

Abstract

Nowadays, Internet of things is witnessing a tremendous evolution due to the increasing growth in communication technologies, weather and environmental sensing, health care sensing. Indeed, sensors are being a kind of intelligent mobile agent able to perceive its environment and transmit information to the cloud for processing. This way of perception allow the development of several kinds of applications to enhance human capacity to understand their environment and make appropriate decision. However, developing such advanced applications relies heavily on the quality of the communication between sensors and between sensors and the infrastructure, therefore, such communication can be realized only with the help of a secure data collection and efficient data treatment and analysis.

Data collection in a vehicular network has been always a real challenge due to the specific characteristics of these highly dynamic networks (frequent changing topology, vehicles speed and frequent fragmentation), which lead to opportunistic and non long-lasting communications. Security, remains another weak aspect in these wireless networks since they are by nature vulnerable to various kinds of attacks aiming to falsify collected data and affect their integrity. Furthermore, collected data are not understandable by themselves and could not be interpreted and understood if directly shown to a driver or sent to other nodes in the network. They should be treated and analyzed to extract meaningful features and information to develop reliable applications. In addition, developed applications always have different requirements regarding quality of service (QoS). Several research investigations and projects have been conducted to overcome the aforementioned challenges. However, they still did not meet perfection and suffer from some weaknesses. For this reason, we focus our efforts during this thesis to develop a platform for a secure and efficient data collection and exploitation to provide vehicular network users with efficient applications to ease their travel with protected and available connectivity. Therefore, we first propose a solution to deploy an optimized number of data harvesters to collect data from an urban area. Then, we propose a new secure intersection based routing protocol to relay data to a destination in a secure manner based on a monitoring architecture able to detect and evict malicious vehicles. This protocol is after that enhanced with a new intrusion detection and prevention mechanism to decrease the vulnerability window and detect attackers before they persist their attacks using Kalman filter. In a second part of this thesis, we concentrate on the exploitation of collected data by developing an application able to calculate the most economic itinerary in a refined manner for drivers and fleet management companies. This solution is based on some information that may affect fuel consumption, which are provided by vehicles and other sources in Internet accessible via specific APIs, and targets to economize money and time. Finally, a spatio-temporal mechanism allowing to choose the best available communication medium is developed. This latter is based on fuzzy logic to assess a smooth and seamless handover, and considers collected information from the network, users and applications to preserve high quality of service.

Résumé

ԿԵՐՔ.Ա

ԿԵՐՔ.Ա Ը ՀՕ.Ա ՏՈՎՎԻ.ԾԻ Օ.Ա ՏԵՒՄ ԽԹ.ԾԵՐ ՏՈՒԾ.Ի Կ ՏԵՄ

ملخص

الأفكار الخضراء عديمة اللون تنام بغضب

Acknowledgements

Dedication

Contents

Abstract	i
Acknowledgements	vii
Dedication	ix
Contents	xi
List of Tables	xix
List of Figures	xxi
List of Abbreviations	xxiii
List of Nomenclatures	xxv
List of Publications	xxvii
1 Introduction	1
1.1 Context & motivation	1
1.2 Methodology and contributions	2
1.3 Organization of the thesis	2
2 State of the art [1]	3
2.1 IoT Applications requirements	6
2.1.1 Transportation and logistics	6
2.1.2 Healthcare	6
2.1.3 Smart environnement	6
2.1.4 personal and social	6
2.1.5 Futuristic	6
2.1.6 Summary and discussion	9
2.2 IoT Sensors	9
2.2.1 OS platform	9
2.2.1.1 Contiki	10
2.2.1.2 RIOT	12
2.2.1.3 TinyOS	12
2.2.1.4 freeRTOS	12
2.2.1.5 SDN platforms	13
2.2.1.6 Summary and conclusion	14
2.2.2 Processing Unit	14
2.2.2.1 OpenMote	15

2.2.2.2	MSB430-H	15
2.2.2.3	Zolertia	15
2.2.3	Radio Unit	15
2.2.3.1	Lora Tranceiver	16
2.2.4	Sensing Unit	16
2.2.5	Summary and discussion	16
2.3	IoT Communication protocols	16
2.3.1	Application	16
2.3.1.1	LwM2M	16
2.3.1.2	CBOR	17
2.3.1.3	DTLS	17
2.3.1.4	OSCOAP	17
2.3.1.5	COAP (COnstrained Application Protocol)	17
	1) Overview	17
	2) Coap Methods	17
	3) Coap Transactions	17
	4) Coap Messages	17
2.3.1.6	MQTT	18
2.3.1.7	XMPP	19
2.3.1.8	AMQP	19
2.3.1.9	DDS	19
2.3.1.10	mDNS	20
2.3.2	Network	20
2.3.2.1	6TiSCH	20
2.3.2.2	OLSRv2	20
2.3.2.3	AODVv2	20
2.3.2.4	LoRaWAN	20
2.3.2.5	ROHC	20
2.3.2.6	IPHC	20
2.3.2.7	SCHC	21
2.3.2.8	NHC	21
2.3.2.9	ROLL	21
2.3.2.10	RPL	21
2.3.2.11	6LowPAN	22
	1) Characteristics	22
	2) Encapsulation Header format	22
	3) Fragment Header	22
	4) Mesh addressing header	22
	5) Header compression (RFC4944)	23
	6) Header compression Improved (draft-hui-6lowpan-hc-01)	23
2.3.3	MAC	25
2.3.3.1	Sharing the channel	26
	1) TDMA, FDMA, CDMA, TSMA	26
2.3.3.2	Transmitting information	26
	1) TFDM, TDSSS, TFHSS	26
2.3.4	Radio	26
2.3.4.1	Digital modulation	26
	1) ASK, APSK, CPM, FSK, MFSK, MSK, OOK, PPM, PSK, QAM, SC-FDE, TCM WDM	26

2.3.4.2	Hierarchical modulation	26
1)	QAM, WDM	26
2.3.4.3	Spread spectrum	26
1)	SS, DSSS, FHSS, THSS	26
2.3.5	Summary and discussion	26
2.4	IoT Norms & Standards	26
2.4.1	Divers	26
2.4.1.1	IPLC	26
2.4.1.2	BACnet	26
2.4.1.3	Z-WAze	26
2.4.1.4	Bluetooth LE	26
2.4.2	SigFox	28
2.4.3	IETF	28
2.4.3.1	6LoWPAN	28
2.4.4	3GPP	29
2.4.4.1	NB-IoT	29
2.4.4.2	EC-GSM	29
2.4.4.3	e-MTC	29
2.4.5	IEEE	29
2.4.5.1	IEEE 802.11	29
2.4.5.2	IEEE 802.15.4	29
1)	Physical Layer	29
2)	Definitions	29
3)	Topologies	29
2.4.5.3	ZigBee	30
2.4.6	LoRaWAN	30
2.4.6.1	ALIANCE	31
1)	Class-A	31
1.1)	Uplink	31
1.2)	Downlink	31
1.3)	Confirmed data	31
2)	Class-B	31
2.1)	Downlink	31
2.2)	Confirmed data	32
2.3)	Requirements	32
3)	Class-C	32
3.1)	Downlink	32
3.2)	Confirmed data	32
2.4.6.2	SEMTECH	32
2.4.7	Summary and discussion	36
2.5	IoT SDN	41
2.5.1	Appication	41
2.5.2	Controle	41
2.5.3	Data	41
2.5.4	Summary and discussion	41
2.6	IoT Security	41
2.6.1	Application	41
2.6.1.1	MQTTS	41
2.6.1.2	Blockchain	41

2.6.2	Network	41
2.6.3	MAC/Radio	41
2.6.3.1	LoraWan	41
2.6.3.2	Zigbee	41
2.6.3.3	802.15.4	41
2.6.4	Summary and discussion	41
3	1 [1]	43
3.1	Introduction	44
3.1.1	problem statement	44
3.1.2	Background	44
3.1.3	Purpose (Goal)	45
3.1.4	Limitations	45
3.1.5	Method	45
3.2	Related work	46
3.3	Background	46
3.3.1	Hardware	46
3.3.2	Operating system	47
3.3.3	Communication protocol	48
3.3.4	Workspace and tools	48
3.4	Proposed	48
3.4.1	Drivers and firmware	49
3.4.2	CoAP server	49
3.4.2.1	Testing	50
3.4.2.2	Final prototype	50
3.5	Experimentation	50
3.5.1	Range	50
3.5.2	Response time	51
3.5.3	Connection speed	52
3.5.4	Power consumption	53
3.6	Results	55
3.6.1	Range	55
3.6.2	Response time	55
3.6.3	Connection speed	56
3.6.4	Power consumption	56
3.6.5	Project execution	56
3.7	Discussion	57
4	2 SDN: Sentilo [20]	59
4.1	Introduction	61
4.1.1	Problem Statement	61
4.1.2	Background	61
4.1.3	Purpose (Goal)	61
4.1.4	Limitations	61
4.1.5	Method	61
4.2	Related work	61
4.3	Background	61
4.3.1	Hardware: Zolertia Z1 Motes	62
4.3.1.1	Peripherals ports	62
4.3.1.2	Internal sensors	62

4.3.1.3	1) Temperature Sensor	62
	2) Accelerometer	62
	External Sensors	62
	1) Analog sensors	62
	1.1) Precision Light Sensor	63
	1.2) Force Sensor	63
4.3.1.4	Relay actuator	63
4.3.1.5	Distance sensor	63
4.3.2	Operating systems	63
4.3.3	Communication protocol	63
4.3.3.1	Composition	64
4.3.3.2	RIME	64
4.4	Proposed	65
4.4.1	Definitions	65
4.4.2	Sentilo Architecture	65
4.4.2.1	PubSub Server	65
4.4.2.2	Web Catalog Application	66
4.4.2.3	Extensions (Agents)	66
4.4.3	Sentilo structure	67
4.4.4	Sentilo API	67
4.5	Experimentation	67
4.5.1	Sensor Network	68
4.5.1.1	Border Router	68
4.5.1.2	Nodes	68
4.5.2	Network connector	68
4.5.2.1	Application workflow	68
4.5.2.2	Sensor registration	68
4.5.2.3	Sensor data publish	69
4.6	Results	69
4.6.1	Range	69
4.6.2	Response time	69
4.6.3	Connection speed	69
4.6.4	Power consumption	69
4.7	Discussion	69
4.7.1	Future lines of work	69
5	3 MEC	71
5.1	Introduction	72
5.1.1	Problem Statement	72
5.1.2	Background	72
5.1.3	Purpose (Goal)	72
5.1.4	Limitations	72
5.1.5	Method	72
5.2	Related work	72
5.3	Background	72
5.3.1	Hardware	72
5.3.2	Operating system	72
5.3.3	Communication protocol	73
5.3.4	Workspace and tools	73

5.4	Proposed	73
5.5	Experimentation	73
5.6	Results	73
5.6.1	Range	75
5.6.2	Response time	75
5.6.3	Connection speed	75
5.6.4	Power consumption	75
5.7	Discussion	75
6	Short paper	77
6.1	Introduction	78
6.2	Related work	78
6.3	Background	78
6.4	Prototype Development	78
6.5	Experimentation	78
6.5.1	Range	78
6.5.2	Response time	78
6.5.3	Connection speed	78
6.5.4	Power consumption	78
6.6	Results	78
6.6.1	Range	78
6.6.2	Response time	78
6.6.3	Connection speed	78
6.6.4	Power consumption	78
6.7	Discussion	78
7	Long paper	79
7.1	Introduction	80
7.2	Related work	81
7.2.1	LoRa Overview	81
7.2.1.1	Bandwidth (BW):	81
7.2.1.2	Carrier Frequency (CF):	81
7.2.1.3	Coding Rate (CR):	81
7.2.1.4	Spreading Factor (SF):	82
7.2.2	From Experimental Measurements to Simulations	82
7.2.3	LoRa Evaluation and Limits	82
7.2.4	LoRa Network Deployment Strategies	83
7.3	Background	83
7.4	Approach	83
7.4.1	Spectrum	84
7.4.2	LoRa Modulation	84
7.4.3	Spreading Factors Orthogonality	84
7.4.4	LoRa SX1276 and SX1301 Transceivers	84
7.4.5	MAC and Application Layers	85
7.4.6	Deployment Scenarios	85
7.5	Experimentation	86
7.6	Results exploitation	87
7.6.1	Deployment Scenarios and Assumptions	87
7.6.2	Homogeneous Deployments Scenarios	87
7.6.3	Heterogeneous vs. Homogeneous Deployment Scenarios	87

7.6.4	Application Period and Packet Duration vs. ERC Regulations	88
7.7	Conclusion	88
8	6	91
8.1	Introduction	91
8.2	Related work	91
8.3	Related work	91
8.4	Approach	91
8.5	Experimentation	91
8.6	Results exploitation	91
8.7	Conclusion	92
9	UTLC	93
9.1	Introduction	94
9.2	Related work	95
9.3	Related work	96
9.4	Use Case and Model Design	96
9.4.1	Use case	96
9.4.2	Design Model	97
9.5	Prototyping	100
9.5.1	6LoWPAN, Contiki OS, Re-Mote and Border Router	100
9.5.2	MQTT and UBIDOTS	101
9.6	Results	102
9.7	Conclusion	104
10	Conclusion	107
10.1	Conclusion	107
10.2	Perspectives	107
A	Appendix A	109
A.1	Introduction	109
A.2	Related work	109
A.3	Related work	109
A.4	Approach	109
A.5	Experimentation	109
A.6	Results exploitation	109
A.7	Conclusion	109
B	Appendix B	111
B.0.0.1	CoAP	113
B.0.0.2	MQTT	113
B.0.0.3	XMPP	114
B.0.0.4	AMQP	114
B.0.0.5	DDS	114
B.0.0.6	mDNS	115
Bibliography		117

