MoDELS*, pages 49–54.

[Alexey Kashevnik, 2012] Alexey Kashevnik, M. S. (2012). **Comparative analysis of indoor positioning systems based on communications supported by smartphones**. Technical Report, Saint-Petersburg Institute for Informatics and Automation of Russian Academy Science Saint-Petersburg. https://fruct.org/publications/fruct12/files/Kas.pdf.

[Alliance, 2015a] Alliance, O. H. (Accessed 2015a). **Android**. Technical Report, Open Handset Alliance. http://www.openhandsetalliance.com/android_overview.html.

[Alliance, 2015b] Alliance, O. H. (Accessed 2015b). **Industry leaders announce open platform for mobile devices**. Technical Report, Open Handset Alliance. http://www.openhandsetalliance.com/press_releases.html.

[Almeida et al., 2014] Almeida, F., Santos, J. D., et Monteiro, J. A. (2014). **e-commerce business models in the context of web3.0 paradigm**. *CoRR*, abs/1401.6102.

[Amalfitano et al., 2011] Amalfitano, D., Fasolino, A., et Tramontana, P. (2011). **A gui crawling-based technique for android mobile application testing**. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on*, pages 252–261.

[Android, 2015] Android (Accessed 2015). **Uiautomation**. Technical Report, Android. https://developer.android.com/reference/android/app/UiAutomation.html.

[android x86, 2015] android x86 (Accessed 2015). **Android-x86**. Technical Report. http://android-x86.org.

[Appium, 2015] Appium (Accessed 2015). **Introduction to appium**. Technical Report, Appium.

[Aringhieri et al., 2006] Aringhieri, R., Damiani, E., Di Vimercati, S. D. C., Paraboschi, S., et Samarati, P. (2006). **Fuzzy techniques for trust and reputation management in anonymous peer-to-peer systems: Special topic section on soft approaches to information retrieval and information access on the web**. *J. Am. Soc. Inf. Sci. Technol.*, 57(4):528–537.

[ARZENŠEK et al.,] ARZENŠEK, B., et HERIČKO, M. **Criteria for selecting mobile application testing tools**. In *Third Workshop on Software Quality Analysis, Monitoring, Improvement and Applications SQAMIA 2014*.

[Awad, 2005] Awad, M. (2005). **A comparison between agile and traditional software development methodologies**. *University of Western Australia*.

[Bahl et al., 2000] Bahl, P., et Padmanabhan, V. N. (2000). **RADAR: an in-building RF-based user location and tracking system**. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784 vol.2. IEEE.

[Bai et al., 2010] Bai, Y., Du, W., Ma, Z., Shen, C., Zhou, Y., et Chen, B. (2010). **Emergency communication system by heterogeneous wireless networking**. In *Wireless Communications, Networking and Information Security (WCNIS), 2010 IEEE International Conference on*, pages 488–492.

[Berners-Lee et al., 1995] Berners-Lee, T., et Connolly, D. (1995). **RFC 1866 – Hypertext Markup Language – 2.0**. http://www.faqs.org/rfcs/rfc1630.html.

[Berners-Lee et al., 2005] Berners-Lee, T., Fielding, R., et Masinter, L. (2005). **Rfc 3986, uniform resource identifier (uri): Generic syntax**.

[Berners-Lee et al., 1996] Berners-Lee, T., Fielding, R. T., et Nielsen, H. F. (1996). **RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0**. http://www.faqs.org/rfcs/rfc1945.html.

[Birrell et al., 1984] Birrell, A. D., et Nelson, B. J. (1984). **Implementing remote procedure calls**. *ACM Trans. Comput. Syst.*, 2(1):39–59.

[Booth et al., 2004] Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., et Orchard, D. (2004). **Web Services Architecture**. Number 11 in W3C Working Group Note.

[Brown et al., 2006] Brown, P. F., et Hamilton, R. M. B. A. (2006). **Reference model for service oriented architecture 1.0**.

[Bryson, 2013] Bryson, S. (2013). **Virtual reality: A definition history - A personal essay**. *CoRR*, abs/1312.4322.

[Calaba, 2015] Calaba (Accessed 2015). **Introduction to calaba**. Technical Report, Calaba.

[Casetti et al., 2014] Casetti, C., Chiasserini, C., Pelle, L. C., Valle, C. D., Duan, Y., et Giaccone, P. (2014). **Content-centric routing in wi-fi direct multi-group networks**. *CoRR*, abs/1412.0880.

[CATJS, 2015] CATJS (Accessed 2015). Technical Report, CATJS.

[CERN, 2015] CERN (Accessed 2015). **The birth of the web**. Technical Report, CERN. http://home.web.cern.ch/topics/birth-web.

[Chandra et al., 2004] Chandra, R., Bahl, P., et Bahl, P. (2004). **Multinet: Connecting to multiple ieee 802.11 networks using a single wireless card**. In *IEEE Infocom*. IEEE Communications Society.

[Chandra et al., 2007] Chandra, R., Padhye, J., Ravindranath, L., et Wolman, A. (2007). **Beacon-stuffing: Wi-fi without associations**. In *Proceedings of the Eighth IEEE Workshop on Mobile Computing Systems and Applications*, HOTMOBILE '07, pages 53–57, Washington, DC, USA. IEEE Computer Society.

[core WG, 2015] core WG (2015). **Constrained restful environments (core)**. Technical Report, IETF. http://datatracker.ietf.org/wg/core/documents/.

[Coskun et al., 2013] Coskun, V., Ozdenizci, B., et Ok, K. (2013). **A survey on near field communication (nfc) technology**. *Wirel. Pers. Commun.*, 71(3):2259–2294.

[Cromar, 2010] Cromar, S. (November 29, 2010). **Smartphones in the u.s.: Market analysis**. Technical Report, Business Strategy for Lawyers. https://www.ideals.illinois.edu/bitstream/handle/2142/18484/Cromar,%20Scott%20-%20U.S.%20Smartphone%20Market%20Report.pdf.

[Cypriani et al., 2010] Cypriani, M., Canalda, P., Lassabe, F., et Spies, F. (2010). **Wi-Fi-based indoor positioning: Basic techniques, hybrid algorithms and open software platform**. In Mautz, R., Kunz, M., et Ingensand, H., editors, *IPIN 2010, Int. Conf. on Indoor Positioning and Indoor Navigation, Session WLAN RSS (Signal Strength Based Methods)*, pages 116–125, Zurich, Switzerland.

[Cypriani et al., 2009] Cypriani, M., Lassabe, F., Canalda, P., et Spies, F. (2009). **Open wireless positionning system: a Wi-Fi-based indoor positionning system**. In *VTC-fall 2009, 70th IEEE Vehicular Technologie Conference*, pages 1–5, Anchorage, Alaska, United States. IEEE Computer Society Press.

[D. M'Raihi Verisign, 2011] D. M'Raihi Verisign, Inc., S. M. D. C. M. P. S. J. R. P. I. (2011). **TOTP: Time-Based One-Time Password Algorithm**. RFC 6238 (Informational).

[Dadeau et al., 2008] Dadeau, F., Héam, P.-C., et Levrey, J. (2008). **A combination of model-based testing and random testing approaches using automata**. Research Report RR2008-10, LIFC - Laboratoire d'Informatique de l'Université de Franche-Comté. 21 pages.

[Davis et al., 2002] Davis, D., et Parashar, M. P. (2002). **Latency performance of soap implementations**. In *Proceedings of the 2Nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '02, pages 407–, Washington, DC, USA. IEEE Computer Society.

[Deak et al., 2012] Deak, G., Curran, K., et Condell, J. (2012). **A survey of active and passive indoor localisation systems**. *Computer Communications*, 35(16):1939–1954.

[Deering et al., 1998] Deering, S., et Hinden, R. (1998). **RFC 2460 Internet Protocol, Version 6 (IPv6) Specification**. Internet Engineering Task Force.

[Del Barrio et al., 1998] Del Barrio, V. M., et Jiménez, A. F. (1998). **Study of the techniques for emulation programing**.

[Delgrande et al., 2011] Delgrande, J. P., et Faber, W., editors (2011). **Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings**, volume 6645 of *Lecture Notes in Computer Science*. Springer.

[Di Lucca et al., 2006] Di Lucca, G. A., et Fasolino, A. R. (2006). **Testing web-based applications: The state of the art and future trends**. *Inf. Softw. Technol.*, 48(12):1172–1186.

[Di Lucca et al., 2004] Di Lucca, G. A., Fasolino, A. R., et Tramontana, P. (2004). **Reverse engineering web applications: The ware approach**. *J. Softw. Maint. Evol.*, 16(1-2):71–101.

[Du et al., 2014] Du, J., Dean, D., Tan, Y., Gu, X., et Yu, T. (2014). **Scalable distributed service integrity attestation for software-as-a-service clouds**. *Parallel and Distributed Systems, IEEE Transactions on*, 25(3):730–739.

[Dwivedi et al., 2011] Dwivedi, Y. K., Williams, M. D., Mitra, A., Niranjan, S., et Weerakkody, V. (2011). **Understanding advances in web technologies: evolution from web 2.0 to web 3.0.** In Tuunainen, V. K., Rossi, M., et Nandhakumar, J., editors, *ECIS*.

[Eduardo Navarro, 2010] Eduardo Navarro, Benjamin Peuker, M. Q. D. C. C. D. J. J. (2010). **Wi-fi localization using rssi fingerprinting**. Technical Report, California Polytechnic State University. http://digitalcommons.calpoly.edu/cpesp/17.

[Ergen, 2014] Ergen, S. C. (2014). **Zigbee/ieee 802.15.4 summary**. Technical Report. http://www.sinemergen.com/zigbee.pdf.

[Ernst et al., 2007] Ernst, N., et Mylopoulos, J. (2007). **Tracing software evolution history with design goals**. In *Software Evolvability, 2007 Third International IEEE Workshop on*, pages 36–41.

[Ertin et al., 2006] Ertin, E., Arora, A., Ramnath, R., Nesterenko, M., Naik, V., Bapat, S., Kulathumani, V., Sridharan, M., Zhang, H., et Cao, H. (2006). **Kansei: a testbed for sensing at scale**. In *Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on*, pages 399–406.

[Evans, 2011] Evans, D. (2011). **How the next evolution of the internet is changing everything**. Technical Report, Cisco. https://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[Fantacci et al., 2010] Fantacci, R., Marabissi, D., et Tarchi, D. (2010). **A novel communication infrastructure for emergency management: the in. sy. eme. vision**. *Wireless Communications and Mobile Computing*, 10(12):1672–1681.

[Fielding, 2000] Fielding, R. T. (2000). **Architectural Styles and the Design of Network-based Software Architectures**. PhD thesis. AAI9980887.

[Fielding et al., 1999] Fielding, R. T., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P., et Berners-Lee, T. (1999). **Rfc 2616: Hypertext transfer protocol – http/1.1**.

[Flora et al., 2014] Flora, H. K., Wang, X., et Chande, S. V. (2014). **An investigation into mobile application development processes: Challenges and best practices**. *International Journal of Modern Education and Computer Science (IJMECS)*, 6(6):1–9.

[Fujitsu, 2010] Fujitsu (November 16, 2010). **Confidence in cloud grows, paving way for new levels of business efficiency**. Technical Report, Fujitsu. http://www.fujitsu.com/uk/news/pr/fs_20101116.html.

[Fujiwara et al., 2004] Fujiwara, T., Iida, N., et Watanabe, T. (2004). **A hybrid wireless network enhanced with multihopping for emergency communications**. In *Communications, 2004 IEEE International Conference on*, volume 7, pages 4177–4181. IEEE.

[Gandon, 2015] Gandon, F. (Accessed 2015). **Données liées et web sémantiquequand le lien fait sens**. Technical Report, INRIA. https://www.inria.fr/content/download/81770/1151570/version/4/file/Gandonrii14lille.pdf.

[Gao et al., 2014] Gao, J., Bai, X., Tsai, W.-T., et Uehara, T. (2014). **Mobile application testing: A tutorial**. *Computer*, 47(2):46–55.

[Gil-Castineira et al., 2011] Gil-Castineira, F., Costa-Montenegro, E., Gonzalez-Castano, F., Lopez-Bravo, C., Ojala, T., et Bose, R. (2011). **Experiences inside the ubiquitous oulu smart city**. *Computer*, 44(6):48–55.

[Giustiniano et al., 2009] Giustiniano, D., Goma, E., Lopez, A., et Rodriguez, P. (2009). **Wiswitcher: an efficient client for managing multiple aps**. In *Proceedings of the 2nd ACM SIGCOMM workshop on Programmable routers for extensible services of tomorrow*, pages 43–48. ACM.

[Gluhak et al., 2011] Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., et Razafindralambo, T. (2011). **A survey on facilities for experimental internet of things research**. *Communications Magazine, IEEE*, 49(11):58–67.

[Grønli et al., 2014] Grønli, T.-M., Hansen, J., Ghinea, G., et Younas, M. (2014). **Mobile application platform heterogeneity: Android vs windows phone vs ios vs firefox os**. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*, pages 635–641.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S., et Palaniswami, M. (2013). **Internet of things (iot): A vision, architectural elements, and future directions**. *Future Gener. Comput. Syst.*, 29(7):1645–1660.

[Gupta et al., 2007] Gupta, V., Beyah, R. A., et Corbett, C. L. (2007). **A characterization of wireless NIC active scanning algorithms**. In *IEEE Wireless Communications and Networking Conference, WCNC 2007, Hong Kong, China, 11-15 March, 2007*, pages 2385–2390.

[Gupta et al., 2012a] Gupta, V., et Rohil, M. K. (2012a). **Article: Information embedding in ieee 802.11 beacon frame**. *IJCA Proceedings on National Conference on Communication Technologies & its impact on Next Generation Computing 2012*, CTNGC(3):12–16. Full text available.

[Gupta et al., 2012b] Gupta, V., et Rohil, M. K. (2012b). **Information embedding in ieee 802.11 beacon frame**. In *National Conference on Communication Technologies & its impact on Next Generation Computing CTNGC*.

[Gupta et al., 2013] Gupta, V., et Rohil, M. K. (2013). **Bit-stuffing in 802. 11 beacon frame: Embedding non-standard custom information**. *International Journal of Computer Applications*, 63(2):6–12.