List of Tables

2.1	Use cases [hancke_role_2012]	7
2.2	Main IoT challenges[2] + [3]	7
2.3	A PPLICATION REQUIREMENTS FOR THE USE CASES OF INTEREST[5] [3].	9
2.4	Common operating systems used in IoT environment [7]	10
2.5	An example table.	13
2.6	SDN-based network and topology management architectures. [10]	14
2.7	Common operating systems used in IoT environment [7]	14
2.8	16
2.9	Standardization efforts that support the IoT	17
2.10	Application protocols comparison	20
2.11	Routing protocols comparison [rpl2]	25
2.12	Routing protocols comparison [rpl2]	25
2.13	25
2.14	uyuyuy	36
2.15	Wireless technologies commonly used in smart buildings [15]	36
2.16	LPWan Characteristics [16]	37
2.17	[17]	38
2.18	[17]	38
2.19	IoT cloud platforms and their characteristics [7]	38
2.20	IEEE 802.15.4 standards [18]	39
2.21	hghg	39
2.22	Receiver sensitivity [dBm]	39
2.23	oioioi	39
2.24	An example table.	40
5.1	Taxonomy of prediction models [short_2007]	73
5.2	Network selection inputs and classification of parameters [21] + QoS pa- rameters [22] [23]	74
7.1	84
7.2	85
7.3	Datasets properties.	86
9.1	Correlation between distributions and empirical results	104
B.1	Social metrics	116

List of Figures

1	Use cases.	6
2	Use cases.	7
3	Use cases.	8
4	802.15.4 use cases [sarwar_iot_].	8
5	.	9
6	contiki.	11
7	LPWAN connectivity.	13
8	.	15
9	LPWAN.	27
10	LPWAN.	27
11	LPWAN.	28
12	uhuhuh.	30
13	Class A.	31
14	Class B.	32
15	Class C.	32
16	LoraWan Parameters.	33
17	.	33
18	.	34
19	.	36
1	Cumulative distribution function of infected users.	86
1	Use case illustration	96
2	Model of our Traffic Lights in UPPAAL	97
3	Model of our Middleware in UPPAAL	98
4	Model of our Cloud variables in UPPAAL	99
5	Architecture of IoT-UTLC	100
6	BR and sink combined on one board	101
7	UTLC network layers	102
8	Normal, Gamma and Logistic distribution	103
9	Cumulative distribution function of RTT delay for two QoS levels	104

List of Abbreviations

List of Nomenclatures

BS Base Station

CCI Co-channel Interference

DC Duty cycle

Jit Jitter

Mob Mobility

PDR Packet delivery ratio

PLR Packet loss rate

PS Payload size

RTT Round time trip

SC Service Cost

SINR Signal-to-interference & noise ratio

SIR Signal-to-Interference Ratio

SL Sleep time

TC Traffic congestion

Th Throughput

ToA Time on Air

Tx Transmission Energy

List of Publications

"There's no absolutely reliable way to achieve a great citation. However, hardworking could be fruitful" - Eraldo Banovac

International Conferences

National Conferences

Journals

Survey

1 | Introduction

"The secret of a good sermon is to have a good beginning and a good ending, then having the two as close as possible" - George Burns

Contents

1.1	Context & motivation	1
1.2	Methodology and contributions	2
1.3	Organization of the thesis	2

1.1 Context & motivation

The exponential growth of 5G networks and the development of IoT that will greatly come with it, would considerably raise the number of Smart Cities applications. The aim of such technology is mainly to improve the comfort and the safety of users through wireless IoT networks. Wireless Sensor Networks (WSNs) are the source of sensed data of cities things, i.e. roads, cars, pedestrians, houses, parking, etc. The cloud is the entity that collects the sensed data and allows users and machines to do data analysis and improve services. For Smart Cities, one objective is improving the welfare of citizens as well as its safety getting real-time information about the city infrastructure. One application would be the transportation systems, and traffic lights control having as an objective avoids congestion and dangerous situations. A static cycle of traffic lights has a direct impact on traffic jams. The long period at red or green light could impact the fluidity of the city traffic. The Internet of Things (IoT) would give an answer to the required interoperability between heterogeneous wireless networks. Our objective is to model, prototype and evaluate a traffic control system. Indeed, different infrastructures have different purposes and technologies, this means that it is not possible to state communication between two infrastructures following a Device-to-Device approach. However, thinking of an indirect or Device-to-Cloud communication between infrastructures seems useful when every connected system has its own technologies, e.g. Zigbee, LoRa, SigFox, ITS-G5. Consequently, IP stack would be the suitable mediator for interconnecting these networks. It removes the barriers of rigid standard specifications of the hardware despite the overhead of the extra network configuration. Furthermore, we want to have a scalable solution not limited only on the traffic light management system. We can deploy sensors and actuators to measure noise or air pollution via panels or roads and offer new services, e.g. where and when jogging is better. To implement our Urban Traffic Light Control based on an IoT network architecture

(IoT-UTLC), we setting a real IEEE 802.15.4 WSN devices that would act as actuators and sensors. All these small traffic light devices are driven by a Border Router (BR) which is a gateway to the Internet. This BR

1.2 Methodology and contributions

1.3 Organization of the thesis

2 | State of the art [1]

"Given one hour to save the planet, I would spend 59 minutes understanding the problem and one minute resolving it." - Albert Einstein

Contents

2.1	IoT Applications requirements	6
2.1.1	Transportation and logistics	6
2.1.2	Healthcare	6
2.1.3	Smart environnement	6
2.1.4	personal and social	6
2.1.5	Futuristic	6
2.1.6	Summary and discussion	9
2.2	IoT Sensors	9
2.2.1	OS platform	9
2.2.1.1	Contiki	10
2.2.1.2	RIOT	12
2.2.1.3	TinyOS	12
2.2.1.4	freeRTOS	12
2.2.1.5	SDN platforms	13
2.2.1.6	Summary and conclusion	14
2.2.2	Processing Unit	14
2.2.2.1	OpenMote	15
2.2.2.2	MSB430-H	15
2.2.2.3	Zolertia	15
2.2.3	Radio Unit	15
2.2.3.1	Lora Tranceiver	16
2.2.4	Sensing Unit	16
2.2.5	Summary and discussion	16
2.3	IoT Communication protocols	16
2.3.1	Application	16
2.3.1.1	LwM2M	16

2.3.1.2	CBOR	17
2.3.1.3	DTLS	17
2.3.1.4	OSCOAP	17
2.3.1.5	COAP (COnstrained Application Protocol)	17
1)	Overview	17
2)	Coap Methods	17
3)	Coap Transactions	17
4)	Coap Messages	17
2.3.1.6	MQTT	18
2.3.1.7	XMPP	19
2.3.1.8	AMQP	19
2.3.1.9	DDS	19
2.3.1.10	mDNS	20
2.3.2	Network	20
2.3.2.1	6TiSCH	20
2.3.2.2	OLSRv2	20
2.3.2.3	AODVv2	20
2.3.2.4	LoRaWAN	20
2.3.2.5	ROHC	20
2.3.2.6	IPHC	20
2.3.2.7	SCHC	21
2.3.2.8	NHC	21
2.3.2.9	ROLL	21
2.3.2.10	RPL	21
2.3.2.11	6LowPAN	22
1)	Characteristics	22
2)	Encapsulation Header format	22
3)	Fragment Header	22
4)	Mesh addressing header	22
5)	Header compression (RFC4944)	23
6)	Header compression Improved (draft-hui-6lowpan-hc-01)	23
2.3.3	MAC	25
2.3.3.1	Sharing the channel	26
1)	TDMA, FDMA, CDMA, TSMA	26
2.3.3.2	Transmitting information	26
1)	TFDM, TDSSS, TFHSS	26
2.3.4	Radio	26
2.3.4.1	Digital modulation	26
1)	ASK, APSK, CPM, FSK, MFSK, MSK, OOK, PPM, PSK, QAM, SC-FDE, TCM WDM	26

2.3.4.2	Hierarchical modulation	26
1)	QAM, WDM	26
2.3.4.3	Spread spectrum	26
1)	SS, DSSS, FHSS, THSS	26
2.3.5	Summary and discussion	26
2.4	IoT Norms & Standards	26
2.4.1	Divers	26
2.4.1.1	IPLC	26
2.4.1.2	BACnet	26
2.4.1.3	Z-WAze	26
2.4.1.4	Bluetooth LE	26
2.4.2	SigFox	28
2.4.3	IETF	28
2.4.3.1	6LoWPAN	28
2.4.4	3GPP	29
2.4.4.1	NB-IoT	29
2.4.4.2	EC-GSM	29
2.4.4.3	e-MTC	29
2.4.5	IEEE	29
2.4.5.1	IEEE 802.11	29
2.4.5.2	IEEE 802.15.4	29
1)	Physical Layer	29
2)	Definitions	29
3)	Topologies	29
2.4.5.3	ZigBee	30
2.4.6	LoaraWAN	30
2.4.6.1	ALIANCE	31
1)	Class-A	31
2)	Class-B	31
3)	Class-C	32
2.4.6.2	SEMTECH	32
2.4.7	Summary and discussion	36
2.5	IoT SDN	41
2.5.1	Appication	41
2.5.2	Controle	41
2.5.3	Data	41
2.5.4	Summary and discussion	41
2.6	IoT Security	41
2.6.1	Application	41
2.6.1.1	MQTT	41

2.6.1.2	Blockchain	41
2.6.2	Network	41
2.6.3	MAC/Radio	41
2.6.3.1	LoraWan	41
2.6.3.2	Zigbee	41
2.6.3.3	802.15.4	41
2.6.4	Summary and discussion	41

2.1 IoT Applications requirements

2.1.1 Transportation and logistics

2.1.2 Healthcare

2.1.3 Smart environnement

2.1.4 personal and social

2.1.5 Futuristic



Figure 1: Use cases.

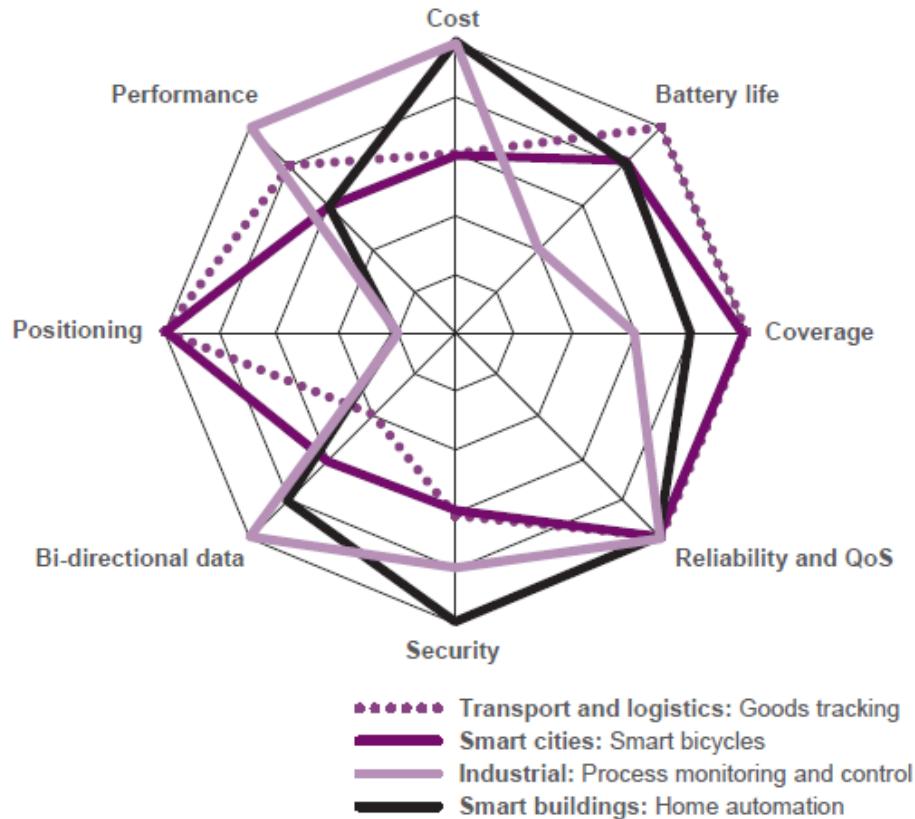


Figure 2: Use cases.

Use cases				
Health Monitoring				
Water Distribution				
Electricity Distribution				
Smart Buildings				
Intelligent Transportation				
Surveillance				
Environmental Monitoring				

Table 2.1: Use cases [hancke_role_2012]

Challenges-Applications	Gids	EHealth	Transportations	Cities	Building
Ressources cistrainsts	+	+++	-	++	+
Mobility	+	++	+++	+++	-
Heterogeneity	++	++	++	+++	+
Scalability	+++	++	+++	+++	++
QoS cistrainsts	++	++	+++	+++	+++
Data management	++	+	+++	+++	++
Lack of standardization	++	++	++	++	+++
Amount of attacks	+	+	+++	+++	+++
Safety	++	++	+++	++	+++

Table 2.2: Main IoT challenges[2] + [3]

voir [4]

Smart systems in smart cities [6]

- ➡ Smart Mobility
- ➡ Smart semaphores controle
- ➡ Smart Red Swarm
- ➡ Smart panels
- ➡ Smart bus scheduling
- ➡ Smart EV management

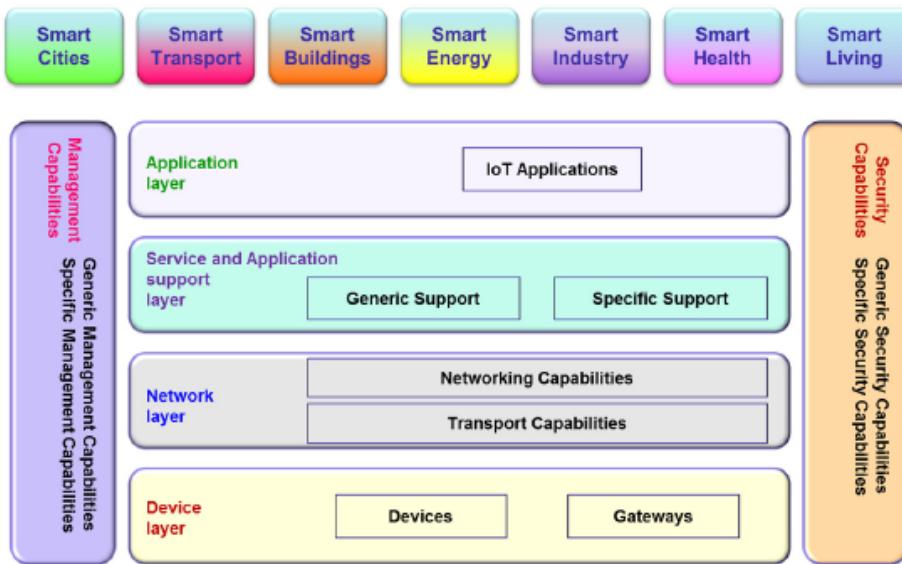


Figure 3: Use cases.

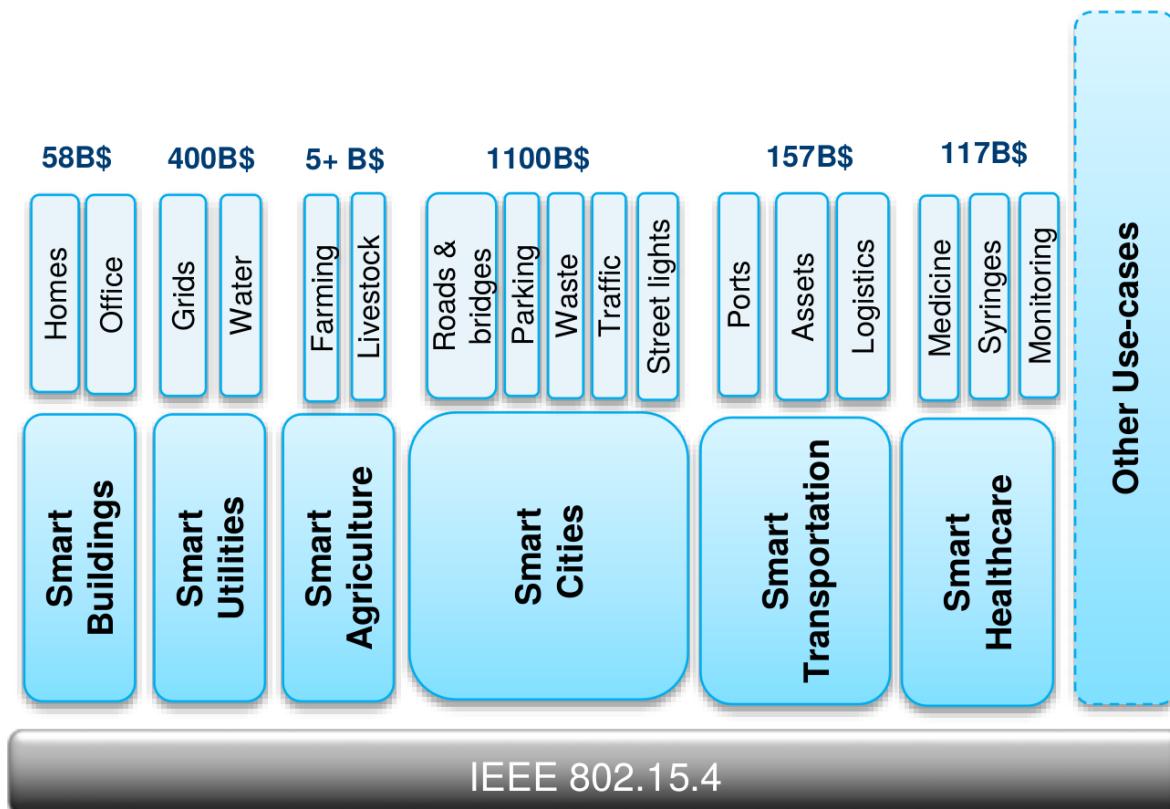


Figure 4: 802.15.4 use cases [sarwar_iot].

- ▶ Smart surface parking
- ▶ Smart signs
- ▶ Smart energy systems
- ▶ Smart lighting
- ▶ Smart water jet systems
- ▶ Smart residuals gathering
- ▶ Smart building construction

Use Case	Packet rate () [packet/day]	Minimum success rate (Ps,min)	Grouping
Wearables	10	90	Group A PL = 10/20B
Smoke Detectors	2	90	
Smart Grid	10	90	
White Goods	3	90	
Waste Management	24	90	
VIP/Pet Tracking	48	90	Group B PL = 50B
Smart Bicycle	192	90	
Animal Tracking	100	90	
Environmental Monitoring	5	90	
Asset Tracking	100	90	
Smart Parking	60	90	
Alarms/Actuators	5	90	Group C PL = 100/200B
Home Automation	5	90	
Machinery Control	100	90	
Water/Gas Metering	8	90	
Environmental Data Collection	24	90	
Medical Assisted Living	8	90	Group D PL = 1KB
Microgeneration	2	90	
Safety Monitoring	2	90	
Propane Tank Monitoring	2	90	
Stationary Monitoring	4	90	
Urban Lighting	5	90	Group E PL = 1000B
Vending Machines Payment	100	90	
Vending Machines General	1	90	

Table 2.3: APPLICATION REQUIREMENTS FOR THE USE CASES OF INTEREST[5] [3].

- ➡ Smart tourism
- ➡ Smart QRinfo
- ➡ Smart monitoring
- ➡ Smart hawkeye

2.1.6 Summary and discussion

2.2 IoT Sensors

2.2.1 OS platform

The operating system is the foundation of the IoT technology as it provides the functions for the connectivity between the nodes. However, different types of nodes need different levels of OS complexity; a passive node generally only needs the communication stack and is not in need of any threading capabilities, as the program can handle all logic in one function. Active nodes and border routers need to have a much more complex OS, as they need to be able to handle several running threads or processes, e.g. routing, data collec-

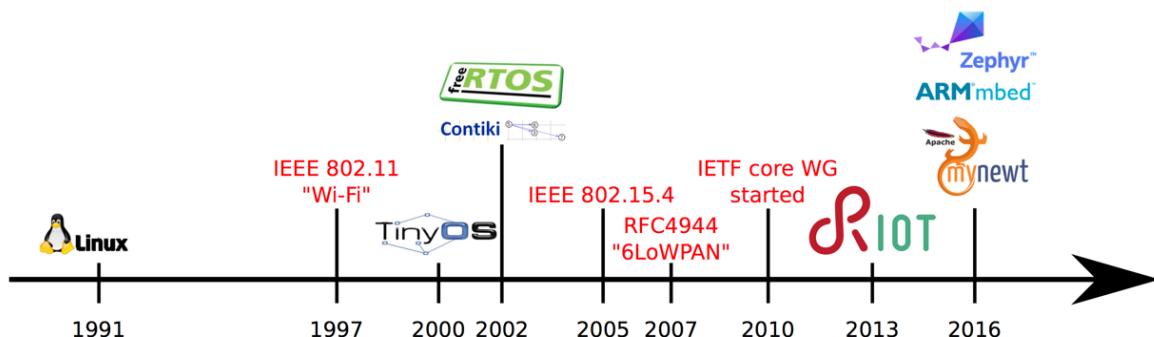


Figure 5: .

tion and interrupts. To qualify as an OS suitable for the IoT, it needs to meet the basic requirements: Low Random-access memory (RAM) footprint Low Read-only memory (ROM) footprint Multi-tasking Power management (PM) Soft real-time These requirements are directly bound to the type of hardware designed for the IoT. As this type of hardware in general needs to have a small form factor and a long battery life, the on-board memory is usually limited to keep down size and energy consumption. Also, because of the limited amount of memory, the implementation of threads is usually a challenging task, as context switching needs to store thread or process variables to memory. The size of the memory also directly affects the energy consumption, as memory in general is very power hungry during accesses. To be able reduce the energy consumption, the OS needs some kind of power management. The power management does not only let the OS turn on and off peripherals such as flash memory, I/O, and sensors, but also puts the MCU itself in different power modes. As the nodes can be used to control and monitor consumer devices, either a hard or soft real-time OS is required. Otherwise, actions requiring a close to instantaneous reaction might be indefinitely delayed. Hard real-time means that the OS scheduler can guarantee latency and execution time, whereas Soft real-time means that latency and execution time is seen as real-time but can not be guaranteed by the scheduler. Operating systems that meet the above requirements are compared in table 2.1 and 2.2.

OS	Architecture	Multi-threading	Scheduling	Dynamic Memory	Memory protection	Network Stack	Virtualization and Completion
Contiki/Contiki-ng	Modular	✓	Interrupts	✓	✗	uIP	Serialized
			execute w.r.t.			Rime	Access
MANTIS	Modular	✗	Priority classes	✓	✗	At Kernel COMM layer	Semaphores.
Nano-RK	Layered	✓	Monotonic harmonized	✗	✗	Socket abstraction	Serialized access semaphores
LiteOS	Monolithic	✓	Round Robin	✓	✓	File	Synchronization primitives

Table 2.4: Common operating systems used in IoT environment [7]

2.2.1.1 Contiki

Contiki is a embedded operating system developed for IoT written in C [12]. It supports a broad range of MCUs and has drivers for various transceivers. The OS does not only support TCP/IPv4 and IPv6 with the uIP stack [9], but also has support for the 6LoWPAN stack and its own stack called RIME. It supports threading with a thread system called Phototreads [13]. The threads are stack-less and thus use only two bytes of memory per thread; however, each thread is bound to one function and it only has permission to control its own execution. Included in Contiki, there is a range of applications such as a HTTP, Constrained Application Protocol (CoAP), FTP, and DHCP servers, as well as other useful programs and tools. These applications can be included in a project and can run simultaneously with the help of Phototreads. The limitations to what applications can be run is the amount of RAM and ROM the target MCU provides. A standard system with IPv6 networking needs about 10 kB RAM and 30 kB ROM but as applications are added the requirements tend to grow.

Contiki is an open source operating system for the Internet of Things. Contiki connects tiny low-cost, low-power micro-controllers to the Internet.

2k RAM, 60k ROM; 10k RAM, 48K ROM Portable to tiny low-power micro-controllers I386 based, ARM, AVR, MSP430, ... Implements uIP stack IPv6 protocol for Wireless Sensor Networks (WSN) Uses the protothreads abstraction to run multiple process in an event based kernel. Emulates concurrency Contiki has an event based kernel (1 stack) Calls a process when an event happens

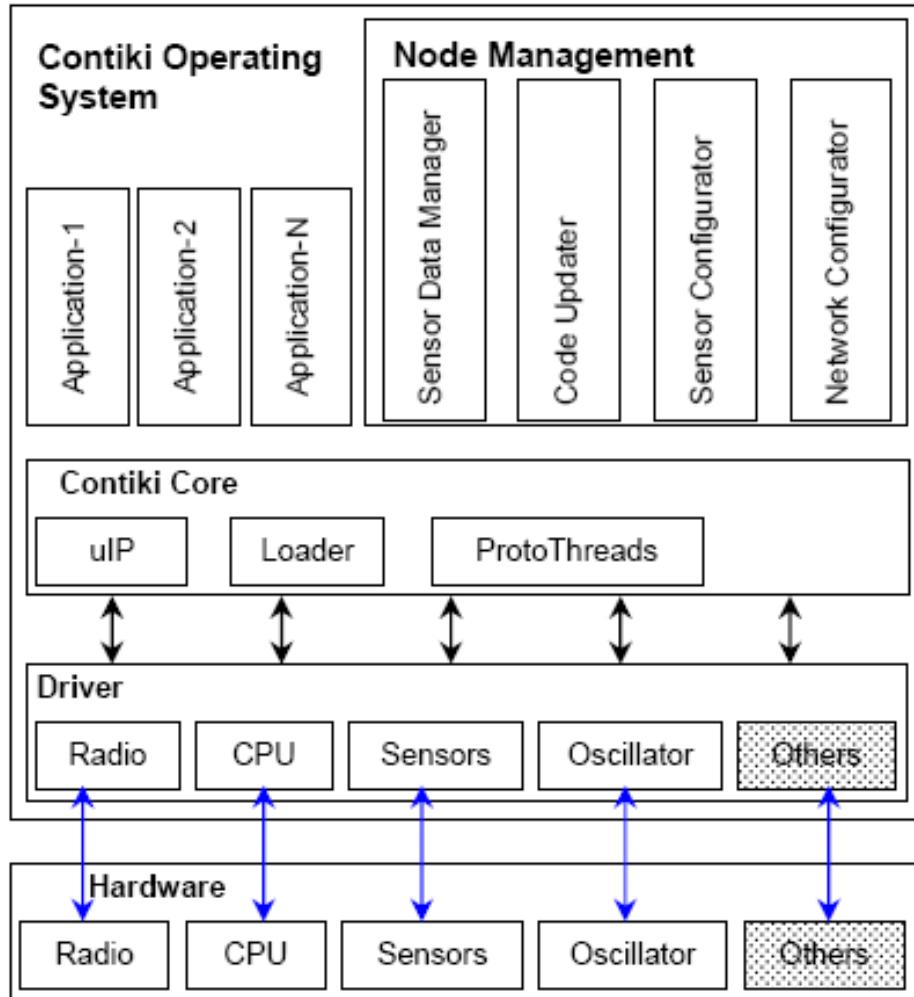


Figure 6: contiki .