[Gutiérrez et al., 2013] Gutiérrez, V., Galache, J., Sánchez, L., Muñoz, L., Hernández-Muñoz, J., Fernandes, J., et Presser, M. (2013). **Smartsantander: Internet of things research and innovation through citizen participation**. In Galis, A., et Gavras, A., editors, *The Future Internet*, volume 7858 of *Lecture Notes in Computer Science*, pages 173–186. Springer Berlin Heidelberg.

[Hall et al., 2012] Hall, W., et Tiropanis, T. (2012). **Web evolution and web science.** *Computer Networks*, 56(18):3859–3865.

[Haller et al., 1998] Haller, N., Metz, C., Nesser, P. J., et Straw, M. (1998). **A one-time password system**. Internet RFC 2289.

[HART, 2015] HART (2015). **Hart overview**. Technical Report, Hart Communication Foundation. http://www.hartcomm.org/.

[Hartenstein et al., 2008] Hartenstein, H., et Laberteaux, K. P. (2008). **A tutorial survey on vehicular ad hoc networks**. *Communications Magazine, IEEE*, 46(6):164–171.

[Hendler, 2009] Hendler, J. (2009). **Web 3.0 emerging**. *Computer*, 42(1):111–113.

[Hostapd, 2015] Hostapd (2015). **Ieee 802.11 ap, ieee 802.1x/wpa/wpa2/eap/radius authenticator**. Technical Report, w1.fi. http://w1.fi/hostapd/.

[IDC, 2015] IDC (Accessed 2015). **Smartphone os market share, q4 2014**. Technical Report, IDC. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[IEEE, 1990] IEEE (1990). **Ieee standard glossary of software engineering terminology**. *IEEE Std 610.12-1990*, pages 1–84.

[IEEE, 2012] IEEE (2012). **Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications**. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793.

[Incki et al., 2012] Incki, K., Ari, I., et Sozer, H. (2012). **A survey of software testing in the cloud**. In *Proceedings of the 2012 IEEE Sixth International Conference on Software Security and Reliability Companion*, SERE-C '12, pages 18–23, Washington, DC, USA. IEEE Computer Society.

[Innocent, 2012] Innocent, A. A. T. (2012). **Cloud infrastructure service management - a review**. *CoRR*, abs/1206.6016.

[Intel, 2014] Intel (2014). **Understanding ieee 802.11 authentication and association**. Technical Report, Intel Wi-Fi. http://www.intel.com/support/wireless/wlan/sb/CS-025325.htm.

[IoT-LAB, 2015] IoT-LAB, F. (Accessed 2015). Technical Report, Future Internet Testbed (FIT) OneLab.

[IPSO, 2011] IPSO (2011). **Rpl: The ip routing protocol designed for low power and lossy networks**. Technical Report, Internet Protocol for Smart Objects (IPSO) Alliance. http://www.cs.berkeley.edu/~jwhui/6lowpan/IPSO-WP-7.pdf.

[Ishaq et al., 2013] Ishaq, I., Carels, D., Teklemariam, G., Hoebeke, J., Abeele, F. V. D., Poorter, E. D., Moerman, I., et Demeester, P. (2013). **Ietf standardization in the field of the internet of things (iot): A survey**. *J. Sensor and Actuator Networks*, pages 235–287.

[ISO, 2015] ISO (2015). **Open systems interconnection (osi)**. Technical Report, ISO. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_ics_browse.htm?ICS1=35&ICS2=100.

[ITU, 2012] ITU (2012). **Overview of the internet of things - y.2060**. Technical Report, ITU. http://www.itu.int/rec/T-REC-Y.2060-201206-I.

[John E. Bentley, 2005] John E. Bentley, Wachovia Bank, C. N. (2005). **Software testing fundamentals—concepts, roles, and terminology**. In *Proceedings of SAS Conference*, pages 10–13.

[Johnson, 1991] Johnson, B. C. (1991). **A distributed computing environment framework: An osf perspective**. Technical Report DEV-DCE-TP6-1, The Open Software Foundation.

[Juttner et al., 2005] Juttner, A., et Magi (2005). **Tree based broadcast in ad hoc networks**. *Mob. Netw. Appl.*, 10(5):753–762.

[Kandula et al., 2008] Kandula, S., Lin, K. C.-J., Badirkhanli, T., et Katabi, D. (2008). **Fatvap: Aggregating ap backhaul capacity to maximize throughput.** In *NSDI*, volume 8, pages 89–104.

[Khanduri et al., 2013] Khanduri, R., et Rattan, S. S. (2013). **Article: Performance comparison analysis between ieee 802.11a/b/g/n standards**. *International Journal of Computer Applications*, 78(1):13–20. Full text available.

[Kirk, 2015] Kirk, S. (2015). **Internet of things – monetization opportunities for b2bs**. Technical Report, CloudExpo. http://cloudcomputing.sys-con.com/node/3268835.

[Kopecký, 2006] Kopecký, J. (2006). **Wsdl rdf mapping: Developing ontologies from standardized xml languages**. In Roddick, J., Benjamins, V., Si-said Cherfi, S., Chiang, R., Claramunt, C., Elmasri, R., Grandi, F., Han, H., Hepp, M., Lytras, M., Mišić, V., Poels, G., Song, I.-Y., Trujillo, J., et Vangenot, C., editors, *Advances in Conceptual Modeling - Theory and Practice*, volume 4231 of *Lecture Notes in Computer Science*, pages 312–322. Springer Berlin Heidelberg.

[Krčo et al., 2012] Krčo, S., Fernandes, J., Jokić, S., Sanchez, L., Natti, M., Theodoridis, E., Vučković, D., Casanueva, J., Galache, J., Gutiérrez, V., et others (2012). **Smartsantander–a smart city experimental platform**. (268-272).

[Lassabe et al., 2005] Lassabe, F., Baala, O., Canalda, P., Chatonnay, P., et Spies, F. (2005). **A friis-based calibrated model for WiFi terminals positioning**. In *Proceedings of IEEE Int. Symp. on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2005)*, pages 382–387, Taormina, Italy.

[Lassabe et al., 2006] Lassabe, F., Canalda, P., Charlet, D., Chatonnay, P., et Spies, F. (2006). **Refining WiFi indoor positionning renders pertinent deploying location-based multimedia guide**. In *Procs of IEEE Int. Workshop on Pervasive Computing and Ad Hoc Communications (PCAC06), in conjunction with the IEEE 20th Int. Conf. on Advanced Information Networking and Applications (AINA06)*, volume 2, pages 126–130, Vienna, Austria.

[Li, 2011] Li, Y. (2011). **A survey on communication networks in emergency warning systems**. Technical Report WUCSE-2011-100, Washington University in St. Louis.

[Lim et al., 2000] Lim, H., et Kim, C. (2000). **Multicast tree construction and flooding in wireless ad hoc networks**. In *Proceedings of the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '00, pages 61–68, New York, NY, USA. ACM.

[Linthicum, 2009] Linthicum, D. S. (2009). **Cloud computing and SOA convergence in your enterprise: a step-by-step guide**. Pearson Education.

[Linux, 2015] Linux (2015). **Linux wireless, nl80211 documentation**. Technical Report, Linux Wireless. http://wireless.kernel.org/en/developers/Documentation/nl80211.

[Lopez-De-Ipina et al., 2007] Lopez-De-Ipina, D., Vazquez, J. I., et Jamardo, I. (2007). **Touch computing: Simplifying human to environment interaction through nfc technology", 1as jornadas científicas sobre rfid**.

[Ludovici et al., 2010] Ludovici, A., et Calveras, A. (2010). **Implementation and evaluation of multi-hop routing in 6lowpan**. *Proceedings of the 9th Conference of Telematic Engineering*.

[Ludwig et al., 2010] Ludwig, B., et Coetzee, S. (2010). **A comparison of platform as a service (paas) clouds with a detailed reference to security and geoprocessing services**. *Proceedings of the First International Workshop on Pervasive Web Mapping, Geoprocessing and Services (WebMGS 2010), Como, Italy, August 10Á12*.

[LWIG-WG, 2014] LWIG-WG (2014). **Terminology for constrained node networks**. Technical Report, IETF. https://tools.ietf.org/html/draft-ietf-lwig-terminology-07.

[Mattern et al., 2010] Mattern, F., et Floerkemeier, C. (2010). **From active data management to event-based systems and more**. chapter From the Internet of Computers to the Internet of Things, pages 242–259. Springer-Verlag, Berlin, Heidelberg.

[Mauthe et al., 2003] Mauthe, A., et Hutchison, D. (2003). **Peer-to-peer computing: Systems, concepts and characteristics**. *Praxis in der Informationsverarbeitung & Kommunikation (PIK), K. G. Sauer Verlag, Special Issue on Peer-to-Peer*, 26(03/03).

[Merriam-Webster, 2012] Merriam-Webster (2012). **Crowdsourcing - definition and more**. Technical Report, Merriam-Webster.com. http://www.merriam-webster.com/dictionary/crowdsourcing.

[Meyer et al., 2011] Meyer, S., Sperner, K., Magerkurth, C., et Pasquier, J. (2011). **Towards modeling real-world aware business processes**. In *Proceedings of the Second International Workshop on Web of Things*, WoT '11, pages 8:1–8:6, New York, NY, USA. ACM.

[MFT, 2015] MFT (Accessed 2015). **Introduction to mobile testing framework**. Technical Report, Mobile Testing Framework.

[Molnar et al., 2004] Molnar, D., et Wagner, D. (2004). **Privacy and security in library rfid: Issues, practices, and architectures**. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, pages 210–219, New York, NY, USA. ACM.

[Montenegro et al.,] Montenegro, G., Kushalnagar, N., Hui, J., et Culler, D. **RFC 4944 – Transmission of IPv6 Packets over IEEE 802.15.4 Networks**. IETF RFC.

[Morien, 2015] Morien, C. (Accessed 2015). **Connectivity 101: The internet of things**. Technical Report, The University of Texas at Austin. https://identity.utexas.edu/id-perspectives/connectivity-101-the-internet-of-things.

[Mosbah et al., 2013] Mosbah, M. M., Soliman, H., et El-Nasr, M. A. (2013). **Current services in cloud computing: A survey.** *CoRR*, abs/1311.3319.

[M'Raihi et al., 2005] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., et Ranen, O. (2005). **HOTP: An HMAC-Based One-Time Password Algorithm**. RFC 4226 (Informational).

[Muralidharan et al., 2008] Muralidharan, K., Dhanapal, K. B., et Chowdhury, A. R. (2008). **Bowl: Design and implementation of a (connectionless) broadcasting system over wireless lan**. In *Proceedings of the 2008 International Symposium on a World of*

Wireless, Mobile and Multimedia Networks, WOWMOM '08, pages 1–6, Washington, DC, USA. IEEE Computer Society.

[Muthukrishnan et al., 2005] Muthukrishnan, K., Lijding, M., et Havinga, P. (2005). **Towards smart surroundings: Enabling techniques and technologies for localization**. In Strang, T., et Linnhoff-Popien, C., editors, *Location- and Context-Awareness*, volume 3479 of *Lecture Notes in Computer Science*, pages 350–362. Springer Berlin Heidelberg.

[Nagesh et al., 2012] Nagesh, A., et Caicedo, C. (2012). **Cross-platform mobile application development**. *ITERA 2012 Conference, Indianapolis, IN*.

[Networks, 2002] Networks, I. (2002). **A Practical Approach to Identifying and Tracking Unauthorized 802.11 Cards and Access Points**. Technical Report, Interlink Networks. http://www.interlinknetworks.com/graphics/news/wireless_detection_and_-tracking.pdf.

[NFC_Forum, 2015] NFC_Forum (Accessed 2015). **What is nfc?** Technical Report, NFC_Forum. http://nfc-forum.org/what-is-nfc/.

[Nicholson et al., 2010] Nicholson, A. J., Wolchok, S., et Noble, B. D. (2010). **Juggler: Virtual networks for fun and profit**. *Mobile Computing, IEEE Transactions on*, 9(1):31–43.

[NIST, 2011] NIST (2011). **The nist definition of cloud computing**. Technical Report, National Institute of Standards and Technology. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[NIST, 2015] NIST (Accessed 2015). **Mobile ad hoc networks (manets)**. Technical Report, NIST. http://www.antd.nist.gov/wahn_mahn.shtml.

[Niu et al., 2013] Niu, J., Lu, B., Cheng, L., 0001, Y. G., et Shu, L. (2013). **Ziloc: Energy efficient wifi fingerprint-based localization with low-power radio.** In *WCNC*, pages 4558–4563. IEEE.

[Obraczka et al., 2001] Obraczka, K., Viswanath, K., et Tsudik, G. (2001). **Flooding for reliable multicast in multi-hop ad hoc networks**. *Wireless Networks*, 7(6):627–634.

[Olsson, 2014] Olsson, J. (2014). **6lowpan demystified**. Technical Report, Texas Instruments. http://www.ti.com/lit/wp/swry013/swry013.pdf.

[Oracle, 2015] Oracle (Accessed 2015). **An overview of rmi applications**. Technical Report, Oracle Java Documentation. https://docs.oracle.com/javase/tutorial/rmi/.

[O'Reilly et al., 2009] O'Reilly, T., et Battelle, J. (2009). **Web squared: Web 2.0 five years on**.

[panOULU, 2015] panOULU (Accessed 2015). Technical Report, Open Ubiquitous Oulu.

[Parata et al., 2007] Parata, C., Scarpa, V., et Convertino, G. (2007). **Flex-wifi: a mixed infrastructure and ad-hoc ieee 802.11 network for data traffic in a home environment**. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pages 1–6. IEEE.

[pcmag, 2015] pcmag (2015). **Definition of:application program**. Technical Report, PC Magazine. http://www.pcmag.com/encyclopedia/term/37919/application-program.

[Priyanka et al., 2012] Priyanka, Chana, I., et Rana, A. (2012). **Empirical evaluation of cloud-based testing techniques: A systematic review**. *SIGSOFT Softw. Eng. Notes*, 37(3):1–9.