Contiki size One of the main aspect of the system, is the modularity of the code. Besides the system core, each program builds only the necessary modules to be able to run, not the entire system image. This way, the memory used from the system, can be reduced to the strictly necessary. This methodology makes more practical any change in any module, if it is needed. The code size of Contiki is larger than that of TinyOS, but smaller than that of the Mantis system. Contiki's event kernel is significantly larger than that of TinyOS because of the different services provided. While the TinyOS event kernel only provides a FIFO event queue scheduler , the Contiki kernel supports both FIFO events and poll handlers with priorities. Furthermore, the flexibility in Contiki requires more run-time code than for a system like TinyOS, where compile time optimization can be done to a larger extent.

The documentation in the doc folder can be compiled, in order to get the html wiki of all the code. It needs doxygen installed, and to run the command make html. This will create a new folder, doc/html, and in the index.html file, the wiki can be opened.

Contiki Hardware Contiki can be run in a number of platforms, each one with a different CPU. Tab.7 shows the hardware platforms currently defined in the Contiki code tree. All these platforms are in the platform folder of the code.

Kernel structure

2.2.1.2 RIOT

RIOT is a open source embedded operating system supported by Freie Universität Berlin, INIRA, and Hamburg University of Applied Sciences [14]. The kernel is written in C but the upper layers support C++ as well. As the project originates from a project with real-time and reliability requirements, the kernel supports hard real-time multi-tasking scheduling. One of the goals of the project is to make the OS completely POSIX compliant. Overhead for multi-threading is minimal with less than 25 bytes per thread. Both IPv6 and 6LoWPAN is supported together with UDP, TCP, and IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL); and CoAP and Concise Binary Object Representation (CBOR) are available as application level communication protocols.

2.2.1.3 TinyOS

TinyOS is written in Network Embedded Systems C (nesC) which is a variant of C [15]. nesC does not have any dynamic memory allocation and all program paths are available at compile-time. This is manageable thanks to the structure of the language; it uses modules and interfaces instead of functions [16]. The modules use and provide interfaces and are interconnected with configurations; this procedure makes up the structure of the program. Multitasking is implemented in two ways: through tasks and events. Tasks, which focus on computation, are non-preemptive, and run until completion. In contrast, events which focus on external events i.e. interrupts, are preemptive, and have separate start and stop functions. The OS has full support for both 6LoWPAN and RPL, and also have libraries for CoAP.

2.2.1.4 freeRTOS

One of the more popular and widely known operating systems is freeRTOS [17]. Written in C with only a few source files, it is a simple but powerful OS, easy to overview and extend. It features two modes of scheduling, pre-emptive and co-operative, which may be selected according to the requirements of the application. Two types of multitasking are featured: one is a lightweight Co-routine type, which has a shared stack for lower RAM usage and is thus aimed to be used on very small devices; the other is simply called Task, has its own stack and can therefore be fully pre-empted. Tasks also support priorities which are used together with the pre-emptive scheduler. The communication methods supported out-of-the-box are TCP and UDP.

Plan de contrôle	Plan de gestion	Plan de données
Contrôle d'admission	Contrôle et supervision de QoS	Contrôle du trafic
Réservation de ressources	Gestion de contrats	Façonnage du trafic
Routage	QoS mapping	Contrôle de congestion
Signalisation	Politique de QoS	Classification de paquets
		Marquage de paquets
		Ordonnancements des paquets
		Gestion de files d'attente

Table 2.5: An example table.

2.2.1.5 SDN platforms

Sensor OpenFlow [20,21] SDWN [60] Smart [14] SDN-WISE [78] SDCSN [88] TinySDN [69,118] Virtual Overlay [59,87,90] Multi-task [122] SDWSN-RL [123] Wireless power transfer [126] Function alternation [65] Statistical machine learning [24] Context-based [91,92] Soft-WSN [9]

- [8] Many studies have identified **SDN** as a potential solution to the WSN challenges, as well as a model for **heterogeneous** integration.
- [8] This **shortfall** can be resolved by using the **SDN approach**.
- [9] **SDN** also enhances better control of **heterogeneous** network infrastructures.
- [9] Anadiotis et al. define a **SDN operating system for IoT** that integrates SDN based WSN (**SDN-WISE**). This experiment shows how **heterogeneity** between different kinds of SDN networks can be achieved.
- [9] In cellular networks, OpenRoads presents an approach of introducing **SDN** based **heterogeneity** in wireless networks for operators.
- [10] There has been a plethora of (industrial) studies **synergising SDN in IoT**. The major characteristics of IoT are low latency, wireless access, mobility and **heterogeneity**.
- [10] Thus a bottom-up approach application of **SDN** to the realisation of **heterogeneous IoT** is suggested.
- [10] Perhaps a more complete IoT architecture is proposed, where the authors apply **SDN** principles in IoT **heterogeneous** networks.
- [11] it provides the **SDWSN** with a proper model of network management, especially considering the potential of **heterogeneity** in SDWSN.
- [11] We conjecture that the **SDN paradigm** is a good candidate to solve the **heterogeneity in IoT**.

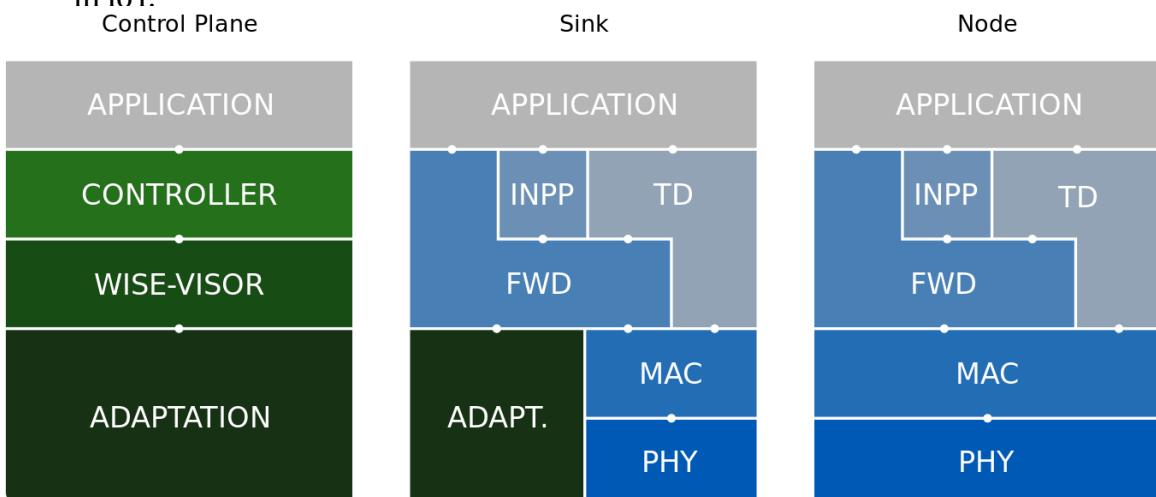


Figure 7: LPWAN connectivity.

Management architecture	Management feature	Controller configuration	Traffic Control	Configuration and monitoring	Scapability and localization	Communication management
[12] Sensor Open Flow	SDN support protocol	Distributed	in/out-band	✓	✓	✓
[13] SDWN	Duty cycling, aggregation, routing	Centralized	in-band	✓		
[14] SDN-WISE	Programming simplicity and aggregation	Distributed	in-band		✓	
[degante_smart_2014] Smart	Efficiency in resource allocation	Distributed	in-band		✓	
SDCSN	Network reliability and QoS	Distributed	in-band		✓	
TinySDN	In-band-traffic control	Distributed	in-band		✓	
Virtual Overlay	Network flexibility	Distributed	in-band		✓	
Context based	Network scalability and performance	Distributed	in-band		✓	
CRLB	Node localization	Centralized	in-band			
Multi-hope	Traffic and energy control	Centralized	in-band			✓
Tiny-SDN	Network task measurement	-	in-band			

Table 2.6: SDN-based network and topology management architectures. [10]

2.2.1.6 Summary and conclusion

	LiteOS	Nano-RK	MANTIS	Contiki
Architecture	Monolithic	Layered	Modular	Modular
Scheduling Memory	Round Robin	Monotonic harmonized	Priority classes	Interrupts execute w.r.t.
Network	File	Socket abstraction	At Kernel COMM layer	uIP, Rime
Virtualization and Completion	Synchronization primitives	Serialized access semaphores	Semaphores	Serialized, Access
Multi threading	✓	✓	✗	✓
Dynamic protection	✓	✗	✓	✓
Memory Stack	✓	✗	✗	✗

Table 2.7: Common operating systems used in IoT environment [7]

2.2.2 Processing Unit

Even though the hardware is in one sense the tool that the OS uses to make IoT possible, it is still very important to select a platform that is future-proof and extensible. To be regarded as an extensible platform, the hardware needs to have I/O connections that can be used by external peripherals. Amongst the candidate interfaces are Serial Peripheral Interface (SPI), Inter-Integrated Circuit (I2C), and Controller Area Network (CAN). These interfaces allow developers to attach custom-made PCBs with sensors for monitoring or actuators for controlling the environment. The best practice is to implement an extension socket with a well-known form factor. A future-proof device is specified as a device that will be as attractive in the future as it is today. For hardware, this is very hard to achieve as there is constant development that follows Moores Law [4]; however, the most important aspects are: the age of the chip, its expected remaining lifetime, and number of current implementations i.e. its popularity. If a device is widely used by consumers, the lifetime of the product is likely to be extended. One last thing to take into consideration is the



Figure 8: .

product family; if the chip belongs to a family with several members the transition to a newer chip is usually easier.

2.2.2.1 OpenMote

OpenMote is based on the Ti CC2538 System on Chip (SoC), which combines an ARM Cortex-M3 with a IEEE 802.15.4 transceiver in one chip [18, 19]. The board follows the XBee form factor for easier extensibility, which is used to connect the core board to either the OpenBattery or OpenBase extension boards [20, 21]. It originates from the CC2538DK which was used by Thingsquare to demo their Mist IoT solution [22]. Hence, the board has full support for Contiki, which is the foundation of Thingsquare. It can run both as a battery-powered sensor board and as a border router, depending on what extension board it is attached to, e.g OpenBattery or OpenBase. Furthermore, the board has limited support but ongoing development for RIOT and also full support for freeRTOS.

2.2.2.2 MSB430-H

The Modular Sensor Board 430-H from Freie Universität Berlin was designed for their ScatterWeb project [23]. As the university also hosts the RIOT project, the decision to support RIOT was natural. The main board has a Ti MSP430F1612 MCU [24], a **Ti CC1100 transceiver**, and a battery slot for dual AA batteries; it also includes a SHT11 temperature and humidity sensor and a MMA7260Q accelerometer to speed up early development. All GPIO pins and buses are connected to external pins for extensibility. Other modules with new peripherals can then be added by making a PCB that matches the external pin layout.

2.2.2.3 Zolertia

As many other Wireless Sensor Network (WSN) products, the Zolertia Z1 builds upon the MSP430 MCU [25, 26]. The communication is managed by the Ti CC2420 which operates in the 2.4 GHz band. The platform includes two sensors: the SHT11 temperature and humidity sensor and the MMA7600Q accelerometer. Extensibility is ensured with: two connections designed especially for external sensors, an external connector with USB, Universal asynchronous receiver/transmitter (**UART**), SPI, and I²C.

2.2.3 Radio Unit

2.2.3.1 Lora Tranceiver

To limit the complexity of the radio unit:

- ➡ limiting message size: maximum application payload size between 51 and 222 bytes, depending on the spreading factor
- ➡ using simple channel codes: Hamming code
- ➡ supporting only half-duplex operation
- ➡ using one transmit-and-receive antenna

~~limiting message size: maximum application payload size between 51 and 222 bytes, depending on the spreading factor using simple channel codes: Hamming code supporting only half-duplex operation using one transmit-and-receive antenna on-chip integrating power amplifier (since transmit power is limited)~~

Ref	Module	Frequency MHz	Tx power	Rx power	Sensitivity	Channels	Distance
[_wasp mote_]	Semtech SX1272	863-870 (EU) 902-928 (US)	14 dBm	dBm	-134 dBm	8 13	22+ km
[_wasp mote_]	rn2483						
[_wasp mote_]	rn2903						
[_wasp mote_]	rak811						
[_wasp mote_]	Semtech sx1276						
[_wasp mote_]	rfm95						
[_wasp mote_]	CMWX1ZZABZ-078						
[_wasp mote_]	LoPy4						
[_wasp mote_]	mDot						
[_wasp mote_]	xDot						
[_wasp mote_]	Laird RM192						
[_wasp mote_]	Laird RM186						
[_wasp mote_]	CMWX1ZZABZ-078						
[_wasp mote_]	Also Laird RM1xx						
[_wasp mote_]	iMST iM88x/iM98x						
[_wasp mote_]	Mic SAM RN34/35						
[_wasp mote_]	Semtech SX1278						

Table 2.8

2.2.4 Sensing Unit

2.2.5 Summary and discussion

2.3 IoT Communication protocols

2.3.1 Application

2.3.1.1 LwM2M

Application protocol	DDS	CoAP	AMQP	MQTT	MQTT-SN	XMPP	HTTP
Service discovery		mDNS				DNS-SD	
Transport				UDP/TCP			
Network		IPv6 RPL			IPv4/IPv6		
MAC		IEEE 802.15.4 2.4GHz, 915, 868MHz	IEEE 802.11 (Wi-Fi) 2.4, 5GHz	RFC 2464 CSMA/CA	IEEE 802.3 (Ethernet) CUTP, FO		RFC 5072 2G, 3G, LTE
		DSS, FSK, OFDM					

Table 2.9: Standardization efforts that support the IoT

2.3.1.2 CBOR

2.3.1.3 DTLS

2.3.1.4 OSCoAP

2.3.1.5 COAP (COnstrained Application Protocol)

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things. More detailed information about the protocol is given in the Contiki OS CoAP section.