[Qiu et al., 2004] Qiu, D., et Srikant, R. (2004). **Modeling and performance analysis of bittorrent-like peer-to-peer networks**. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 367–378, New York, NY, USA. ACM.

[Quadri et al., 2010] Quadri, S., et Farooq, S. U. (2010). **Software testing–goals, principles, and limitations**. *International Journal of Computer Applications*, 6(9):7–10.

[Ramani et al., 2014] Ramani, S. V., et Tank, Y. N. (2014). **Indoor navigation on google maps and indoor localization using RSS fingerprinting**. *CoRR*, abs/1405.5669.

[Research, 2015] Research, G. (2015). **Google turns on indoor mapping with google maps 6.0 for android**. Technical Report, GIGAOM Research. http://gigaom.com/2011/11/29/google-turns-on-indoor-mapping-with-google-maps-6-0-for-android/.

[RFC6650, 2012] RFC6650 (2012). **Rpl: Ipv6 routing protocol for low-power and lossy networks**. Technical Report, IETF. https://tools.ietf.org/html/rfc6550.

[RFC7252, 2014] RFC7252 (2014). **The constrained application protocol (coap)**. Technical Report, IETF. https://tools.ietf.org/html/rfc7252.

[Ricca et al., 2001] Ricca, F., et Tonella, P. (2001). **Analysis and testing of web applications**. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, pages 25–34, Washington, DC, USA. IEEE Computer Society.

[Robotium, 2015] Robotium (Accessed 2015). Technical Report, Robotium.

[Rodriguez, 2008] Rodriguez, A. (2008). **Restful web services: The basics**. Technical Report, IBM. http://www.ibm.com/developerworks/library/ws-restful/.

[roll WG, 2015] roll WG (2015). **Routing over low power and lossy networks (roll)**. Technical Report, IETF. http://datatracker.ietf.org/wg/roll/documents/.

[Ryser et al., 1999] Ryser, J., et Glinz, M. (1999). **A scenario-based approach to validating and testing software systems using statecharts**. In *Proc. 12th International Conference on Software and Systems Engineering and their Applications*.

[Schauer et al., 2013] Schauer, L., Dorfmeister, F., et Maier, M. (2013). **Potentials and limitations of wifi-positioning using time-of-flight**. In *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*, pages 1–9.

[Shaw, 2014] Shaw, E. (2014). **A survey of android app quality using third party markets**.

[Silva et al., 2014] Silva, E. M., Maló, P., et others (2014). **Iot testbed business model**. *Advances in Internet of Things*, 4(04):37.

[Simmel, 2012] Simmel, F. C. (2012). **Dna-based assembly lines and nanofactories**. *Current opinion in biotechnology*, 23(4):516–521.

[SmartSantander, 2015] SmartSantander (Accessed 2015). **Smartsantander experimental test facilities**. Technical Report FP7-257992, Smart Santander Consortium.

[Software, 2015] Software, S. (2015). **What is mobile testing?** Technical Report, SmartBear Software. http://smartbear.com/all-resources/articles/what-is-mobile-testing/.

[Spies et al., 2015] Spies, F., Chatonnay, P., et Abu Oun, O. (2015). **Adaptive indoor positioning algorithm using auto-calibration**. In *IPIN 2015, 6-th Int. Conf. on Indoor Positioning and Indoor Navigation*, Banff, Canada.

[Stallings, 2002] Stallings, W. (2002). **Ieee 802.11 wireless lan standard**. *Wireless Communications and Networks*, pages 458–477.

[Starov et al., 2013] Starov, O., Vilkomir, S., et Kharchenko, V. (2013). **Cloud testing for mobile software systems-concept and prototyping.** In *ICSOFT*, pages 124–131.

[StatCounter, 2015] StatCounter (Accessed 2015). **Top 8 mobile and tablet operating systems from june 2014 to may 2015**. Technical Report, StatCounter Global Stats. http://gs.statcounter.com/#mobile+tablet-os-ww-monthly-201406-201505-bar.

[Steinmetz et al., 2005] Steinmetz, R., et Wehrle, K. (2005). **2. what is this about?** In Steinmetz, R., et Wehrle, K., editors, *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, pages 9–16. Springer Berlin Heidelberg.

[Stoica et al., 2001] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., et Balakrishnan, H. (2001). **Chord: A scalable peer-to-peer lookup service for internet applications**. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA. ACM.

[Strembeck et al., 2004] Strembeck, M., et Zdun, U. (2004). **Scenario-based component testing using embedded metadata.** In *SOQUA/TECOS*, pages 31–45.

[Sun, 1988] Sun (1988). **RPC: Remote procedure call**. Proposal RFC1050, Internet Engineering Task Force.

[Sundmaeker et al., 2010] Sundmaeker, H., Guillemin, P., Friess, P., et Woelfflé, S., editors (2010). **Vision and Challenges for Realising the Internet of Things**. Publications Office of the European Union, Luxembourg.

[Szabolcs Karsai, 2014] Szabolcs Karsai, Z. T. (2014). **Comparison of wifi-based indoor positioning techniques**. In *1st International Conference and Exhibition on Future RFID Technologies*, pages 53–60. University of Applied Sciences and Bay Zoltan Nonprofit.

[Tanenbaum, 2002] Tanenbaum, A. (2002). **Computer Networks**. Prentice Hall Professional Technical Reference, 4th edition.

[Tian et al., 2013] Tian, Z., Tang, X., Zhou, M., et Tan, Z. (2013). **Fingerprint indoor positioning algorithm based on affinity propagation clustering**. *EURASIP J. Wireless Comm. and Networking*, 2013:272.

[Tseng et al., 2002] Tseng, Y.-C., Ni, S.-Y., Chen, Y.-S., et Sheu, J.-P. (2002). **The broadcast storm problem in a mobile ad hoc network**. *Wireless networks*, 8(2-3):153–167.

[tutorialspoint, 2015] tutorialspoint (Accessed 2015). **Wap - introduction**. Technical Report, tutorialspoint.com. http://www.tutorialspoint.com/wap/wap_quick_guide.htm.

[Violino, 2005] Violino, B. (2005). **The basics of rfid technology**. Technical Report, RFID Journal. http://www.rfidjournal.com/articles/view?1337.

[Vouk, 2008] Vouk, M. A. (2008). **Cloud computing - issues, research and implementations**. *CIT*, 16(4):235–246.

[WANG et al., 2010] WANG, Q.-w., Shi, H.-s., et Qi, Q. (2010). **A dynamic probabilistic broadcasting scheme based on cross-layer design for manets**. *International Journal of Modern Education and Computer Science (IJMECS)*, 2(1):40.

[webfoundation, 2015] webfoundation (Accessed 2015). **History of the web**. Technical Report, webfoundation. http://webfoundation.org/about/vision/history-of-the-web/.

[Weis, 2007] Weis, S. A. (2007). **Rfid (radio frequency identification): Principles and applications**. *System*, 2:3Principles.

[Weiser, 1991] Weiser, M. (1991). **The computer for the 21st century**. *Scientific american*, 265(3):94–104.

[Weiser, 2015] Weiser, M. (Accessed 2015). **Ubiquitous computing**. Technical Report, ubiq.com. http://www.ubiq.com/hypertext/weiser/UbiHome.html.

[Werner-Allen et al., 2005] Werner-Allen, G., Swieskowski, P., et Welsh, M. (2005). **Motelab: a wireless sensor network testbed**. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 483–488.

[Westcott et al., 2011] Westcott, D. A., Coleman, D. D., Miller, B., et Mackenzie, P. (2011). **CWAP Certified Wireless Analysis Professional Official Study Guide: Exam PW0-270**. John Wiley & Sons.

[White, 1976] White, J. E. (1976). **A high-level framework for network-based resource sharing**. In *Proceedings of the June 7-10, 1976, National Computer Conference and Exposition*, AFIPS '76, pages 561–570, New York, NY, USA. ACM.

[Williams et al., 2002] Williams, B., et Camp, T. (2002). **Comparison of broadcasting techniques for mobile ad hoc networks**. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking &Amp; Computing*, MobiHoc '02, pages 194–205, New York, NY, USA. ACM.

[Williams, 2004] Williams, L. (2004). **Testing overview and black-box testing techniques**. *Alamat situs: http:// www.agile.csc.ncsu.edu/ SEMaterials/ BlackBox.pdf*.

[Williamson, 2012] Williamson, L. (April 2012). **A mobile application development primer**. Technical Report, IBM Corporation. http://www-304.ibm.com/industries/publicsector/fileserve?contentid=250729.

[Wireless, 2015] Wireless, S. (2015). **Skyhook location sdk**. Technical Report, Skyhook Wireless. http://www.skyhookwireless.com/.

[Wisc, 2015] Wisc, U. o. W.-M. (Accessed 2015). **Help shape the vision of the uw iot lab**. Technical Report, WISC.

[Yamaguchi et al., 2012] Yamaguchi, H., Higuchi, T., et Higashino, T. (2012). **Collaborative indoor positioning of mobile nodes**. In *Proceedings of The Sixth International Conference on Mobile Computing and Ubiquitous Networking (ICMU 2012)*.

[Younan et al., 2015] Younan, M., Khattab, S., et BAHGAT, R. (2015). **An integrated testbed environment for the web of things**. In *ICNS 2015, The Eleventh International Conference on Networking and Services*, Rome, Italy.

[Yun et al., 2012] Yun, M., Kim, D., seok Lee, H., et Lee, J. (2012). **Silent broadcast: Experience of connectionless messaging using wi-fi p2p**. In *Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on*, volume 2, pages 239–242.

[Zahid Farid, 2013] Zahid Farid, Rosdiadee Nordin, M. I. (2013). **Recent advances in wireless indoor localization techniques and system**. *Journal of Computer Networks and Communications*, 1(1).

[Zhang Da, 2010] Zhang Da, Feng Xia, Z. Y. L. Y. W. Z. (2010). **Localization technologies for indoor human tracking**. *CoRR*, abs/1003.1833.

[ZigBee, 2015] ZigBee (2015). **The zigbee alliance**. Technical Report, ZigBee. http://www.zigbee.org/.

List of Figures

List of Tables

V

ANNEXES

A

NETWORK-BASED APPLICATIONS TECHNIQUES AND EVALUATION

According to *Internet World Stats*, there were more than 3 billions worldwide Internet users in the world by June 30, 2014. That means more than 42.3% of population had access to Internet. Many techniques have been developed in order to implement network-based applications and to give network services. In this chapter, we present a brief summary of these techniques and how they evolved by time, we present also the transition to the era of the web services.

A.1/ INTRODUCTION

Distributed Computing is the process of aggregating the power of several computing entities, which are logically distributed and may even be geologically distributed, to collaboratively run a single computational task in a transparent and coherent way, so that they appear as a single, centralized system [Ernst et al., 2007].

Telnet (Telecommunications Network) can be considered the first step in distributed computing protocols, in which it allows users to run programs on another system. The syntax and semantics of Telnet vary from one system to another and are unregulated by the protocol, the user and server processes simply shuttle characters between the human user and the target system [White, 1976].

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. Procedure calls are a well-known and well- understood mechanism for transfer of control and data within a program running on a single computer. Therefore, it is proposed that this same mechanism be extended to provide transfer of control and data across a communication network. When a remote procedure is invoked, the calling environment is suspended, the parameters are passed across the network to the environment where the procedure is to execute (which we will refer to as the callee), and the desired procedure is executed there. When the procedure finishes and produces its results, the results are passed backed to the calling environment, where execution resumes as if returning from a simple single-machine call [Birrell et al., 1984]. Several software companies began to develop RPC implementations, such as Open Network Connectivity (ONC) [Sun, 1988] by Sun Microsystems and Distributed Computing Environment (DCE) [Johnson, 1991] by Open Software Foundation (OSF), the latter is a consortium that in-

cluded Apollo Computer (later part of Hewlett-Packard), IBM, Digital Equipment Corporation, and others. Later, Microsoft released Distributed Component Object Model (DCOM) as an extension of the DCE-RPC standard.
Common Object Request Broker Architecture (CORBA) is a standard defined by the Object Management Group (OMG) designed to facilitate the communication of systems that are deployed on diverse platforms. CORBA is an object-based distributed computing specification. Rather than remote procedures, self-contained components are transferred by middleware mediators [Ernst et al., 2007].
Remote Method Invocation (RMI) is an RPC-like remote object protocol. The real advantages of RMI are the excellent integration in Java, simplicity and efficiency.
Enterprise JavaBeans (EJB) was proposed by Sun Microsystems as a solution for developing complex distributed systems, in which architectures, like CORBA or RMI, weren't effective in handling complexity. EJB release was the first development of application servers, where a central server, contains the application and business logic, delivers applications to client machines.
Starting from the first appearance of the World Wide Web (Web), distributed computing technologies leveraged this new opportunity. In section A.2, we present a summary of the Web protocols.
Tim Berners-Lee's (inventor of the Web) vision for the Web was a two-way (or even multi-person to multi-person) medium, more close to a peer-to-peer system than a client-server one. In section A.3, we discuss the structure of Peer-to-Peer applications. Section A.4 presents a review of mobile applications. Section A.5 discusses Cloud-Based Applications.

A.2/ World Wide Web (Web)

Web, as an example for Internet application, had been invented by Tim Berners-Lee in 1989 at CERN. The Web was originally conceived and developed to meet the demand for automatic information-sharing between scientists in universities and institutes around the world. Three fundamental technologies [CERN, 2015] [webfoundation, 2015] had been specified (Figure A.1), in which they remain the foundation of today's Web: HTML , URI and HTTP.
These three technologies are presented in the following subsection A.2.1, A.2.2 and A.2.3.

A.2.1/ HyperText Markup Language (HTML)

The publishing format for the Web, including the ability to format documents and link them to other documents and resources. HTML is a simple markup language used to create hypertext documents that are platform independent. HTML documents are Standard Generalized Markup Language (SGML) documents with generic semantics that are appropriate for representing information from a wide range of domains. HTML markup can represent hypertext news, mail, documentation, and hypermedia; menus of options; database query results; simple structured documents with in-lined graphics; and hypertext views of existing bodies of information [Berners-Lee et al., 1995].
HTML5 adds many new features. These include the new tags to support video, audio and canvas elements, as well as the integration of Scalable Vector Graphics (SVG) content.