1) Overview Like HTTP, CoAP is a document transfer protocol. Unlike HTTP, CoAP is designed for the needs of constrained devices. The packets are much smaller than HTTP TCP flows. Packets are simple to generate and can be parsed in place without consuming extra RAM in constrained devices. CoAP runs over UDP, not TCP. Clients and servers communicate through connectionless datagrams. Retries and reordering are implemented in the application stack. It follows a client/server model. Clients make requests to servers, servers send back responses. Clients may GET, PUT, POST and DELETE resources. CoAP implements the REST model from HTTP, with the primitives GET, POST, PUT and DELETE.

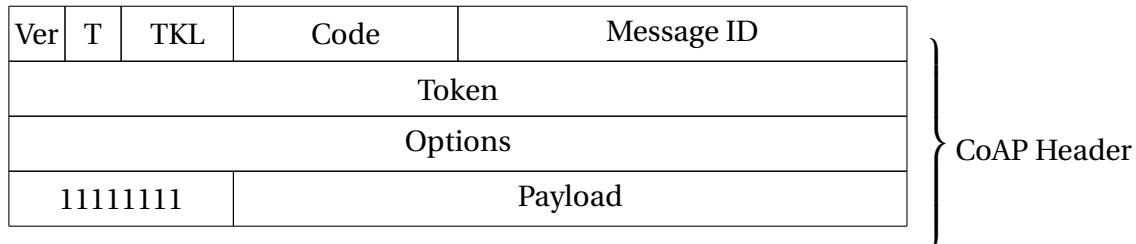
2) Coap Methods CoAP extends the HTTP request model with the ability to observe a resource. When the observe flag is set on a CoAP GET request, the server may continue to reply after the initial document has been transferred. This allows servers to stream state changes to clients as they occur. Either end may cancel the observation. CoAP defines a standard mechanism for resource discovery. Servers provide a list of their resources (along with metadata about them) at /.well-known/core. These links are in the application/link-format media type and allow a client to discover what resources are provided and what media types they are.

3) Coap Transactions

4) Coap Messages The CoAP message structure is designed to be simpler than HTTP, for reduced transmission data. Each field responds to a specific purpose.

- ⇒ Constrained Application Protocol
- ⇒ The IETF Constrained RESTful Environments
- ⇒ CoAP is bound to UDP
- ⇒ CoAP can be divided into two sub-layers
 - ⇒ messaging sub-layer
 - ⇒ request/response sub-layer
 - a) Confirmable.

- b) Non-confirmable.
 - c) Piggybacked responses.
 - d) Separate response
- ⇒ CoAP, as in HTTP, uses methods such as:
- ⇒ GET, PUT, POST and DELETE to
 - ⇒ Achieve, Create, Retrieve, Update and Delete
 - ⇒ Ex: the GET method can be used by a server to inquire the clients temperature



Ver: is the version of CoAP

T: is the type of Transaction

TKL: Token length

Code: represents the request method (1-10) or response code (40-255).

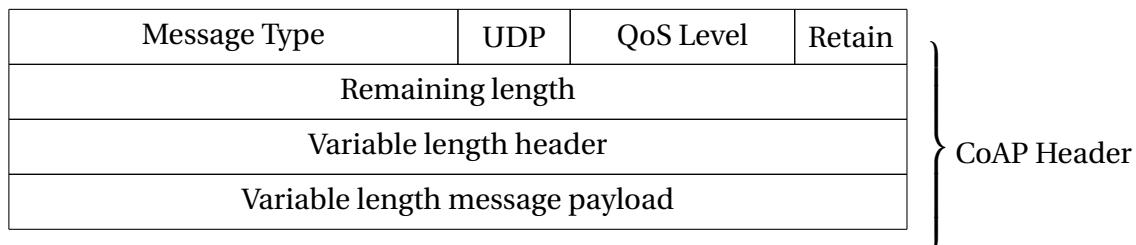
⇒ Ex: the code for GET, POST, PUT, and DELETE is 1, 2, 3, and 4, respectively.

Message ID: is a unique identifier for matching the response.

Token: Optional response matching token.

2.3.1.6 MQTT

- ⇒ Message Queue Telemetry Transport
- ⇒ Andy Stanford-Clark of IBM and Arlen Nipper of Arcom
 - ⇒ Standardized in 2013 at OASIS
- ⇒ MQTT uses the publish/subscribe pattern to provide transition flexibility and simplicity of implementation
- ⇒ MQTT is built on top of the TCP protocol
- ⇒ MQTT delivers messages through three levels of QoS
- ⇒ Specifications
 - ⇒ MQTT v3.1 and MQTT-SN (MQTT-S or V1.2)
 - ⇒ MQTT v3.1 adds broker support for indexing topic names
- ⇒ The publisher acts as a generator of interesting data.



Message type: CONNECT (1), CONNACK (2), PUBLISH (3), SUBSCRIBE (8) and so on

DUP flag: indicates that the message is duplicated

QoS Level: identify the three levels of QoS for delivery assurance of Publish messages

Retain field: retain the last received Publish message and submit it to new subscribers as a first message

2.3.1.7 XMPP

- Extensible Messaging and Presence Protocol
- Developed by the Jabber open source community
- An IETF instant messaging standard used for:
 - ▶ multi-party chatting, voice and telepresence
- Connects a client to a server using a XML stanzas
- An XML stanza is divided into 3 components:
 - ▶ message: fills the subject and body fields
 - ▶ presence: notifies customers of status updates
 - ▶ iq (info/query): pairs message senders and receivers
- Message stanzas identify:
 - ▶ the source (from) and destination (to) addresses
 - ▶ types, and IDs of XMPP entities

2.3.1.8 AMQP

- Advanced Message Queuing Protocol
- Communications are handled by two main components
 - ▶ exchanges: route the messages to appropriate queues.
 - ▶ message queues: Messages can be stored in message queues and then be sent to receivers
- It also supports the publish/subscribe communications.
- It defines a layer of messaging on top of its transport layer.
- AMQP defines two types of messages
 - ▶ bare messages: supplied by the sender
 - ▶ annotated messages: seen at the receiver
- The header in this format conveys the delivery parameters:
 - ▶ durability, priority, time to live, first acquirer & delivery count.
- AMQP frame format
 - Size the frame size.
 - DOFF the position of the body inside the frame.
 - Type the format and purpose of the frame.
 - * Ex: 0x00 show that the frame is an AMQP frame
 - * Ex: 0x01 represents a SASL frame.

2.3.1.9 DDS

- Data Distribution Service
- Developed by Object Management Group (OMG)
- Supports 23 QoS policies:
 - ▶ like security, urgency, priority, durability, reliability, etc
- Relies on a broker-less architecture
 - ▶ uses multicasting to bring excellent Quality of Service
 - ▶ real-time constraints
- DDS architecture defines two layers:
 - DLRL Data-Local Reconstruction Layer
 - * serves as the interface to the DCPS functionalities
 - DCPS Data-Centric Publish/Subscribe
 - * delivering the information to the subscribers

Application protocol	Rest-Full	Transport	Publish/Subscribe	Request/Response	Security	QoS	Header size (Byte)
COAP	✓	UDP	✓	✓	DTLS	✓	4
MQTT	✗	TCP	✓	✗	SSL	✓	2
MQTT-SN	✗	TCP	✓	✗	SSL	✓	2
XMPPTCPS	✗	TCP	✓	✓	SSL	✗	-
AMQP	✗	TCP	✓	✗	SSL	✓	8
DDS	✗	UDP TCP	✓	✗	SSL DTLS	✓	-
HTTP	✓	TCP	✗	✓	SSL	✗	-

Table 2.10: Application protocols comparison

- ➡ 5 entities are involved with the data flow in the DCPS layer:
 - ➡ Publisher: disseminates data
 - ➡ DataWriter: used by app to interact with the publisher
 - ➡ Subscriber: receives published data and delivers them to app
 - ➡ DataReader: employed by Subscriber to access received data
 - ➡ Topic: relate DataWriters to DataReaders
- ➡ No need for manual reconfiguration or extra administration
- ➡ It is able to run without infrastructure
- ➡ It is able to continue working if failure happens.
- ➡ It inquires names by sending an IP multicast message to all the nodes in the local domain
 - ➡ Clients ask devices that have the given name to reply back
 - ➡ the target machine receives its name and multicasts its IP @
 - ➡ Devices update their cache with the given name and IP @

2.3.1.10 mDNS

- ➡ Requires zero configuration aids to connect machine
- ➡ It uses mDNS to send DNS packets to specific multicast addresses through UDP
- ➡ There are two main steps to process Service Discovery:
 - ➡ finding host names of required services such as printers
 - ➡ pairing IP addresses with their host names using mDNS
- ➡ Advantages
 - ➡ IoT needs an architecture without dependency on a configuration mechanism
 - ➡ smart devices can join the platform or leave it without affecting the behavior of the whole system
- ➡ Drawbacks
 - ➡ Need for caching DNS entries

2.3.2 Network

2.3.2.1 6TiSCH

2.3.2.2 OLSRv2

2.3.2.3 AODVv2

2.3.2.4 LoRaWAN

2.3.2.5 ROHC

2.3.2.6 IPHC

2.3.2.7 SCHC

2.3.2.8 NHC

2.3.2.9 ROLL

2.3.2.10 RPL

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.

An RPL Instance consists of multiple Destination Oriented Directed Acyclic Graphs (DODAGs). Traffic moves either up towards the DODAG root or down towards the DODAG leafs. The graph building process starts at the root or LBR (LowPAN Border Router). There could be multiple roots configured in the system. The RPL routing protocol specifies a set of ICMPv6 control messages to exchange graph related information. These messages are called DIS (DODAG Information Solicitation), DIO (DODAG Information Object) and DAO (DODAG Destination Advertisement Object). The root starts advertising the information about the graph using the DIO message. The nodes in the listening vicinity (neighbouring nodes) of the root will receive and process DIO messages potentially from multiple nodes and makes a decision based on certain rules (according to the objective function, DAG characteristics, advertised path cost and potentially local policy) whether to join the graph or not. Once the node has joined a graph it has a route toward the graph (DODAG) root. The graph root is termed as the parent of the node. The node computes the rank of itself within the graph, which indicates the coordinates of the node in the graph hierarchy. If configured to act as a router, it starts advertising the graph information with the new information to its neighbouring peers. If the node is a leaf node, it simply joins the graph and does not send any DIO message. The neighbouring peers will repeat this process and do parent selection, route addition and graph information advertisement using DIO messages. This rippling effect builds the graph edges out from the root to the leaf nodes where the process terminates. In this formation each node of the graph has a routing entry towards its parent (or multiple parents depending on the objective function) in a hop-by-hop fashion and the leaf nodes can send a data packet all the way to root of the graph by just forwarding the packet to its immediate parent. This model represents a MP2P (Multipoint-to-point) forwarding model where each node of the graph has reachability toward the graph root. This is also referred to as UPWARD routing. Each node in the graph has a rank that is relative and represents an increasing coordinate of the relative position of the node with respect to the root in graph topology. The notion of rank is used by RPL for various purposes including loop avoidance. The MP2P flow of traffic is called the up direction in the DODAG.

The DIS message is used by the nodes to proactively solicit graph information (via DIO) from the neighbouring nodes should it become active in a stable graph environment using the poll or pull model of retrieving graph information or in other conditions. Similar to MP2P or up direction of traffic, which flows from the leaf towards the root there is a need for traffic to flow in the opposite or down direction. This traffic may originate from outside the LLN network, at the root or at any intermediate nodes and destined to a (leaf) node. This requires a routing state to be built at every node and a mechanism to populate these routes. This is accomplished by the DAO (Destination Advertisement Object) message. DAO messages are used to advertise prefix reachability towards the leaf nodes in support of the down traffic. These messages carry prefix information, valid life-

time and other information about the distance of the prefix. As each node joins the graph it will send DAO message to its parent set. Alternately, a node or root can poll the sub-dag for DAO message through an indication in the DIO message. As each node receives the DAO message, it processes the prefix information and adds a routing entry in the routing table. It optionally aggregates the prefix information received from various nodes in the subdag and sends a DAO message to its parent set. This process continues until the prefix information reaches the root and a complete path to the prefix is setup. Note that this mode is called the storing mode of operation where intermediate nodes have available memory to store routing tables. RPL also supports another mode called non-storing mode where intermediate node do not store any routes.

2.3.2.11 6LowPAN

6LoWPAN is a networking technology or adaptation layer that allows IPv6 packets to be carried efficiently within a small link layer frame, over IEEE 802.15.4 based networks. As the full name implies, IPv6 over Low-Power Wireless Personal Area Networks, it is a protocol for connecting wireless low power networks using IPv6.

As the full name implies, IPv6 over Low-Power Wireless Personal Area Networks, it is a protocol for connecting wireless low power networks using IPv6.

1) Characteristics

- ▶ Compression of IPv6 and UDP/ICMP headers
- ▶ Fragmentation / reassembly of IPv6 packets
- ▶ Mesh addressing
- ▶ Stateless auto configuration
- ▶

2) Encapsulation Header format All LowPAN encapsulated datagrams are prefixed by an encapsulation header stack. Each header in the stack starts with a header type field followed by zero or more header fields.

3) Fragment Header The fragment header is used when the payload is too large to fit in a single IEEE 802.15.4 frame. The Fragment header is analogous to the IEEE 1394 Fragment header and includes three fields: Datagram Size, Datagram Tag, and Datagram Offset. Datagram Size identifies the total size of the unfragmented payload and is included with every fragment to simplify buffer allocation at the receiver when fragments arrive out-oforder. Datagram Tag identifies the set of fragments that correspond to a given payload and is used to match up fragments of the same payload. Datagram Offset identifies the fragments offset within the unfragmented payload and is in units of 8-byte chunks.

4) Mesh addressing header The Mesh Addressing header is used to forward 6LoWPAN payloads over multiple radio hops and support layer-two forwarding. The mesh addressing header includes three fields: Hop Limit, Source Address, and Destination Address. The Hop Limit field is analogous to the IPv6 Hop Limit and limits the number of hops for forwarding. The Hop Limit field is decremented by each forwarding node, and if decremented to zero the frame is dropped. The source and destination addresses indicate the end-points of an IP hop. Both addresses are IEEE 802.15.4 link addresses and may carry either a short or extended address.

5) Header compression (RFC4944) RFC 4944 defines HC1, a stateless compression scheme optimized for link-local IPv6 communication. HC1 is identified by an encoding byte following the Compressed IPv6 dispatch header, and it operates on fields in the upper-layer headers. 6LoWPAN elides some fields by assuming commonly used values. For example, it compresses the 64-bit network prefix for both source and destination addresses to a single bit each when they carry the well-known link-local prefix. 6LoWPAN compresses the Next Header field to two bits whenever the packet uses UDP, TCP, or ICMPv6. Furthermore, 6LoWPAN compresses Traffic Class and Flow Label to a single bit when their values are both zero. Each compressed form has reserved values that indicate that the fields are carried inline for use when they don't match the elided case. 6LoWPAN elides other fields by exploiting cross-layer redundancy. It can derive Payload Length which is always elided from the 802.15.4 frame or 6LoWPAN fragmentation header. The 64-bit interface identifier (IID) for both source and destination addresses are elided if the destination can derive them from the corresponding link-layer address in the 802.15.4 or mesh addressing header. Finally, 6LoWPAN always elides Version by communicating via IPv6.

The HC1 encoding is shown in Figure 11. The first byte is the dispatch byte and indicates the use of HC1. Following the dispatch byte are 8 bits that identify how the IPv6 fields are compressed. For each address, one bit is used to indicate if the IPv6 prefix is linklocal and elided and one bit is used to indicate if the IID can be derived from the IEEE 802.15.4 link address. The TF bit indicates whether Traffic Class and Flow Label are both zero and elided. The two Next Header bits indicate if the IPv6 Next Header value is 7UDP, TCP, or ICMP and compressed or carried inline. The HC2 bit indicates if the next header is compressed using HC2. Fully compressed, the HC1 encoding reduces the IPv6 header to three bytes, including the dispatch header. Hops Left is the only field always carried inline.

RFC 4944 uses stateless compression techniques to reduce the overhead of UDP headers. When the HC2 bit is set in the HC1 encoding, an additional 8-bits is included immediately following the HC1 encoding bits that specify how the UDP header is compressed. To effectively compress UDP ports, 6LoWPAN introduces a range of well-known ports (61616 - 61631). When ports fall in the well-known range, the upper 12 bits may be elided. If both ports fall within range, both Source and Destination ports are compressed down to a single byte. HC2 also allows elision of the UDP Length, as it can be derived from the IPv6 Payload Length field.

The best-case compression efficiency occurs with link-local unicast communication, where HC1 and HC2 can compress a UDP/IPv6 header down to 7 bytes. The Version, Traffic Class, Flow Label, Payload Length, Next Header, and linklocal prefixes for the IPv6 Source and Destination addresses are all elided. The suffix for both IPv6 source and destination addresses are derived from the IEEE 802.15.4 header.

However, RFC 4944 does not efficiently compress headers when communicating outside of link-local scope or when using multicast. Any prefix other than the linklocal prefix must be carried inline. Any suffix must be at least 64 bits when carried inline even if derived from a short 802.15.4 address. As shown in Figure 8, HC1/HC2 can compress a link-local multicast UDP/IPv6 header down to 23 bytes in the best case. When communicating with nodes outside the LoWPAN, the IPv6 Source Address prefix and full IPv6 Destination Address must be carried inline.

6) Header compression Improved (draft-hui-6lowpan-hc-01) To provide better compression over a broader range of scenarios, the 6LoWPAN working group is standardizing

an improved header compression encoding format, called HC. The format defines a new encoding for compressing IPv6 header, called IPHC. The new format allows Traffic Class and Flow Label to be individually compressed, Hop Limit compression when common values (E.g., 1 or 255) are used, makes use of shared-context to elide the prefix from IPv6 addresses, and supports multicast addresses most often used for IPv6 ND and SLAAC. Contexts act as shared state for all nodes within the LoWPAN. A single context holds a single prefix. IPHC identifies the context using a 4-bit index, allowing IPHC to support up to 16 contexts simultaneously within the LoWPAN. When an IPv6 address matches a contexts stored prefix, IPHC compresses the prefix to the contexts 4-bit identifier. Note that contexts are not limited to prefixes assigned to the LoWPAN but can contain any arbitrary prefix. As a result, share contexts can be configured such that LoWPAN nodes can compress the prefix in both Source and Destination addresses even when communicating with nodes outside the LoWPAN.

The improved header compression encoding is shown in Figure 8. The first three bits (011) form the header type and indicate the use of IPHC. The TF bits indicate whether the Traffic Class and/or Flow Label fields are compressed. The HLIM bits indicate whether the Hop Limit takes the value 1 or 255 and compressed, or carried inline.

Bits 8-15 of the IPHC encoding indicate the compression methods used for the IPv6 Source and Destination Addresses. When the Context Identifier (CID) bit is zero, the default context may be used to compress Source and/or Destination Addresses. This mode is typically when both Source and Destination Addresses are assigned to nodes in the same LoWPAN. When the CID bit is one, two additional 4-bit fields follow the IPHC encoding to indicate which one of 16 contexts is in use for the source and destination addresses. The Source Address Compression (SAC) indicates whether stateless compression is used (typically for link-local communication) or stateful context-based compression is used (typically for global communication). The Source Address Mode (SAM) indicates whether the full Source Address is carried inline, upper 16 or 64-bits are elided, or the full Source Address is elided. When SAC is set and the Source Addresses prefix is elided, the identified context is used to restore those bits. The Multicast (M) field indicates whether the Destination Address is a unicast or multicast address. When the Destination Address is a unicast address, the DAC and DAM bits are analogous to the SAC and SAM bits. When the Destination Address is a multicast address, the DAM bits indicate different forms of multi-cast compression. HC also defines a new framework for compressing arbitrary next headers, called NHC. HC2 in RFC 4944 is only capable of compressing UDP, TCP, and ICMPv6 headers, the latter two are not yet defined. Instead, the NHC header defines a new variable length Next Header identifier, allowing for future definition of arbitrary next header compression encodings. HC initially defines a compression encoding for UDP headers, similar to that defined in RFC 4944. Like RFC 4944, HC utilizes the same well-known port range (61616-61631) to effectively compress UDP ports down to 4-bits each in the best case. However, HC no longer provides an option to carry the Payload Length in line, as it can always be derived from the IPv6 header. Finally, HC allows elision of the UDP Checksum whenever an 10upper layer message integrity check covers the same information and has at least the same strength. Such a scenario is typical when transportor application-layer security is used. As a result, the UDP header can be compressed down to two bytes in the best case.

- ➡ Routing over low-power and lossy links (ROLL)
- ➡ Support minimal routing requirements.
 - ➡ like multipoint-to-point, point-to-multipoint and point-to-point.
- ➡ A Destination Oriented Directed Acyclic Graph (DODAG)

Routing protocol	Control Cost	Link Cost	Node Cost
OSPF/IS-IS	✗	✓	✗
OLSRv2	?	✓	✓
RIP	✓	?	✗
DSR	✓	✗	✗
RPL	✓	✓	✓

Table 2.11: Routing protocols comparison [[_rpl2_](#)]

- ⇒ Directed acyclic graph with a single root.
- ⇒ Each node is aware of its parents
- ⇒ but not about related children
- ⇒ RPL uses four types of control messages
 - ⇒ DODAG Information Object (DIO)
 - ⇒ Destination Advertisement Object (DAO)
 - ⇒ DODAG Information Solicitation (DIS)
 - ⇒ DAO Acknowledgment (DAO-Ack)
- ⇒ Standard topologies to form IEEE 802.15.4e networks are
 - Star contains at least one FFD and some RFDs
 - Mesh contains a PAN coordinator and other nodes communicate with each other
 - Cluster consists of a PAN coordinator, a cluster head and normal nodes.
- ⇒ The IEEE 802.15.4e standard supports 2 types of network nodes
 - FFD Full function device: serve as a coordinator
 - * It is responsible for creation, control and maintenance of the net
 - * It stores a routing table in their memory and implement a full MAC
 - RFD Reduced function devices: simple nodes with restricted resources
 - * They can only communicate with a coordinator

Preamble	PHY layer header	MAC header	Medium Access Control	MAC payload	FPort	Payload	MIC	CRC
----------	------------------	------------	-----------------------	-------------	-------	---------	-----	-----

Routing protocol	Control Cost	Link Cost	Node Cost
OSPF/IS-IS	✗	✓	✗
OLSRv2	?	✓	✓
RIP	✓	?	✗
DSR	✓	✗	✗
RPL	✓	✓	✓

Table 2.12: Routing protocols comparison [[_rpl2_](#)]

2.3.3 MAC

Channel based	FDMA	OFDMA WDMA SC-FDMA	
	TDMA	MF-TDMA STDMA	
	CDMA	W-CDMA TD-CDMA TD-SCDMA DS-CDMA FH-CDMA MC-CDMA	
	SDMA	HC-SDMA	
Packet-based	Collision recovery	ALOHA Slotted ALOHA R-ALOHA AX.25 CSMA/CD	
	Collision avoidance	MACA MACAW CSMA CSMA/CA DCF PCF HCF CSMA/CARP	
	Collision-free	Token ring Token bus MS-ALOHA	
Duplexing methods	Delay and disruption tolerant	MANET VANET DTN Dynamic Source Routing	

Table 2.13

2.3.3.1 Sharing the channel

1) TDMA, FDMA, CDMA, TSMA

2.3.3.2 Transmitting information

1) TFDM, TDSSS, TFHSS

2.3.4 Radio

2.3.4.1 Digital modulation

1) ASK, APSK, CPM, FSK, MFSK, MSK, OOK, PPM, PSK, QAM, SC-FDE, TCM WDM

2.3.4.2 Hierarchical modulation

1) QAM, WDM

2.3.4.3 Spread spectrum

1) SS, DSSS, FHSS, THSS

2.3.5 Summary and discussion

2.4 IoT Norms & Standards

Several different wireless communication protocols, such as Wireless LAN (WLAN), BLE, 6LoWPAN, and ZigBee may be suitable for IoT applications. They all operate in the 2.4GHz frequency band and this, together with the limited output power in this band, means that they all have a similar range. The main differences are located in the MAC, PHY, and network layer. WLAN is much too power hungry as seen in table 2.6 and is only listed as a reference for the comparisons.

2.4.1 Divers

2.4.1.1 IPLC

2.4.1.2 BACnet

2.4.1.3 Z-WAze

2.4.1.4 Bluetooth LE

BLE is developed to be backwards compatible with Bluetooth, but with lower data rate and power consumption [28]. Featuring a data rate of 1Mbit/s with a peak current consumption less than 15mA, it is a very efficient protocol for small amounts of data. Each frame can be transmitted 47bytes in 1Mbit/s = 376Mus; thanks to the short transmission time, the transceivers consumes less power as the transceiver can be in receive mode or completely off most of the time. BLE uses its own addressing methods and as the MAC frame size (figure 2.6) is only 39bytes, thus IPv6 addressing is not possible.

Starting from Bluetooth version 4.2, there is support for IPv6 addressing with the Internet Protocol Support Profile; the new version allows the BLE frame to be variable between

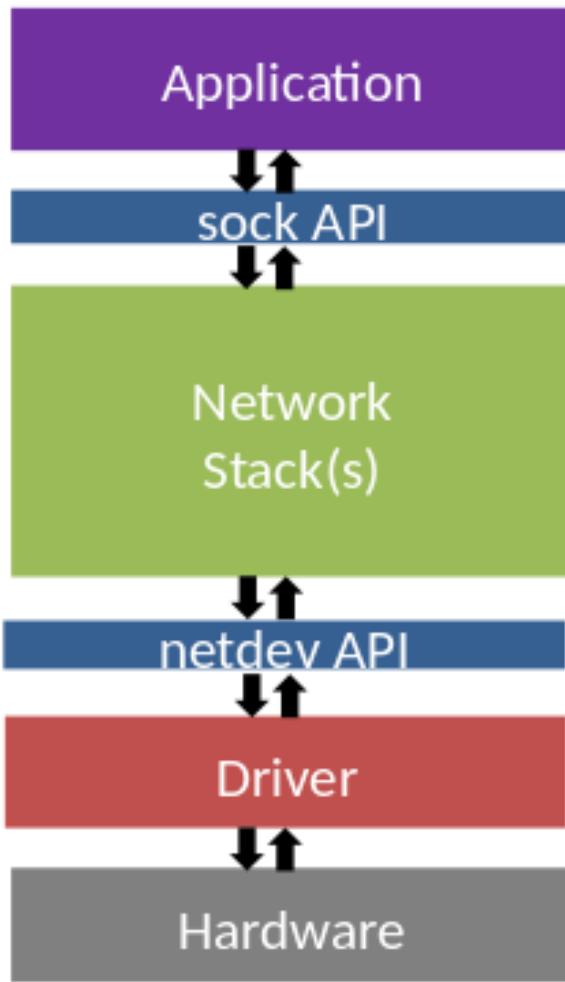


Figure 9: LPWAN.