HTLML5 presented MathML for mathematical formulas, in addition to many tags to enrich the semantic content of documents.

A.2.2/ Uniform Resource Identifier (URI)

It is a compact sequence of characters that identifies an abstract or physical resource. Uniform Resource Locator (URL) is a subset of the URI that specifies where an identified resource is available and the mechanism for retrieving it [Berners-Lee et al., 2005].

A.2.3/ Hypertext Transfer Protocol (HTTP)

HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems. It allows retrieval of linked resources from across the Web. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and header [Berners-Lee et al., 1996].
The set of common methods [Fielding et al., 1999] for HTTP/1.1 is defined below. It is important to understand these methods in general and separate way from its use with hypertext:

- **OPTIONS:** It allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.
- **GET:** It retrieves whatever information (in the form of an entity) is identified by the Request-URI.
- **HEAD:** It can be used for obtaining meta-information about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.
- **POST:** It is used to request that the origin server accept (from the client) the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line.
- **PUT:** It requests that the enclosed entity be stored (to be created if the resource doesn't exist, otherwise to replace the original one) under the supplied Request-URI. The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity, while the URI in a PUT request identifies the entity enclosed with the request.
- **DELETE:** It requests that the origin server delete the resource identified by the Request-URI.
- **TRACE:** It is used to invoke a remote, application-layer loopback of the request message. It allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information.
- **CONNECT:** It can be used with a proxy that can dynamically switch to being a tunnel.

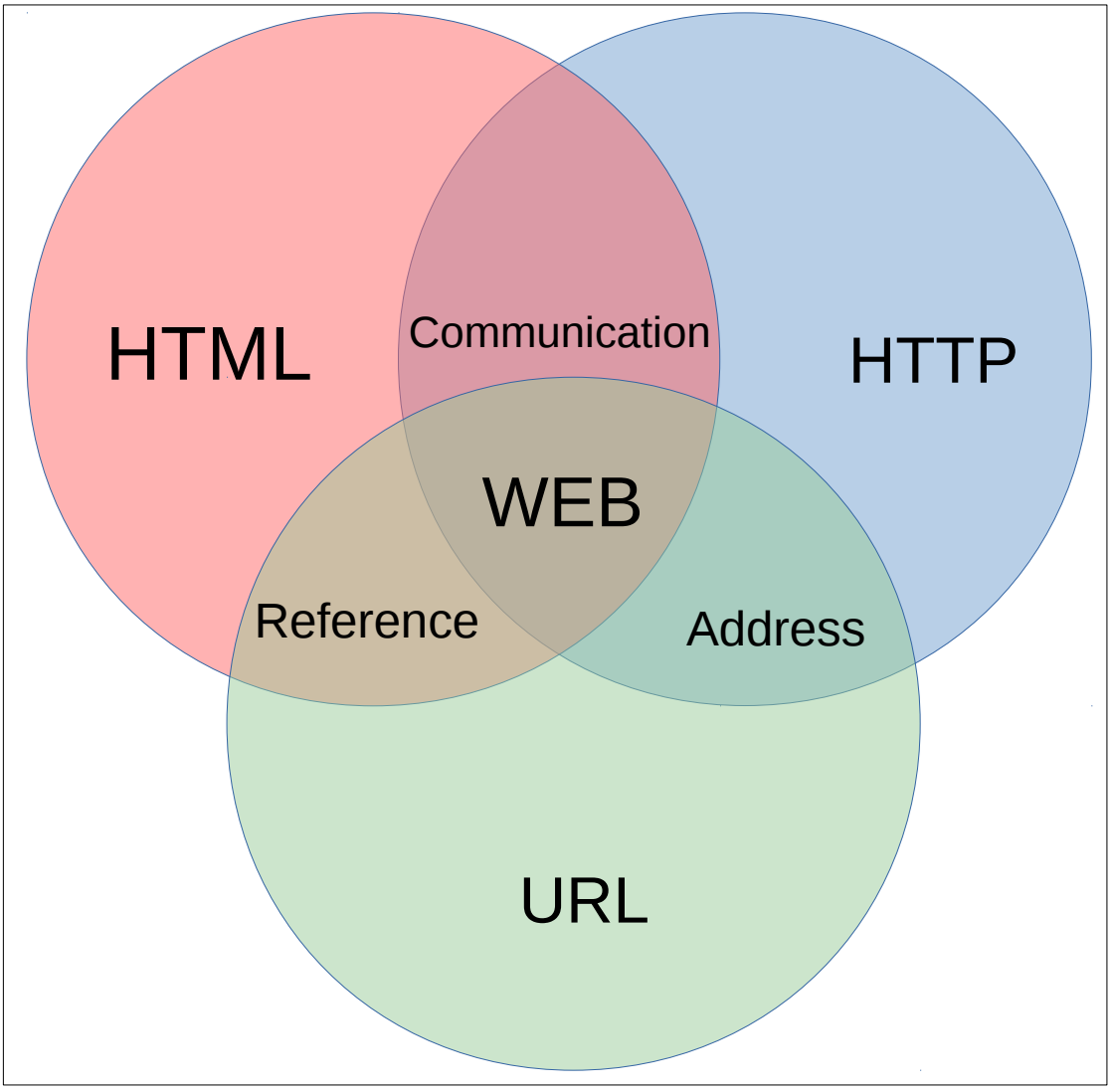

Figure A.1: Web - Fundamental Technologies [Gandon, 2015]

A.2.4/ WEB STAGES

The Web has gone through several distinct evolutionary stages. These stages have been characterized mainly on how the data is gathered and processed [Hall et al., 2012]:

1. **The Web of documents (Web 1.0):** At that stage, the Web appeared to be a technological artifact could be accessed using a personal computer, and which was initially a source of information and news and, later, a place to make purchases. That period is often referred to as the read-only Web. From the name, we can say that the element was the documents, and mostly, they weren't provided by users. Web 1.0 lasted from 1989 to 2005.

2. **The Web of people (Web 2.0):** It is called also the "read-write Web", in which people started to be a more significant part by contributing to the web contents. One of the fundamental ideas underlying Web 2.0, namely that *successful network applications are systems for harnessing collective intelligence*. Such appli-

cations depend on managing, understanding, and responding to massive amounts of user-generated data in real time. [O'Reilly et al., 2009]. This idea has led to the concept of *Crowdsourcing*, which is the process of obtaining needed services, ideas, or content by soliciting contributions from a large group of people, and especially from an online community, rather than from traditional employees or suppliers [Merriam-Webster, 2012]. Since then, Crowdsourcing has been utilized to build sites such as *Wikipedia*

3. **The Web of data and social networks (Web 3.0):** The main idea behind the Web 3.0, also known by its pseudonym as Semantic Web, was the creation of Web content by not using natural language but a form of script that could be understood and gauged by software agents in order to allow them to find, share or integrate information much more easily and efficiently, meeting the first stepping stone towards intelligent applications [Dwivedi et al., 2011]. While the specific nature of Web 3.0 technologies are difficult to define precisely, the outline of emerging applications has become clear over the past year. Web 3.0 is about linking and connecting web of data but transforming it in knowledge. With the web 3.0, search engines produce different results by the user: one user, one question, one result according to his/her profile. The Web 3.0 organizes and assembles the pages found in a search engine, by themes, topics. The idea is to read, analyze and identify the directions of semantical words so as to relate the information to each other. Additionally it is able to deal with the interests previously defined by people themselves. In addition, the browsers themselves over time analyze the main interests of the person and use it to improve the quality of searches. The more we research, the more the browser learns [Almeida et al., 2014] [Hendler, 2009]. Some references describe this stage as read-write-execute, while others consider it as an extension for Web 2.0 because we use the same technologies that had been developed in Web 2.0.

4. **Web 4.0 and Web of Things:** There is still no exact definition of it. Web 4.0 is also known as symbiotic web, it is expected to give a standardization for interaction between humans and machines.

A.2.5/ WEB SERVICES

A Web service is a software system designed to support inter-operable machine-to-machine interaction over a network [Booth et al., 2004]. In other words, it provides a method of communication between two electronic devices over a network. With web services, the web protocol suite can be used to achieve the same goal as other technologies such as: RPC, RMI, etc. A Web service enables this communication by using a combination of open protocols and standards as follows:

- **Data Structure - *Extensible Markup Language (XML):*** It is a markup language, similar to HTTP, that defines a set of rules for structuring of electronic documents and describing data in a text-based format which is both human-readable and machine-readable.

- **Message Transfer - *Web Service Definition Language (WSDL):*** It enables language-independent description of Web services. In particular, it can describe the

structure of the messages the service accepts and produces, simple message exchanges (called operations) and all necessary networking details [Kopecký, 2006].

- **Service Definition - *Simple Object Access Protocol (SOAP):*** It is a lightweight XML-based protocol for exchange of information in a decentralized, distributed environment. SOAP doesn't specify a transport mechanism, most SOAP implementations use HTTP [Davis et al., 2002]. SOAP is not bound to a particular programming language or software platform.

Web services have two advantages over the other distributed-based technologies. Firstly, HTTP-based SOAP implementations utilize the standard ports so they can transverse firewalls seamlessly, because firewalls will treat SOAP messages similarly as other HTTP messages. Secondly, SOAP enables interoperability with services developed on other platforms (such as Microsoft .NET, Java, etc.).

A.3/ PEER-TO-PEER APPLICATIONS

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality [Stoica et al., 2001].
In the year 2000, a music-sharing application called Napster was the beginning of peer-to-peer networks, as we know them today. Millions of users connecting to the Internet have started using their ever more powerful home computers for more than just browsing the Web and trading email. Over 50% of traffic is due to Peer-to-Peer applications, sometimes even more than 75% [A. Duraisamy, 2013].
there are three common characteristics defining a P2P application [Aringhieri et al., 2006]:

- The ability to discover other peers without the need of a centralized index.
- The ability to query other peers.
- The ability to share content with other peers.

The following characteristics are general features that can be used to identify peer-to-peer systems [Mauthe et al., 2003]:

- **Decentralization** is One of the major concepts of peer-to-peer computing. This includes distributed storage, processing, information sharing, etc. Even control information can be held in a distributed manner rather than centrally. The advantage of decentralization is an increased extensibility, higher system availability and improved resilience, transferral of ownership and control (of data, information and computational resources) to the application users.
- **Extensibility:** Peer-to-peer applications can potentially grow very large since resources can be added almost indefinitely. Though, another issue here is that especially in heterogeneous systems no performance guarantees can be given since it is not known which instance is serving a request.

- **Self-organizing:** different system components work together without any central management instance assigning roles and tasks.
- **Fault-tolerant:** There is no central point of failure, so peers can easily compensate the loss of a peer or even a number of peers.

The concept of Napster has inspired new structures and philosophies in many areas of human interaction. The following list summarizes the most important categorizes with some example:

- **File-sharing networks:** Many peer-to-peer file sharing networks, such as Kazza, Gnuttella, eDonkey/overnet, BitTorrent network popularized peer-to-peer technologies [Qiu et al., 2004].
- **Multimedia:** such as P2PTV and PDTP protocols.
- **Commercial applications:** Tradepal, M-commerce, Bitcoin, Midpoint and CurrencyFair.

Due to its main design principle of being completely decentralized and self-organizing; as opposed to the Internet's traditional Client-Server paradigm; the Peer-to-Peer concept emerges as a major design pattern for future applications, system components, and infrastructural services, particularly with regard to scalability and resilience. The growth in the usage of these applications is enormous and even more rapid than that of the World Wide Web. Nowadays, according to several Internet service providers, such applications represent more than 50% of Internet [Steinmetz et al., 2005].

A.4/ MOBILE APPLICATIONS

In the 1990s, the mobile phone systems emerged. Mobile was just a phone that can make and receive telephone calls over a radio link while moving around a wide geographic area. Mobile applications were simple applications, such as Short Messaging System (SMS) applications, small arcade games, ring tone editors, calculators, calendars, and so forth. Mobile systems and applications were produced in-house. Mobile Internet services had been introduced around the end of the 1990s. At the beginning, these services were provided mainly with characters and without graphics or pictures on the existing black-and-white small displays of mobile handsets [Adachi, 2006]. But there were two limitations:

- **Mobile phone:** It includes CPU performance, amount of memory, electricity (for battery-powered units), and input-output interface.
- **Mobile Network:** It includes transmission bandwidth, delay, and stability.

With the emergence of mobile Internet service, Wireless Application Protocol (WAP) was presented as a major technological evolution. WAP is a standardized technology for cross-platform, distributed computing very similar to the Internet's combination of HTML

and HTTP, except that it is optimized for: low-display capability, low-memory and low-bandwidth devices, such as personal digital assistants (PDAs), wireless phones, and pagers [tutorialspoint, 2015].
A smartphone (or smart phone) is a mobile phone which combines the features of a cell phone with different features which vary from device to another, such as PDA, media player (Global Positioning System) GPS navigation unit, camera, Bluetooth, Wi-Fi, video camera, speech recognition, voice recorder, near field communication (NFC), infrared blaster, etc. In 2007, Apple Inc. introduced the iPhone, one of the first smartphones to use a multi-touch interface. In 2008, Android released its first smartphone, followed by Windows Phone from Microsoft Inc in 2010. Other smartphone operating systems have been developed over the years, such as Firefox OS, Sailfish OS, Tizen, Ubuntu Touch, BlackBerry 10, etc.
Mobile platform are different from desktop systems in many ways. The main differences are:

- **Hardware Limitation:** Mobile platforms are constrained in terms of available resources such as: CPU, memory capacity and bandwidth, power consumption and physical size.
- **Display:** Mobile devices provide smaller screen with lower resolution pixel density compared to computer displays.
- **User input technology:** Mobile devices have pioneered the use of non-keyboard "gestures"; such as Touch, swipe, and pinch; as an effective and popular method of user input [Williamson, 2012]. In addition to tactile user input, mobile devices are a natural target for voice-based user input and camera-based input. This difference has a significant impact on the usability and user interaction design.