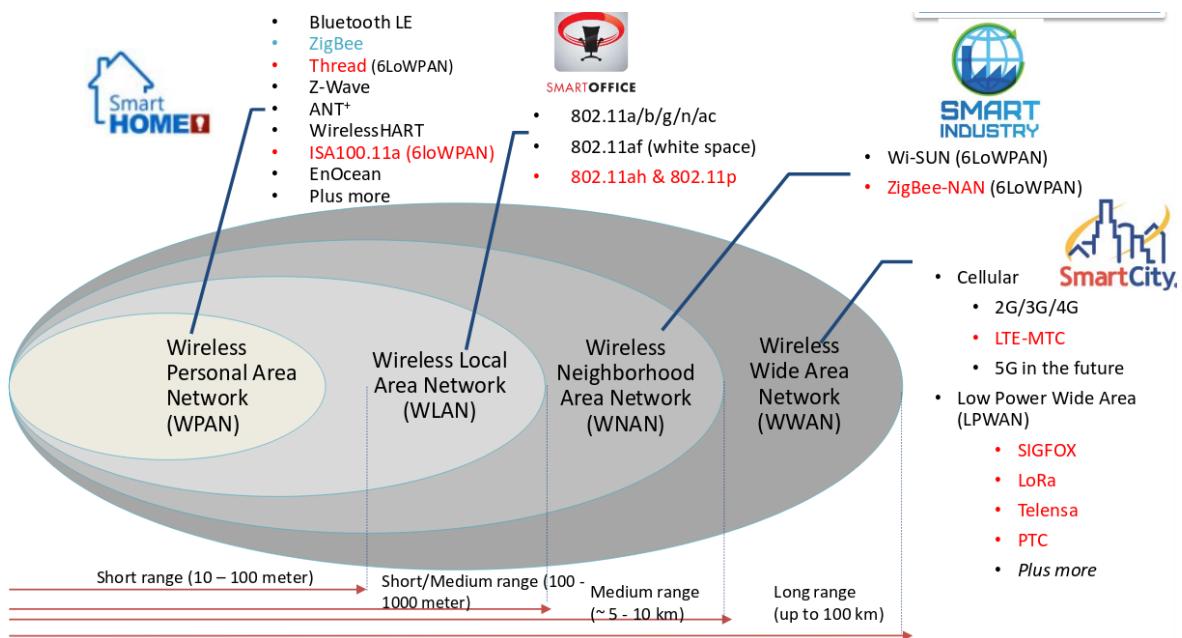


Figure 10: LPWAN.

2257 bytes. The network set-up is controlled by the standard Bluetooth methods, whereas

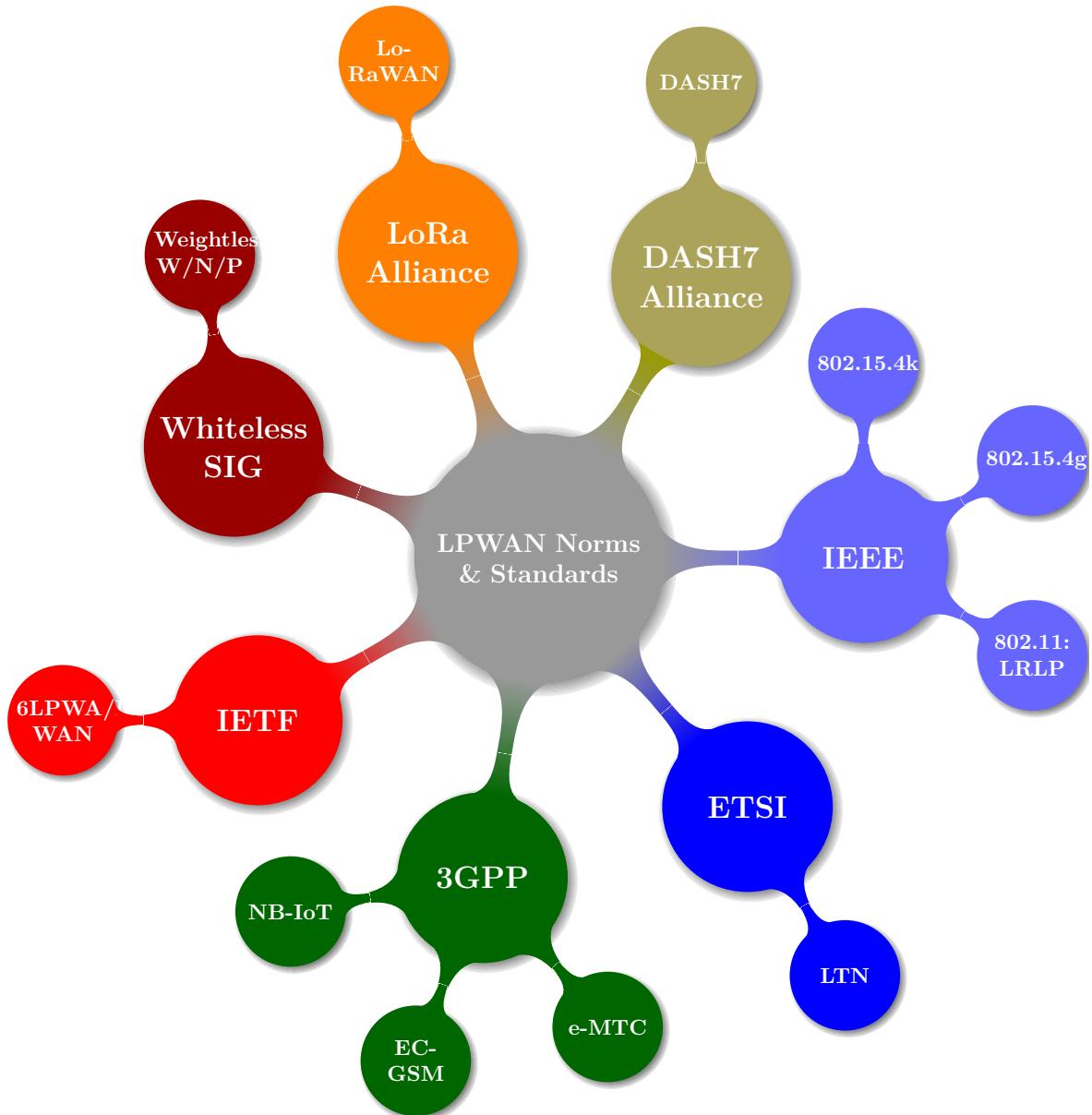


Figure 11: LPWAN.

IPv6 addressing is handled by 6LoWPAN as specified in IPv6 over Bluetooth Low Energy [29].

2.4.2 SigFox

2.4.3 IETF

2.4.3.1 6LoWPAN

is a relatively new protocol that is maintained by the Internet Engineering Task Force (IETF) [7, 6]. The purpose of the protocol is to enable IPv6 traffic over a IEEE 802.15.4 network with as low overhead as possible; this is achieved by compressing the IPv6 and UDP header. A full size IPv6 + UDP header is 40+8 bytes which is tild 38% of a IEEE 802.15.4 frame, but with the header compression this overhead can be reduced to 7 bytes, thus reducing the overhead to tild 5%, as seen in figures 2.3 and 2.4.

2.4.4 3GPP

2.4.4.1 NB-IoT

2.4.4.2 EC-GSM

2.4.4.3 e-MTC

2.4.5 IEEE

2.4.5.1 IEEE 802.11

2.4.5.2 IEEE 802.15.4

At present days, there are several technology standards. Each one is designed for a specific need in the market. For the Wireless Sensor Networks, the aim is to transmit little information, in a small range, with a small power consumption and low cost. The IEEE 802.15.4 standard offers physical and media access control layers for low-cost, low-speed, low-power Wireless Personal Area Networks (WPANs)

1) Physical Layer The standard operates in 3 different frequency bands: - 16 channels in the 2.4GHz ISM band - 10 channels in the 915MHz ISM band - 1 channel in the European 868MHz band

2) Definitions Coordinator: A device that provides synchronization services through the transmission of beacons. PAN Coordinator: The central coordinator of the PAN. This device identifies its own network as well as its configurations. There is only one PAN Coordinator for each network. Full Function Device (FFD): A device that implements the complete protocol set, PAN coordinator capable , talks to any other device. This type of device is suitable for any topology. Reduced Function Device (RFD): A device with a reduced implementation of the protocol, cannot become a PAN Coordinator. This device is limited to leafs in some topologies.

3) Topologies Star topology: All nodes communicate via the central PAN coordinator , the leafs may be any combination of FFD and RFD devices. The PAN coordinator usually uses main power.

Peer to peer topology: Nodes can communicate via the central PAN coordinator and via additional point-to-point links . All devices are FFD to be able to communicate with each other.

Combined Topology: Star topology combined with peer-to-peer topology. Leafs connect to a network via coordinators (FFDs) . One of the coordinators serves as the PAN coordinator .

IEEE 802.15.4

The IEEE 802.15.4 standard defines the PHY and MAC layers for wireless communication [6]. It is designed to use as little transmission time as possible but still have a decent payload, while consuming as little power as possible. Each frame starts with a preamble and a start frame delimiter; it then continues with the MAC frame length and the MAC frame itself as seen in figure 2.2. The overhead for each PHY packet is only 4+1+1 133 tild 4.5%; when using the maximum transmission speed of 250kbit/s, each frame can be sent 133byte in 250kbit/s = 4.265ms. Furthermore, it can also operate in the 868MHz and

915MHz bands, maintaining the 250kbit/s transmission rate by using Offset quadrature phase-shift keying (O-QPSK).

Several network layer protocols are implemented on top of IEEE 802.15.4. The two that will be examined are 6LoWPAN and ZigBEE.

2.4.5.3 ZigBee

is a communication standard initially developed for home automation networks; it has several different protocols designed for specific areas such as lighting, remote control, or health care [27, 6]. Each of these protocols uses their own addressing with different overhead; however, there is also the possibility of direct IPv6 addressing. Then, the overhead is the same as for uncompressed 6LoWPAN, as seen in figure 2.5.

A new standard called ZigBee 3.0 aims to bring all these standards together under one roof to simplify the integration into IoT. The release date of this standard is set to Q4 2015.

2.4.6 LoRaWAN

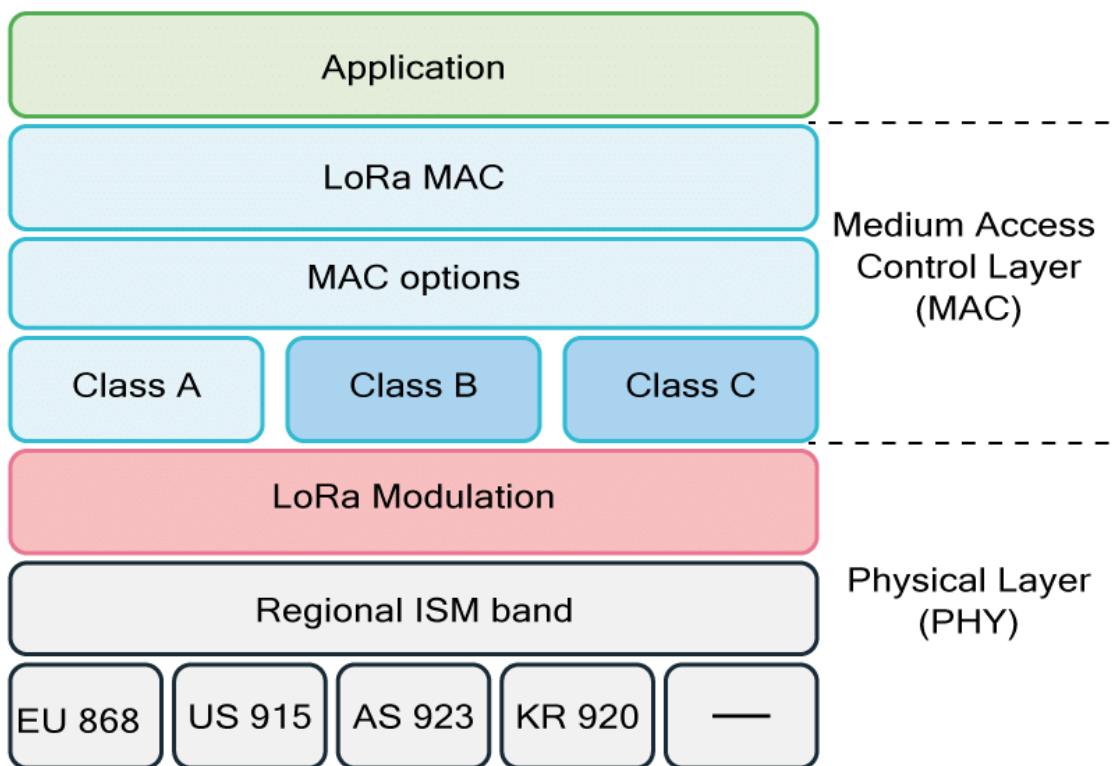


Figure 12: uhuhuh.

LoRa (Long Range) is a proprietary spread spectrum modulation technique by Semtech. It is a derivative of Chirp Spread Spectrum (CSS). The LoRa physical layer may be used with any MAC layer; however, LoRaWAN is the currently proposed MAC which operates a network in a simple star topology.

As LoRa is capable to transmit over very long distances it was decided that LoRaWAN only needs to support a star topology. Nodes transmit directly to a gateway which is powered and connected to a backbone infrastructure. Gateways are powerful devices with powerful radios capable to receive and decode multiple concurrent transmissions (up to

50). Three classes of node devices are defined: (1) Class A enddevices: The node transmits to the gateway when needed. After transmission the node opens a receive window to obtain queued messages from the gateway. (2) Class B enddevices with scheduled receive slots: The node behaves like a Class A node with additional receive windows at scheduled times. Gateway beacons are used for time synchronisation of end-devices. (3) Class C end-devices with maximal receive slots: these nodes are continuous listening which makes them unsuitable for battery powered operations.