Therefore, traditional software engineering approaches and methods used in the development of desktop applications may not be directly applicable to a mobile environment. There is still lack of research initiatives and insufficient understanding of real issues and challenges faced during the development of mobile applications [Flora et al., 2014]. Lack of standard to develop mobile applications complicates applications evaluation mission.
Mobile applications can be classified into three categories [Nagesh et al., 2012]:

- **Native:** Native applications are those which are developed using mobile software development kits (SDK), tools and languages that are native to a particular mobile OS.
- **Mobile-web:** Mobile-web applications are those which use an instance of a mobile web browser to run the application. The user interface (UI) is developed in HTML5 and JavaScript and the logic is defined by JavaScript.
- **Hybrid:** They are developed using open source libraries but also have access to some of the native capabilities of a device such as Camera, GPS, Accelerometer, File System, etc.

The next list gives a short introduction to each of the main mobile platforms:

- **Android:** Android is built on the open Linux Kernel. It utilizes a custom virtual machine that was designed to optimize memory and hardware resources in a mobile

environment. Android is open source; it can be liberally extended to incorporate new cutting edge technologies as they emerge [Alliance, 2015a]. Each android application is executed within a Dalvik Virtual Machine (DVM) running under a unique UNIX User Identifier (UID).

- **iOS:** It is the operating system for several Apple devices, one of the most important of which is the iPhone. Applications for iOS are written in Objective-C using the Cocoa Touch library. Objective-C is an extension to the C language, while Cocoa Touch is a collection of classes [Grønli et al., 2014].
- **Windows Phone:** Applications for Windows Phone 7 are written in .NET managed code. Managed code is code written in languages that are available for use with the Microsoft .NET Framework, for example C#. One of the benefits is that many of the error-prone and often complex tasks, such as type safety checking, memory management and destruction of unneeded objects, are taken care of [Grønli et al., 2014].

Android dominates the market with a 76.6% share in 4th Quarter 2014, followed by iOS with 19.7% and Windows Phone with a 2.8%, 0.9% of the market share is divided between the rest of the mobile OS [IDC, 2015].

A.5/ CLOUD-BASED APPLICATIONS

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [NIST, 2011]. In other words, Cloud-based applications are distributed applications but on centralized facilities operated by third-party compute and storage utilities.
Cloud computing has been enabled by the developments in virtualization, distributed computing, utility computing, web and software services technologies [Vouk, 2008]. It is especially based on two key concepts. The first one is *Service-Oriented Architecture (SOA)*, which is the delivery of an integrated and orchestrated suite of functions to an end-user. The functions can be both loosely or tightly coupled. SOA enables end-users to easily search, use and release services on-demand and at a desired quality level. Workflows allow integration of services to deliver a business-valued application. The second key concept is *virtualization*. Virtualization allows abstraction and isolation of lower level functionalities and hardware, which enables portability of higher level functions and sharing and/or aggregation of the physical resources [Incki et al., 2012].

There are four deployment models [NIST, 2011]:

- **Private cloud:** The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.
- **Community cloud:** The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns

(e.g., mission, security requirements, policy , and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

- **Public cloud:** The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.
- **Hybrid cloud:** The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

There are five acknowledged types of service offerings:

- **Software-as-a-Service (SaaS):** SaaS cloud systems enable application service providers to deliver their applications via massive cloud computing infrastructures [Du et al., 2014].
- **Platform-as-a-Service (PaaS):** It is also known as Cloud platform service, provides a computing platform or solution stack on which software can be developed for later deployment in a cloud [Ludwig et al., 2010].
- **Infrastructure-as-a-Service (IaaS):** IaaS refers to computing resources as a service. IaaS provides virtual machines, virtual storage, networking technology, data center space, and other hardware assets as resources that clients can provision. The IaaS service provider manages the entire infrastructure, while the client is responsible for the deployment aspects [Innocent, 2012].
- **Testing as a Service (TaaS):** gives the ability to test local or cloud delivered systems by using remotely hosted testing software, hardware, services. It has the ability to test cloud applications, web sites and systems of the internal enterprise which do not need a hardware or software. An amount of leading companies are providing testing as a service such as SOASTA, and PushToTest, and so on [Mosbah et al., 2013].
- **Mobile Testing as a Service (mTaaS):** One of the most important phases in the Software Development Model is Testing. Testing is the phase where the developer can run the application in different scenarios, to find out the bugs, to determine the applicable scenarios and to make sure of the application behavior and performance by applying some real circumstances.
  Unlike the familiar realm of desktop-based and web-based software, Mobile applications developers and testers are immediately confronted by multiple operating systems (and different versions of each OS, especially with Android), multiple devices (different makes and models of phones, tablets, phablets), multiple carriers (including international ones), multiple speeds of data transference (3G, LTE, Wi-Fi), multiple screen sizes (and resolutions and aspect ratios), multiple input controls (including BlackBerry's eternal physical keypads), and multiple technologies

— GPS, accelerometers — that web and desktop applications almost never use [Software, 2015].

B

INTERNET OF THINGS (IOT)

B.1/ COMMUNICATION TECHNOLOGIES

Radio frequency identification (RFID) is a method of uniquely identifying an object using a tag that carries a unique code or identification.
NFC as one of the enablers for ubiquitous computing is a combination of contactless identification and interconnection technologies [Lopez-De-Ipina et al., 2007]

B.1.1/ RADIO FREQUENCY IDENTIFICATION(RFID)

An RFID tag is a small, low-cost device that can hold a limited amount of data and report that data when queried over radio by a reader [Molnar et al., 2004].
RFID systems are composed of three core components (Figure B.1)

- **Tag or transponder** (often used interchangeably) uses a silicon microchip to store a unique serial number and usually some additional information. A tag is composed of: an Integrated circuit (IC) and an antenna. There are 3 categories of tags [Weis, 2007]:
  - **Passive or battery-less tags** have no on-board power source and must passively harvest all energy from an RF (Radio Frequency) signal. They have the shortest read range of all three powering types (max range is 10 meters).
  - **Active or battery-powered tags** require a power source for additional range (20 to 100 meters), processing capabilities, and autonomy. There are two types of active tags: transponders and beacons. Active transponders are woken up when they receive a signal from a reader. These are used in toll payment collection, checkpoint control and other systems. Beacons are used in most Real-Time Locating Systems (RTLS), where the precise location of an asset needs to be tracked. A beacon emits a signal with its unique identifier at pre-set intervals (it could be every three seconds or once a day, depending on how important it is to know the location of an asset at a particular moment in time) [Violino, 2005].
- **Reader** uses RF signals to communicate with Tags to obtain identifying information.
- **Database** associates records with tag identifying data.

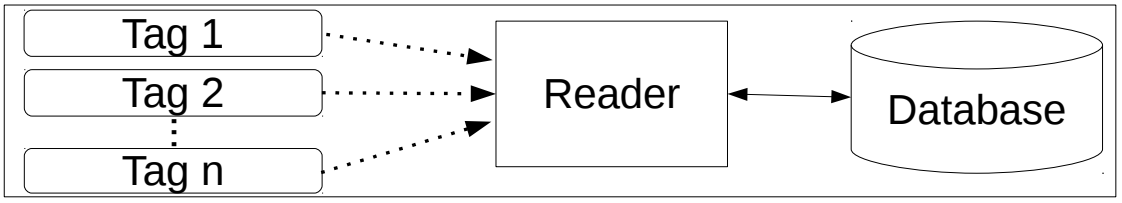

Figure B.1: RFID System Interaction

B.1.2/ IEEE 802.15.4

IEEE 802.15.4 is developed for applications with relaxed throughput requirements which cannot handle the power consumption of heavy protocol stacks. It provides network flexibility, low cost, very low power consumption, and low data rate. IEEE 802.15.4 is a low-rate wireless personal area network (LR-WPAN) [Ergen, 2014]. The standard covers (Figure B.2):

- **Physical layer (PHY)**: The features of the PHY are activation and deactivation of the radio transceiver, ED, LQI, channel selection,clear channel assessment (CCA), and transmitting as well as receiving packets across the physical medium.
- **Medium Access Control (MAC)**: The features of the MAC sublayer are beacon management, channel access, GTS management, frame validation, acknowledged frame delivery, association, and disassociation. In addition, the MAC sublayer provides hooks for implementing application-appropriate security mechanisms.

Some of the main characteristics of LoWPANs are as follows:

1. **Small packet size:** Given that the maximum physical layer packet is 127 bytes, the resulting maximum frame size at the media access control layer is 102 octets. Link-layer security imposes further overhead, which in the maximum case (21 octets of overhead in the AES-CCM-128 case, versus 9 and 13 for AES-CCM-32 and AES-CCM-64, respectively), leaves 81 octets for data packets.
2. **Low bandwidth:** Data rates of 250 kbps, 40 kbps, and 20 kbps for each of the currently defined physical layers (2.4 GHz, 915 MHz, and 868 MHz, respectively).
3. **MAC Addresses:** Support for both 16-bit short or IEEE 64-bit extended media access control addresses.

Two different device types can participate in an IEEE 802.15.4 network:

- **Full-Function Device (FFD)** is a device that is capable of serving as a personal area network (PAN) coordinator or a coordinator.
- **Reduced-Function Device (RFD)** is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor; it does not have the need to send large amounts of data and only associates with a single FFD at a time.

IEEE 802.15.4 LR-WPAN operates in either of two topologies (Figure B.3):

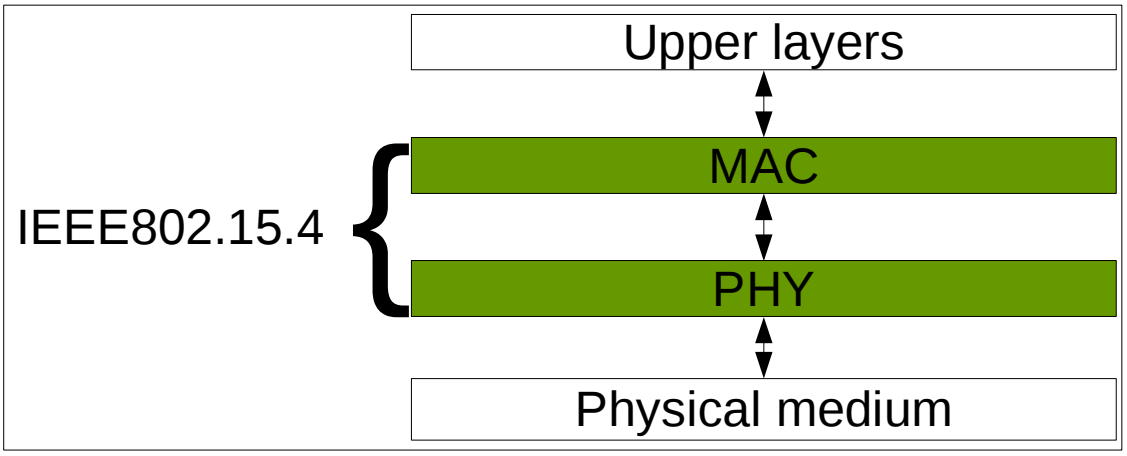

Figure B.2: LR-WPAN device architecture

- **Star topology**: FFD can establish its own network and become the PAN coordinator. All star networks operate independently from all other star networks currently in operation.
- **Peer-to-Peer**: Each device is capable of communicating with any other device within its radio communications range. One device is nominated as the PAN coordinator, for instance, by virtue of being the first device to communicate on the channel.

Several protocols have been built on top of the IEEE 802.15.4 protocol. A brief overview of the most commonly used is given below.

- **ZigBee** is a low data rate, low power consumption, low cost, wireless networking protocol targeted towards automation and remote control applications [ZigBee, 2015].
- **HART** (Highway Addressable Remote Transducer) is a bi-directional communication protocol that provides data access between intelligent field instruments and host systems. A host can be any software application from technician's hand-held device or laptop to a plant's process control, asset management, safety or other system using any control platform [HART, 2015].
- **ISA100** addresses wireless manufacturing and control systems, developed by the International Society of Automation (ISA).

B.1.3/ NEAR FIELD COMMUNICATION (NFC)

According to NFC Forum [NFC_Forum, 2015], NFC is a standards-based short-range wireless connectivity technology that makes life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and connect electronic devices with a touch. NFC is compatible with hundreds of millions of contactless cards and readers already deployed worldwide. The communication occurs between two compatible devices within few centimeters with 13.56 MHz

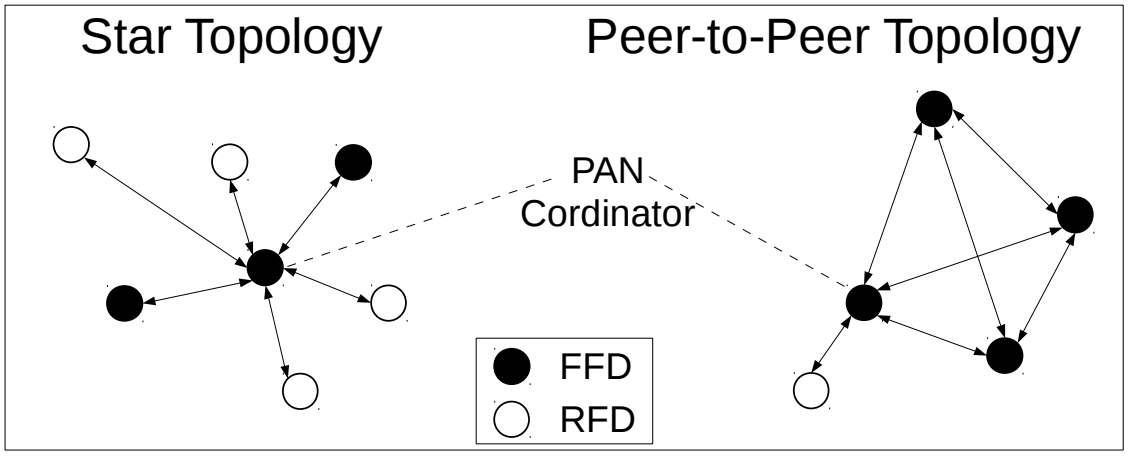

Figure B.3: IEEE 802.15.4 Topologies

operating frequency. NFC protocol supports different data transmission rates such as 106 kbps, 212 kbps, and 424 kbps.
NFC devices can operate in three different modes based on the ISO/IEC 18092, NFC IP-1 and ISO/IEC 14443 contactless smart card standards.

- **Device/Tag (Read/Write):** NFC-enabled device can read or write data to any of the supported tag types in a standard NFC data format.
- **Peer-to-Peer (Communication):** Two NFC-enabled devices can exchange data, which can be setup parameters for different type of connection (such as, Wi-Fi or Bluetooth) or any other type of data.
- **Card Emulation:** NFC-enabled device can be a card or a tag for existing reader.

NFC protocol supports two communication modes (according to the device that generates the RF signal) [Coskun et al., 2013]:

- **Active mode:** both devices use their own energy to generate their own RF field to transmit the data.
- **Passive mode:** only initiator generates the RF field while the target device makes use of the energy that is created by the active device.

B.2/ IETF - CONSTRAINED NETWORKS

IoT aims at integrating constrained devices into the Internet. Constrained Device is a small device with limited CPU, memory, and power resources (also known as sensor, smart object, or smart device). Several constrained devices can constitute a network, becoming "constrained nodes" in that network [LWIG-WG, 2014]. This integration collided with the limitation of the existing Internet technologies, which were not designed for this class of devices.
Constrained devices form constrained networks, the latter have different characteristics (such as, traffic patterns, high packet loss, low throughput, frequent topology changes

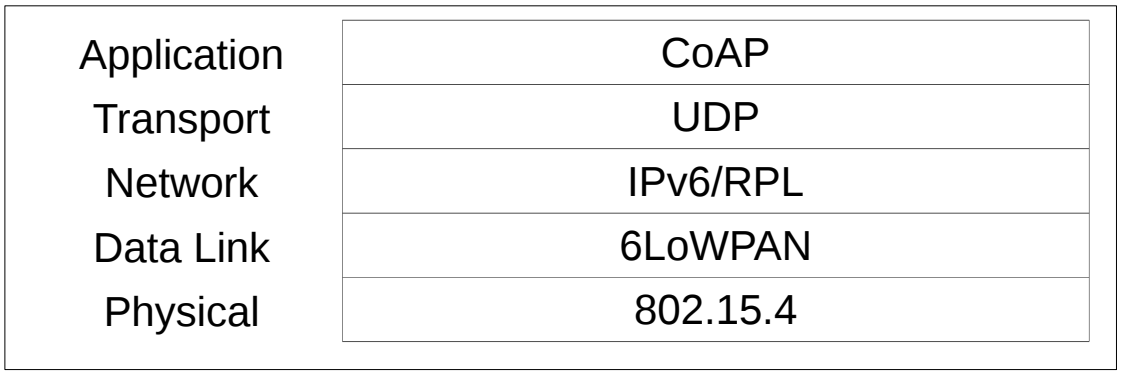

Figure B.4: Constrained Network - Protocol Stack

Name	*data size (e.g., RAM)*	*code size (e.g., Flash)*
Class 0, C0	$\ll$ 10 KiB	$\ll$ 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

Table B.1: Classes of Constrained Devices (KiB = 1024 bytes)

and small useful payload sizes) than the traditional networks [Ishaq et al., 2013]. Each vendor developed a protocol stack to address this problem. This protocol stack bridges the constrained network with the Internet, and performs the routing inside the constrained network itself. Such scenario limits the interoperability between devices from different vendors.

To address the need for a standard protocol stack, the IETF has released three standards (Figure B.4):

- **IPv6 over Low Power WPAN (6LoWPAN).**
- **IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)** : Routing Over Low Power and Lossy Networks (ROLL) [roll WG, 2015].
- **Constrained Application Protocol (CoAP)** : Constrained Restful Environments (CORE) [core WG, 2015].

Due to the popularity of IEEE 802.15.4 [802.15.4, 2011], this standard is used at the physical layer and the medium access control layer. Classes of Constrained Devices and the three IETF standards are briefly described in the following subsections.

B.2.1/ Classes of Constrained Devices

IETF divides Constrained Devices into the following three classes (Table B.1) [LWIG-WG, 2014]:

- **Class 0** devices are very constrained sensor-like motes. They are so severely constrained in memory and processing capabilities that most likely they will not

have the resources required to communicate directly with the Internet in a secure manner. Class 0 devices will participate in Internet communications with the help of larger devices acting as proxies, gateways or servers. They generally cannot be secured or managed comprehensively in the traditional sense. They will most likely be preconfigured (and will be reconfigured rarely, if at all), with a very small data set. For management purposes, they could answer keepalive signals and send on/off or basic health indications.

- **Class 1** devices are quite constrained in code space and processing capabilities, such that they cannot easily talk to other Internet nodes employing a full protocol stack such as using HTTP, TLS and related security protocols and XML-based data representations. However, they have enough power to use a protocol stack specifically designed for constrained nodes (such as CoAP over UDP) and participate in meaningful conversations without the help of a gateway node. In particular, they can provide support for the security functions required on a large network. Therefore, they can be integrated as fully developed peers into an IP network, but they need to be parsimonious with state memory, code space, and often power expenditure for protocol and application usage.

- **Class 2** devices are less constrained and fundamentally capable of supporting most of the same protocol stacks as used on notebooks or servers. However, even these devices can benefit from lightweight and energy-efficient protocols and from consuming less bandwidth. Furthermore, using fewer resources for networking leaves more resources available to applications. Thus, using the protocol stacks defined for more constrained devices also on Class 2 devices might reduce development costs and increase the interoperability.

B.2.2/ IPv6 over Low Power WPAN (6LoWPAN)

Since IPv6 requires support of packet sizes much larger than the largest IEEE 802.15.4 frame size, an adaptation layer is required. The IPv6-over-IEEE 802.15.4 [Montenegro et al.,] document specifies how IPv6 is carried over an IEEE 802.15.4 network with the help of an adaptation layer that sits between the Media Access Control (MAC) layer and the IP network layer. A link in a Low-power Wireless Personal Area Network (LoWPAN) is characterized as lossy, low-power, low-bit-rate, short-range; with many nodes saving energy with long sleep periods [6lowpan WG, 2012].
The 6LoWPAN format specifies the adaptation layer's key elements. 6LoWPAN adaptation layer has four primary elements:

- **Header compression:** IPv6 header fields are compressed by assuming usage of common values. Header fields are elided from a packet when the adaptation layer can derive them from link-level information carried in the 802.15.4 frame or based on simple assumptions of shared context.

- **Fragmentation and Reassembly layer:** the IEEE 802.15.4 data units may be as small as 81 bytes. This is obviously far below the minimum IPv6 packet size of 1280 octets, and in keeping with Section 5 of the IPv6 specification [Deering et al., 1998], a fragmentation and reassembly adaptation layer must be provided at the layer below IP.

- **Stateless auto configuration** Stateless auto configuration is the process where devices inside the 6LoWPAN network automatically generate their own IPv6 address. There are methods to avoid the case where two devices get the same address; this is called duplicate address detection (DAD) [Olsson, 2014].
- **Routing and Forwarding:** The adaptation layer can also be responsible to take routing and forwarding decisions instead of the network layer. Depending on which layer is in charge of routing and packet forwarding, 6LoWPAN divides routing in two schemes: *mesh under* if routing is done at the adaptation layer and *route over* if done at the network layer [Ludovici et al., 2010].

B.2.3/ IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL)

There are two main points that distinguish LLNs from the traditional networks. Firstly, they consist largely of constrained nodes (LLNs may potentially comprise up to thousands of nodes). Secondly, the traffic patterns are not simply point-to-point, but in many cases point-to-multipoint or multipoint-to-point. These characteristics offer unique challenges to a routing solution (such as AODV, OLSR or OSPF).
In order to address this challenge, IETF formed new working group ROLL (Routing Over Low Power and Lossy Networks) [roll WG, 2015], which released IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [RFC6650, 2012].
RPL is a Distance Vector IPv6 routing protocol for LLNs that specifies how to build a Destination Oriented Directed Acyclic Graph (DODAG) using an objective function and a set of metrics/constraints. The objective function operates on a combination of metrics and constraints to compute the best path. There could be several objective functions in operation on the same node and mesh network because deployments vary greatly with different objectives and a single mesh network may need to carry traffic with very different requirements of path quality [IPSO, 2011].

B.2.4/ Constrained Restful Environments (CORE)

CoRE (Constrained RESTful Environments) working group is providing a framework for resource-oriented applications intended to run on constrained IP networks. Constrained Devices *Class 1* are in the focus of the CoRE working group, yet *Class 2* devices can still benefit from lightweight and energy-efficient protocols to free resources for the application or reduce operational costs. CoRE defined a Constrained Application Protocol (CoAP) for the manipulation of Resources on a Device.
First, we are going to discuss REST architecture, which plays an important role in the web services in traditional Internet and IoT. Then, a brief study follows to describe CoAP.

B.2.4.1/ Representational State Transfer (REST)

REST is an architectural style consisting of the set of constraints applied to elements within the architecture. By examining the impact of each constraint as it is added to the evolving style, we can identify the properties induced by the Web's constraints. Additional constraints can then be applied to form a new architectural style that better reflects the

desired properties of a modern Web architecture [Fielding, 2000].
REST is neither a product nor a standard, REST is an architecture that describes how the Web should work. REST has gained widespread acceptance across the Web as a simpler alternative to SOAP- (Simple Object Access Protocol) and WSDL (Web Services Description Language)-based Web services. REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.

REST constraints are design rules that are applied to establish the distinct characteristics of the REST architectural style.

- **Client-Server:** A server component, offering a set of services, listens for requests upon those services. A client component, desiring that a service be performed, sends a request to the server via a connector. The server either rejects or performs the request and sends a response back to the client.
- **Stateless:** Communication must be stateless in nature, each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
- **Cache:** In order to improve network efficiency, the data within a response to a request should be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- **Uniform Interface:** The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components, By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. Implementations are decoupled from the services they provide, which encourages independent evolvability. The trade-off, though, is that a uniform interface degrades efficiency, since information is transferred in a standardized form rather than one which is specific to an application's needs. The REST interface is designed to be efficient for large-grain hypermedia data transfer, optimizing for the common case of the Web, but resulting in an interface that is not optimal for other forms of architectural interaction.
- **Layered System:** The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot "see" beyond the immediate layer with which they are interacting. Layers can be used to encapsulate legacy services and to protect new services from legacy clients, simplifying components by moving infrequently used functionality to a shared intermediary. Intermediaries can also be used to improve system scalability by enabling load balancing of services across multiple networks and processors.
- **Code-On-Demand:** REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented. Allowing features to be downloaded after deployment improves system extensibility. However, it also reduces visibility, and thus is only an optional constraint within REST.

REST	*HTTP*
Create resource	POST
Retrieve resource	GET
Change resource	PUT
Delete resource	DELETE

Table B.2: REST operations mapping

Basic REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods (POST,GET,PUT and Delete), according to the mapping in (Table B.2) [Rodriguez, 2008].

B.2.4.2/ CONSTRAINED APPLICATION PROTOCOL (COAP)

The Constrained Application Protocol (CoAP) [RFC7252, 2014] is a specialized web transfer protocol for use with constrained nodes and constrained networks. The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation. CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs (Uniform Resource Identifier) and Internet media types. CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialized requirements such as multicast support, very low overhead, and simplicity for constrained environments. A CoAP request is equivalent to that of HTTP and is sent by a client to request an action (using a Method Code) on a resource (identified by a URI) on a server. The server then sends a response with a Response Code; this response may include a resource representation. CoAP is a single protocol, but logically, CoAP uses a two-layer approach:

- **Messaging Layer:** CoAP uses it to deal with UDP and the asynchronous nature of the interactions. CoAP defines four types of messages:
  - **Confirmable (CON):** These are the messages that require an acknowledgement. When no packets are lost, each Confirmable message elicits exactly one return message of type Acknowledgement or type Reset.
  - **Non-confirmable (NON):** Some messages do not require an acknowledgement. This is particularly true for messages that are repeated regularly for application requirements, such as repeated readings from a sensor.
  - **Acknowledgement (ACK):** An Acknowledgement message acknowledges that a specific Confirmable message arrived. The Acknowledgement message may also carry a Piggybacked Response.
  - **Reset (RST):** A Reset message indicates that a specific message (Confirmable or Non-confirmable) was received, but some context is missing to properly process it. Provoking a Reset message (e.g., by sending an Empty Confirmable message) is also useful as an inexpensive check of the liveness of an endpoint ("CoAP ping").
- **Request/Response:** CoAP request and response semantics are carried in CoAP messages, which include either a Method Code or Response Code, respectively.

Requests can be carried in Confirmable and Non-confirmable messages, and responses can be carried in these as well as piggybacked in Acknowledgement messages.

CoAP provides the support for other functions such as:

- **Caching:** CoAP endpoints MAY cache responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. The goal of caching in CoAP is to reuse a prior response message to satisfy a current request.
- **Intermediaries or Proxying:** A proxy is a CoAP endpoint that can be tasked by CoAP clients to perform requests on their behalf. This may be useful, for example, when the request could otherwise not be made, or to service the response from a cache in order to reduce response time and network bandwidth or energy consumption.
  - **Forward-Proxy:** An endpoint selected by a client, usually via local configuration rules, to perform requests on behalf of the client, doing any necessary translations. Some translations are minimal, such as for proxy requests for "coap" URIs, whereas other requests might require translation to and from entirely different application-layer protocols.
  - **Reverse-Proxy:** An endpoint that stands in for one or more other server(s) and satisfies requests on behalf of these, doing any necessary translations. Unlike a forward-proxy, the client may not be aware that it is communicating with a reverse-proxy; a reverse-proxy receives requests as if it was the origin server for the target resource.
  - **Cross-Proxy:** A cross-protocol proxy, or "cross-proxy" for short, is a proxy that translates between different protocols, such as a CoAP-to-HTTP proxy or an HTTP-to-CoAP proxy. While this specification makes very specific demands of CoAP-to-CoAP proxies, there is more variation possible in cross-proxies.
- **Resource Discovery:** The discovery of resources offered by a CoAP endpoint is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. A CoAP client can ask a CoAP server about which resources it offers.
- **Multicast:** CoAP supports making requests to an IP multicast group. CoAP endpoints that offer services that they want other endpoints to be able to find using multicast service discovery join one or more of the appropriate all-CoAP-node multicast addresses (IPv4:224.0.1.187, IPv6:FF0X::FD).

B.3/ IoT-A (Internet-of-Things Architecture)

The European Lighthouse Integrated Project has addressed for three years (2010-2013) the Internet-of-Things Architecture. The IoT-A project based its work on the current state of the art, rather than using a clean-slate approach. Due to this choice, common traits are derived to form the base line of the IoT Architectural Reference Model (ARM). ARM

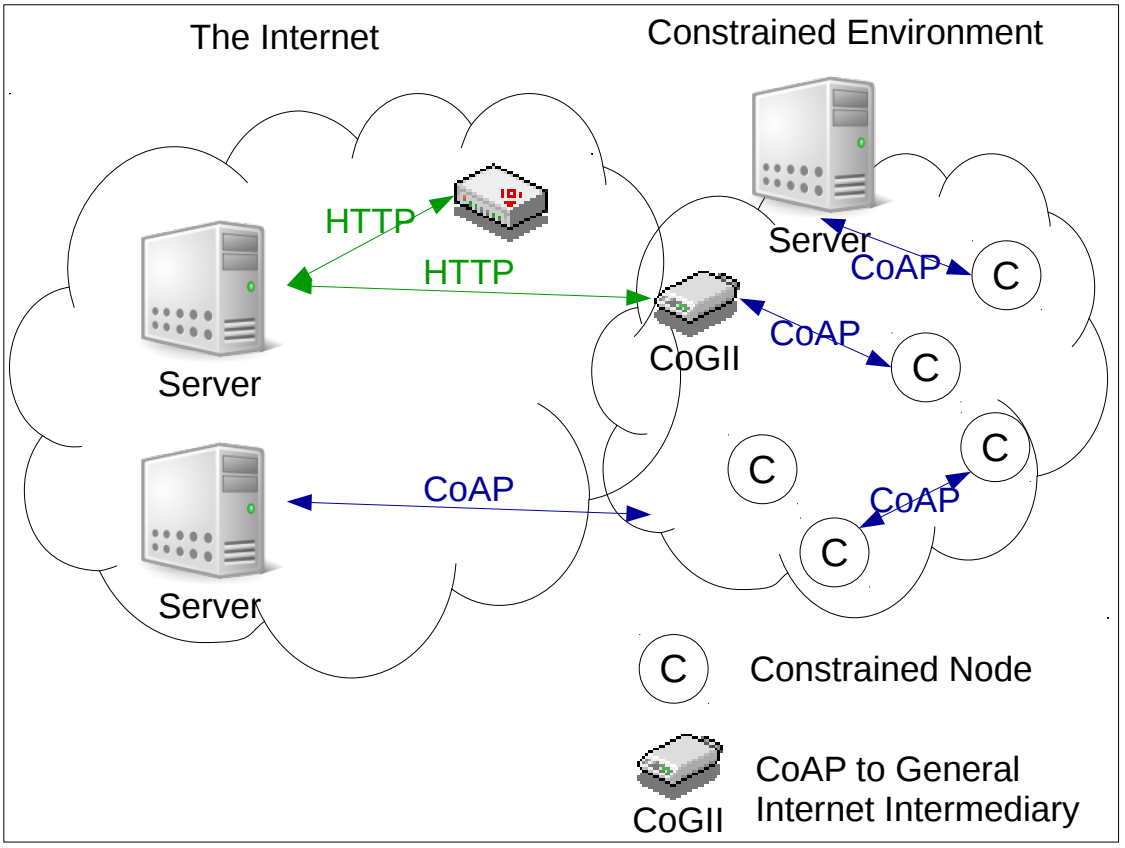

Figure B.5: The Constrained RESTful Environment architecture

is built as an abstract design, which can be used by organizations to create compliant IoT architectures in different application domains.

The IoT ARM consists of three parts (Figure B.6.):

1. **IoT Reference Model** provides the highest abstraction level for the definition of the IoT Architectural Reference Model. It promotes a common understanding of the IoT domain.

2. **IoT Reference Architecture** is the reference for building compliant IoT architectures. As such, it provides views and perspectives on different architectural aspects that are of concern to stakeholders of the IoT.

3. **IoT Guidelines** aim at explaining the usage of the IoT ARM.

Studying the ARM in details is out of the scope of this thesis. Our goal is to clarify the complexity of the IoT systems. This goal can be achieved by studying the IoT reference model.
Reference Model provides the highest abstraction level for the definition of the IoT Architectural Reference Model. It consists of four sub-models: Domain Model, Information Model, Functional Model and Communication Model, which are described in the following.

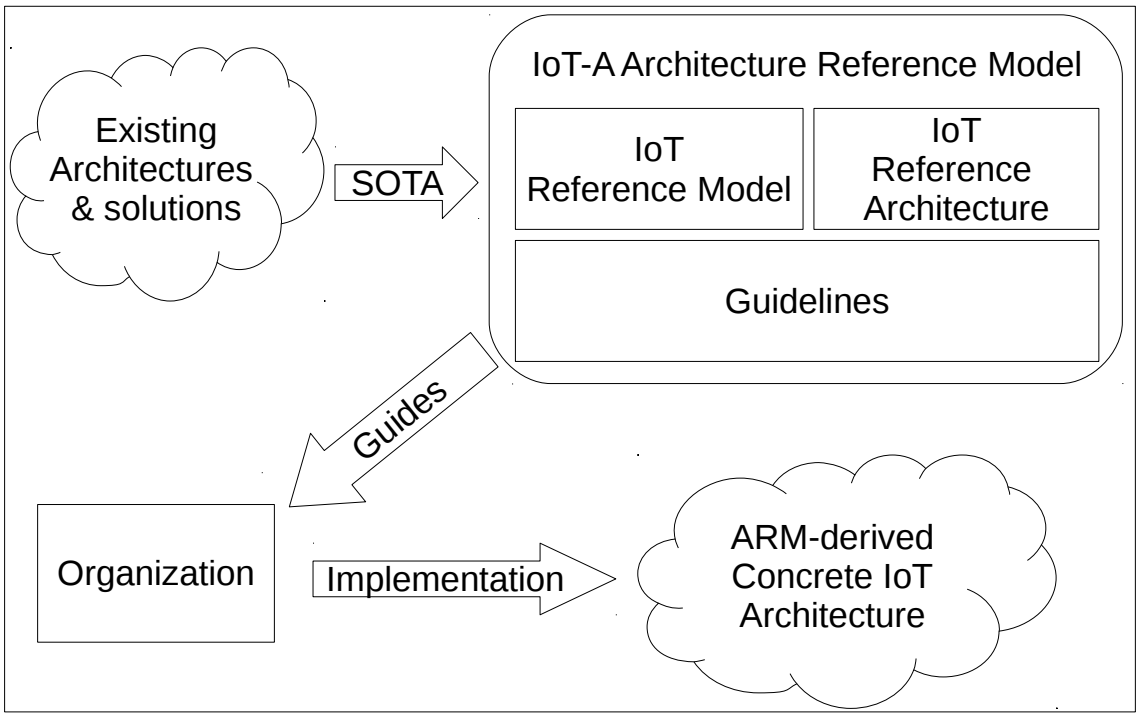

Figure B.6: IoT Architectural Reference Model building blocks

B.3.1/ DOMAIN MODEL

The IoT-A defines a domain model as a description of concepts belonging to a particular area of interest, in addition to the responsibilities of these concepts. Domain Model defines the relationships between the concepts, and models the data exchange between them. Therefore, several other parts of the IoT Reference Model, for instance the IoT Information Model, directly depend on the IoT Domain Model.
Since the technologies used will change over time, Domain Model does not include particular technologies, but rather abstractions thereof.
Concepts can be any physical or virtual entities. The Physical Entity is an identifiable part of the physical environment, which means that it can be almost any object; from humans or animals to cars; from store or logistics chain items to computers; from electronic appliances to clothes. Physical Entities are represented in the digital world by a Virtual Entity.
The Virtual Entity can be a 3D model, avatar, database entry, object (or instance of a class in an object-oriented programming language), and even a social-network account because it digitally represents certain aspects of its human owner, such as a photograph or a list of his/her hobbies. An Augmented Entity is the composition of one Virtual Entity and the Physical Entity it is associated to. There are two fundamental properties for Virtual Entities: firstly, Virtual Entity can represent only one Physical Entity, while a Physical Entity can be associated to several Virtual Entities. Secondly, Virtual Entities are synchronized representations of a given set of aspects (or properties) of the Physical Entity (e.g., sensor). In the same way, changes that affect the Virtual Entity could manifest themselves in the Physical Entity (e.g., actuator).

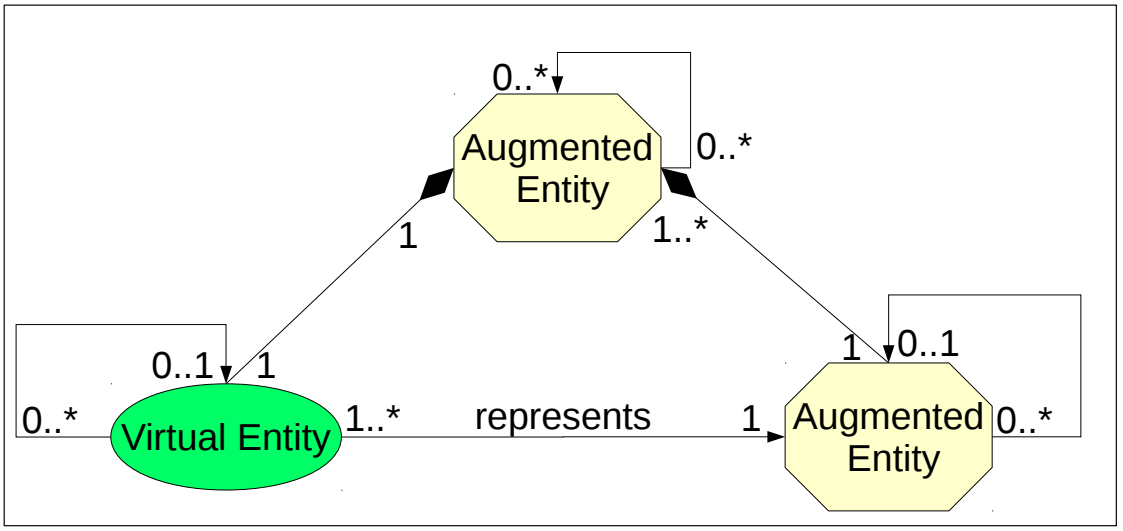

Figure B.7: IoT ARM Entities

The interactions between Physical Entities (that have no projections in the digital world) and Virtual Entities (which have no projections in the physical world) are done by utilizing Devices. Devices are thus technical objects for bridging the real world of Physical Entities with the digital world of the Internet. A device can be aggregations of several Devices of different types, and it can be a Physical Entity, especially in the context of certain applications. An example for such an application is Device management, whose main concern is the Devices themselves and not the entities or environments that these Devices monitor. From an ARM point of view, the following three basic types of Devices are of interest:

- **Sensors:** provide information, knowledge, or data about the Physical Entity they monitor.

- **Tags:** are used to identify Physical Entities, to which the Tags are usually physically attached. The primary purpose of Tags is to facilitate and increase the accuracy of the identification process. Tags can be optical, as in the case of barcodes and QR codes (Quick Response Code), or it can be RF-based (Radio-Frequency Based), as in the case of microwave car-plate recognition systems and RFID (Radio-Frequency Identification).

- **Actuators:** can modify the physical state of a Physical Entity, like changing the state (translate, rotate, stir, inflate, switch on/off,...) of simple Physical Entities or activating/deactivating functionalities of more complex ones.

Since that the Devices can be aggregations of several Devices of different types, the same Device can contains both Sensors (e.g., movement sensing) as well as Actuators (e.g., wheel engines). Figure B.8 provides UML representation of the device types in IoT.

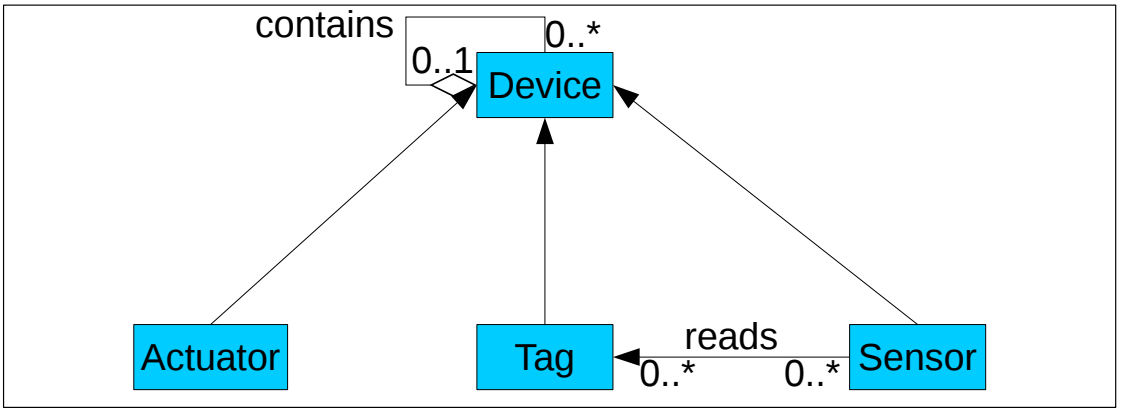

Figure B.8: IoT ARM Devices

Resources are software components that provide some functionality. When associated with a Physical Entity, they either provide some information about or allow changing some aspects in the digital or physical world pertaining to one or more Physical Entities. ARM defines two types of resources:

- **On-Device Resources:** can run on a device, and they are typically sensor resources that provide sensing data or actuator resources. On-Device Resources may also be storage Resources, e.g.,store a history of sensor measurements.

- **Network Resources:** run on a dedicated server in the network or in the “cloud”, they do not rely on special hardware.

IoT Services provide well-defined and standardized interfaces, hiding the complexity of accessing the resources and the virtual entities. Three different types of services can be identified in IoT:

- **Resource-level Services:** expose the functionality of a Device by accessing its hosted Resources, or read/modify data in case of Network resources.

- **Virtual Entity-level Services:** provide access to read information from a Virtual Entity-level, or for updating attributes in order to trigger associations.

- **Integrated Services:** are the result of a Service composition of Resource-level or Virtual Entity-level Services as well as any combinations of both Service abstractions.

Figure B.9 clarifies the relationships between all the mentioned concepts: Physical Entity, Virtual Entity, Device, Resource and Service.

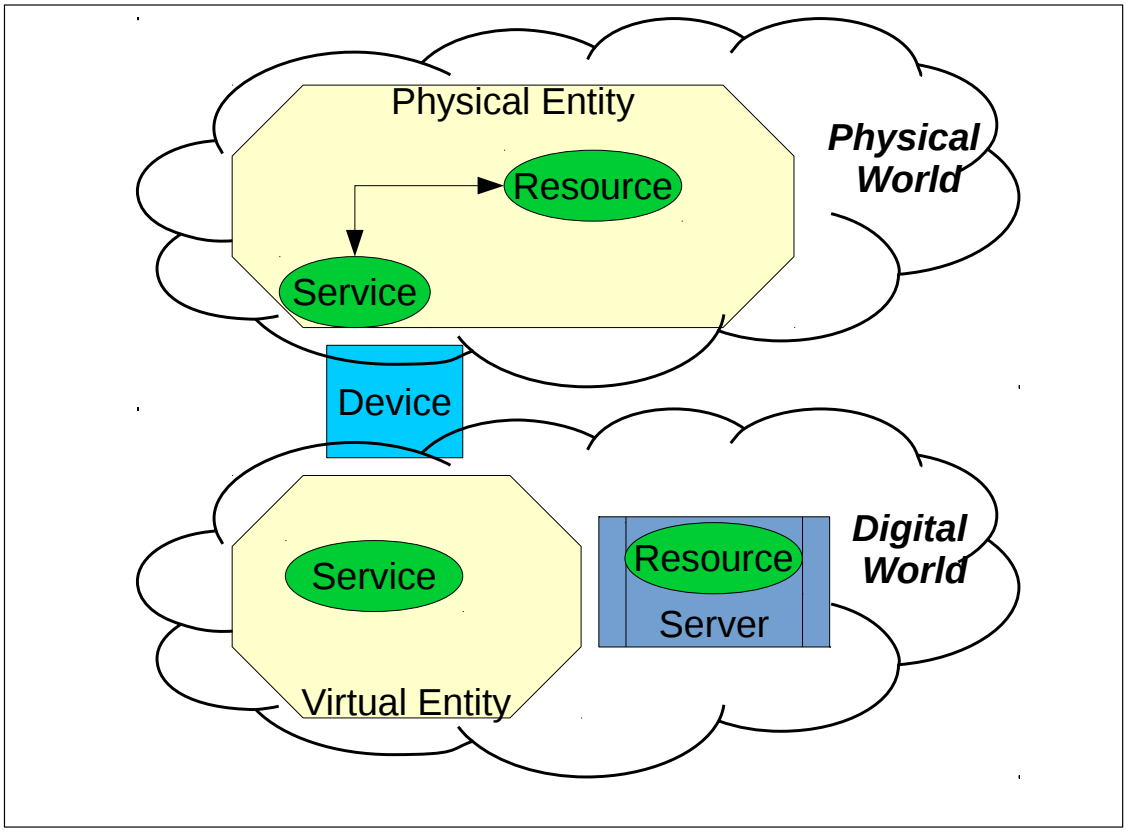

Figure B.9: IoT ARM Domain Concepts Relationships

B.3.2/ INFORMATION MODEL

The IoT Information Model details the modeling of a Virtual Entity. Information model details the association between a virtual entities and a service. The IoT Information Model models all the concepts of the Domain Model that are to be explicitly represented and manipulated in the digital world.
The IoT Information Model provides the basis for all aspects of the system that deal with the representation, gathering, processing, storage and retrieval of information and as such is used as a basis for defining the functional interfaces of the IoT system.
In addition to modeling the virtual entities, all other information related models can be illustrated and structured using Information Model. The following list summarizes these models:

- **Entity Model** specifies which attributes and features of real word objects are represented by the virtual entity.
- **Resource Model** contains the information that is essential to identify Resources by a unique identifier and to classify Resources by their type, like sensor, actuator, processor or tag.
- **Service Description Model** describes a Service, using for instance a service description language such as USDL (Unified Service Description Language) world.

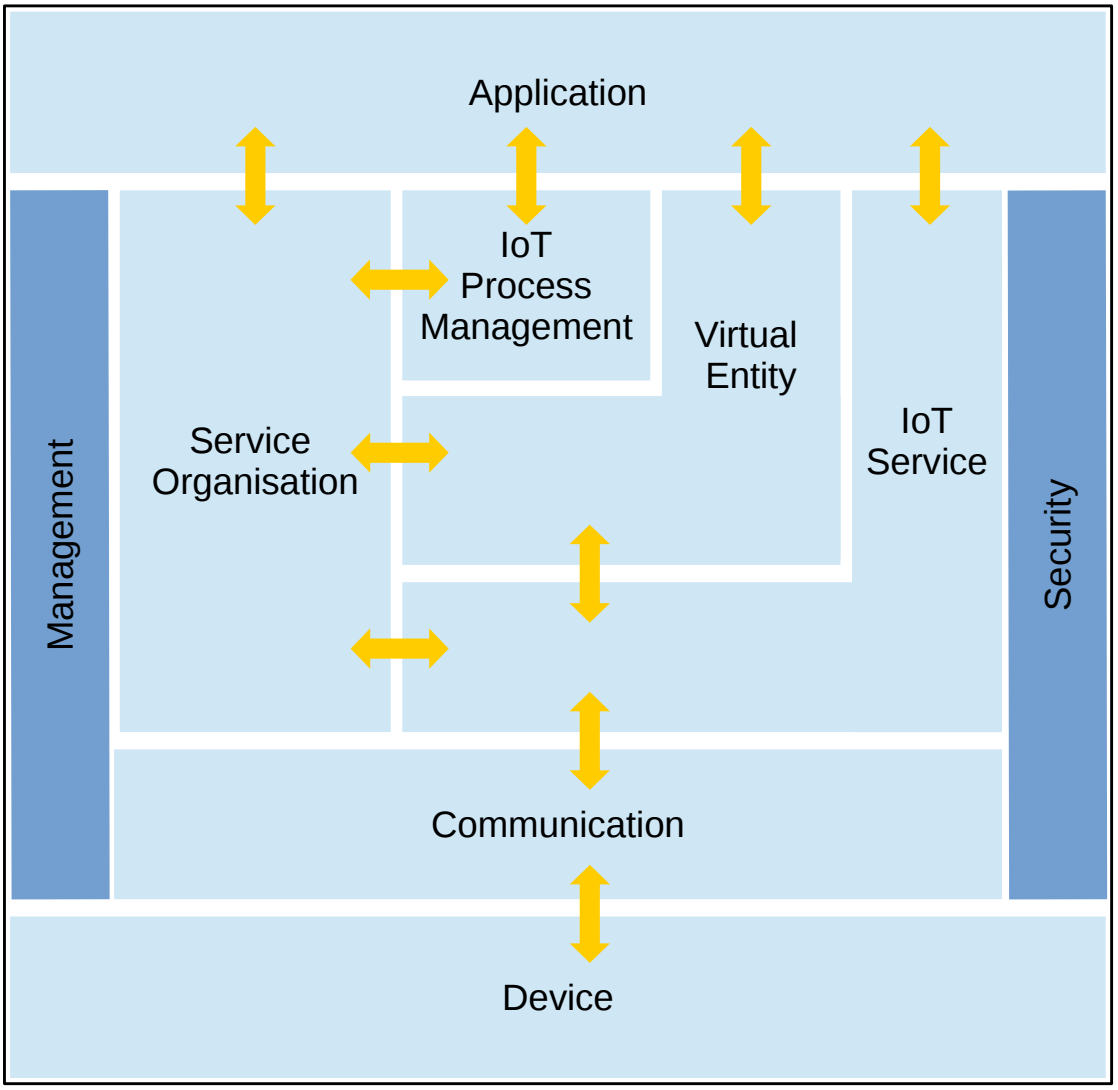

Figure B.10: IoT-A Functional Model

- **Event Model** are used to track dynamic changes in a (software) system, showing who or what has triggered it and when, where and why the change occurred.

B.3.3/ FUNCTIONAL MODEL

The Functional Model is an abstract framework for understanding the main Functionality Groups (FG) and their interactions. This framework defines the common semantics of the main functionalities and will be used for the development of IoT-A compliant Functional Views [Brown et al., 2006].

The IoT Functional Model contains seven longitudinal Functionality complemented by two transversal Functionality (Management and Security) (Figure B.10):

- **Application** is a set of one or more programs designed to permit the user to perform a group of coordinated functions, tasks, or activities. Application software cannot

run on itself but is dependent on system software to execute [pcmag, 2015].

- **Service Organization** is a central Functionality Group that acts as a communication hub between several other Functionality Groups. It effectively links the Service requests from high level FGs such as the IoT Process Management FG, or even external applications, to basic services that expose Resources. It is responsible for resolving and orchestrating IoT Services and it also deals with the composition and choreography of Services.
- **IoT Process Management** provides the functional concepts necessary to conceptually integrate the idiosyncrasies of the IoT world into traditional (business) processes.
  Applications that interact with the IoT Process Management FG can effectively be shielded from IoT-specific details of lower layers of the functional model, which greatly reduces integration costs and thus contributes to an increased adoption of IoT-A based IoT systems [Meyer et al., 2011].
- **Virtual Entity** contains functions for interacting with the IoT System, as well as functionalities for discovering and looking up services that can provide information about virtual entities, and functionalities to interact with them.
- **IoT Service** contains IoT Services as well as functionalities for discovery, look-up, and name resolution of IoT Services.
- **Communication** provides a simple interface for instantiating and for managing high-level information flow.
- **Device:** a Device can be as simple and limited as a Tag and as complex and powerful as a server. It was already introduced in Domain Model (Section B.3.1, p.179).
- **Management** combines all functionalities that are needed to govern an IoT system.
- **Security** is responsible for ensuring the security and privacy of IoT-A-compliant systems.

IoT-A Functional Model sheds light on the complexity of IoT systems, which complicates testing the applications that work on these systems.

B.3.4/ Communication Model

IoT-A proposes a Communication Model that leverages on the ISO OSI 7-layer [ISO, 2015] model for networks, and aims at highlighting the inter-operations among different stacks using methods such as application layer gateways, transparent proxy, network virtualization, etc. The IoT Communication Model helps to model and to analyze how constrained Devices can actively participate in an IoT-A compliant communication and to study possible solutions, such as the usage of application layer gateways, to integrate legacy technologies.

Document generated with LaTeX and:
the LaTeX style for PhD Thesis created by S. Galland — http://www.multiagent.fr/ThesisStyle
the tex-upmethodology package suite — http://www.arakhne.org/tex-upmethodology/

Abstract:

Researchers across many domains are working to provide solutions that enable integration of objects and systems into the Internet of Things (IoT). There are two domains which are among the most important ones in IoT. First domain is IoT testing and evaluation. The billions of objects which are connected to IoT would intercommunicate without any human intervention. Enterprises and developers should be able to test and evaluate different operational scenarios of their systems in different environments. Testing environments should be able to exchange services. We present our architecture IoTaaS (Internet of Things Testing As A Service). IoTaaS is a hierarchy of a distributed cloud for testing and evaluating IoT. We also present the pilot implementation of IoTaaS under the code name CEMAT (Cloud Environment for Mobile Application Testing) which focuses on mobile devices and their connected objects. The second domain is about communication methods and techniques. Several IoT applications depend on public data exchange. We present our design to exchange small amount of public data on wireless network without establishing a connection. COLDE (Connectionless Data Exchange) is our extension to protocol the IEEE 802.11. COLDE utilizes the management frames to allow Wi-Fi devices and access points to exchange small amounts of data without having any association. COLDE describes how we can exchange data without being connected. Lightweight services concept explains which data could be transferred using COLDE. Simulation experiments and real world implementation are presented along with their results.

Keywords: Internet of Things, IoT, IoT testbed, Testing and Evaluating, IoTaaS, CEMAT, Connectionless, Wi-Fi, IEEE 802.11, COLDE, lightweight services, Crowd Networking

Résumé :

Les chercheurs au travers de plusieurs domaines essaient de fournir des solutions pour intégrer les objets et les systèmes dans l'internet des objets (IdO ou IoT pour Internet of Things en anglais). Il y a deux domaines parmi les domaines les plus importants d'IdO. Le premier est de tester et évaluer l'IdO. Plusieurs milliards d'objets sont déjà connectés à l'internet. Ces objets peuvent communiquer entre eux directement sans intervention humaine. Les entreprises et les développeurs doivent être capable de tester et d'évaluer des scénarios différents dans plusieurs environnements. Ces environnements doivent être capable d'échanger des services entre eux. Nous présentons notre architecture IoTaaS (Internet of Things Testing As A Service). IoTaaS est un environnement distribué pour tester et évaluer l'IdO. Nous présentons aussi la première mise en œuvre expérimentale qui est appelée CEMAT (Cloud Environment for Mobile Application Testing) qui permet de tester les applications mobiles et les objets connectés. Le deuxième domaine correspond aux méthodes de communication. Nous présentons notre conception de communications COLDE (Connectionless Data Exchange) pour échanger une petite quantité de données publiques entre les clients Wi-Fi et les points d'accès sans qu'ils aient besoin d'être associés. Alors que COLDE présente la méthode nécessaire pour échanger ces données, le concept de services Lightweight décrit les données échangées en utilisant le protocole COLDE.

Mots-clés : Internet des objets, tester et évaluer, sans connexion, sans association

SPIM

■ École doctorale SPIM 16 route de Gray F - 25030 Besançon cedex

■ tél. +33 (0)3 81 66 66 02 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr

UFC UNIVERSITÉ DE FRANCHE-COMTÉ