2.4.6.1 ALIANCE

1) Class-A

1.1) **Uplink** LoRa Server supports Class-A devices. In Class-A a device is always in sleep mode, unless it has something to transmit. Only after an uplink transmission by the device, LoRa Server is able to schedule a downlink transmission. Received frames are de-duplicated (in case it has been received by multiple gateways), after which the mac-layer is handled by LoRa Server and the encrypted application-playload is forwarded to the application server.

1.2) **Downlink** LoRa Server persists a downlink device-queue for to which the application-server can enqueue downlink payloads. Once a receive window occurs, LoRa Server will transmit the first downlink payload to the device.

1.3) **Confirmed data** LoRa Server sends an acknowledgement to the application-server as soon one is received from the device. When the next uplink transmission does not contain an acknowledgement, a nACK is sent to the application-server.

Note: After a device (re)activation the device-queue is flushed.

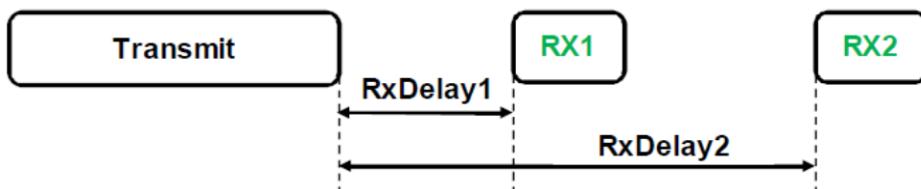


Figure 13: Class A.

2) **Class-B** LoRa Server supports Class-B devices. A Class-B device synchronizes its internal clock using Class-B beacons emitted by the gateway, this process is also called a beacon lock. Once in the state of a beacon lock, the device negotiates its ping-interval. LoRa Server is then able to schedule downlink transmissions on each occurring ping-interval.

2.1) **Downlink** LoRa Server persists all downlink payloads in its device-queue. When the device has acquired a beacon lock, it will schedule the payload for the next free ping-slot in the queue. When adding payloads to the queue when a beacon lock has not yet been acquired, LoRa Server will update all device-queue to be scheduled on the next free ping-slot once the device has acquired the beacon lock.

2.2) Confirmed data LoRa Server sends an acknowledgement to the application-server as soon one is received from the device. Until the frame has timed out, LoRa Server will wait with the transmission of the next downlink Class-B payload.

Note: The timeout of a confirmed Class-B downlink can be configured through the device-profile. This should be set to a value less than the interval between two ping-slots.

2.3) Requirements

Device The device must be able to operate in Class-B mode. This feature has been tested against the develop branch of the Semtech LoRaMac-node source.

Gateway The gateway must have a GNSS based time-source and must use at least the Semtech packet-forwarder version 4.0.1 or higher. It also requires LoRa Gateway Bridge 2.2.0 or higher.

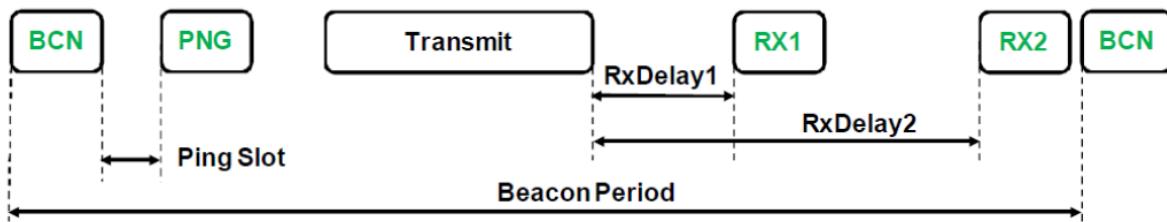


Figure 14: Class B.

3) Class-C

3.1) Downlink LoRa Server supports Class-C devices and uses the same Class-A downlink device-queue for Class-C downlink transmissions. The application-server can enqueue one or multiple downlink payloads and LoRa Server will transmit these (semi) immediately to the device, making sure no overlap exists in case of multiple Class-C transmissions.

3.2) Confirmed data LoRa Server sends an acknowledgement to the application-server as soon one is received from the device. Until the frame has timed out, LoRa Server will wait with the transmission of the next downlink Class-C payload.

Note: The timeout of a confirmed Class-C downlink can be configured through the device-profile.

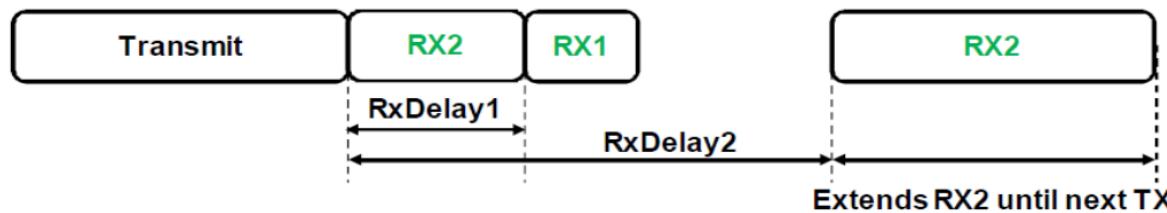


Figure 15: Class C.

2.4.6.2 SEMTECH

LoRa has four configurable parameters:

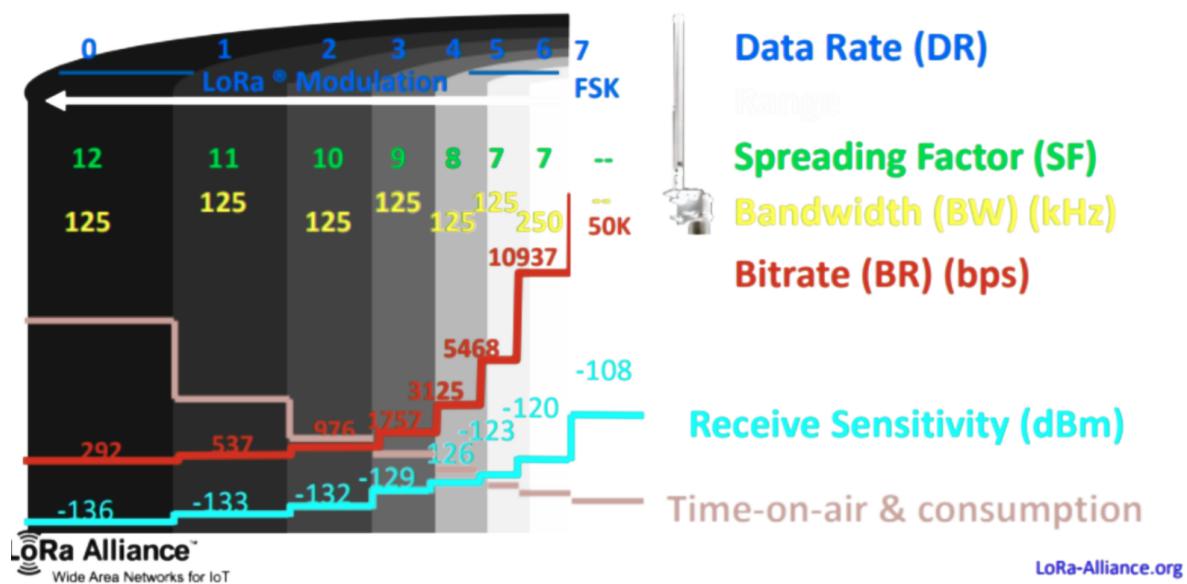


Figure 16: LoraWan Parameters.

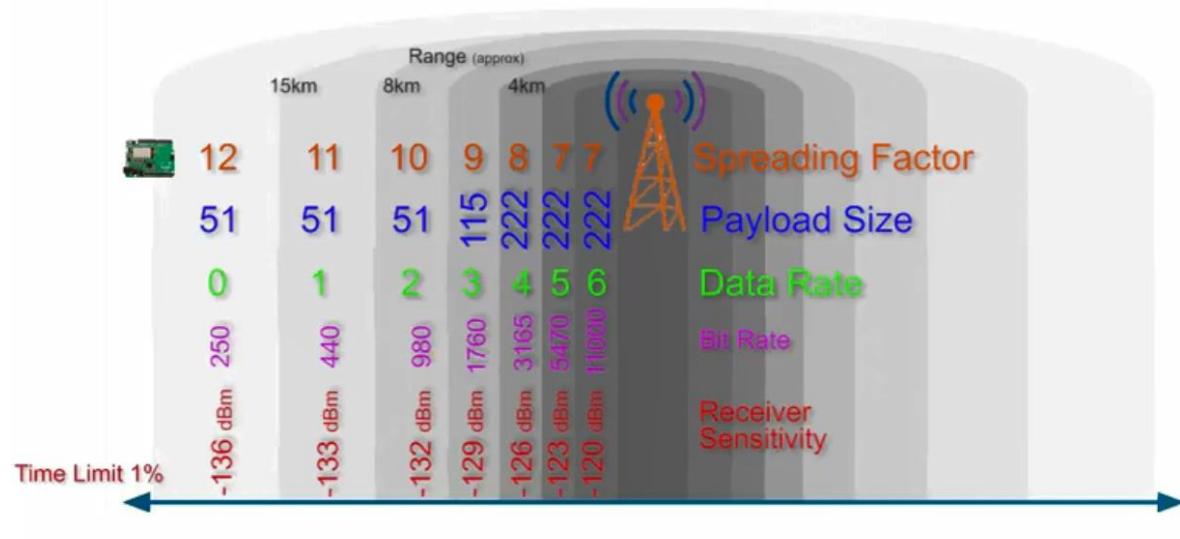


Figure 17: .

BW Bandwidth: Bandwidth (BW) is the range of frequencies in the transmission band. Higher BW gives a higher data rate (thus shorter time on air), but a lower sensitivity (due to integration of additional noise). A lower BW gives a higher sensitivity, but a lower data rate. Lower BW also requires more accurate crystals (less ppm). Data is sent out at a chip rate equal to the bandwidth. So, a bandwidth of 125 kHz corresponds to a chip rate of 125 kcps. The SX1272 has three programmable bandwidth settings: 500 kHz, 250 kHz and 125 kHz. The Semtech SX1272 can be programmed in the range of 7.8 kHz to 500 kHz, though bandwidths lower than 62.5 kHz require a temperature compensated crystal oscillator (TCXO).

CF Carrier Frequency: Carrier Frequency (CF) is the centre frequency used for the transmission band. For the SX1272 it is in the range of 860 MHz to 1020 MHz, programmable in steps of 61 Hz. The alternative radio chip Semtech SX1276 allows

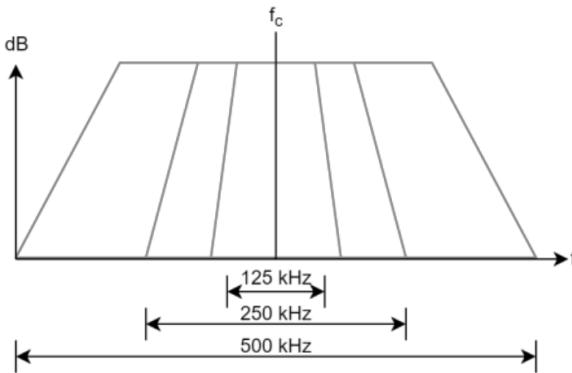


Figure 18: .

adjustment from 137 MHz to 1020 MHz.

CR Coding Rate: Coding Rate (CR) is the FEC rate used by the LoRa modem and offers protection against bursts of interference. A higher CR offers more protection, but increases time on air. Radios with different CR (and same CF/SF/BW), can still communicate with each other. CR of the payload is stored in the header of the packet, which is always encoded at 4/8.

SF Spreading Factor: SF is the ratio between the symbol rate and chip rate. A higher spreading factor increases the Signal to Noise Ratio (SNR), and thus sensitivity and range, but also increases the air time of the packet. The number of chips per symbol is calculated as 2^{sf} . For example, with an SF of 12 (SF12) 4096 chips/symbol are used. Each increase in SF halves the transmission rate and, hence, doubles transmission duration and ultimately energy consumption. Spreading factor can be selected from 6 to 12. SF6, with the highest rate transmission, is a special case and requires special operations. For example, implicit headers are required. Radio communications with different SF are orthogonal to each other and network separation using different SF is possible.

Tx Transmission power:

Payload Payload length:

LoRa has four metrics performance:

➡ Radio

SINR signal-to-interference-plus-noise ratio:

SIR Signal-to-Interference Ratio:

SNR Signal-to-noise ratio:

ToA Time on Air:

PL Path Loss:

➡ Network

PDR Packet delivery ratio:

BER Bit error rate:

PRR Packet Reception Ratio:

$$\text{SF} = \log_2 \frac{R_c}{\text{SR}} \quad (2.1)$$

$$T_s = \frac{2^{\text{SF}}}{\text{BW}_{[\text{Hz}]}} \quad (2.2)$$

$$\text{SR}_{[\text{sps}]} = \frac{\text{BW}}{2^{\text{SF}}} \quad (2.3)$$

$$\text{DR}_{[\text{bps}]} = \text{SF} * \frac{\text{BW}_{[\text{Hz}]}}{2^{\text{SF}}} * \text{CR} \quad (2.4)$$

$$\text{BR}_{[\text{bps}]} = \text{SF} * \frac{\frac{4}{4+\text{CR}}}{\frac{2^{\text{SF}}}{\text{BW}}} \quad (2.5)$$

$$Sen_{[\text{dBm}]} = -174 + 10 \log_{10} \text{BW} + \text{NF} + \text{SNR} \quad (2.6)$$

$$\text{SNR}_{[\text{dB}]} = 20 \log \left(\frac{S}{N} \right) \quad (2.7)$$

$$\text{SNR}_{[\text{dB}]} = 20 \log \left(\frac{S}{N} \right) \quad (2.8)$$

$$\text{BER}_{[\text{bps}]} = \frac{8}{15} \cdot \frac{1}{16} \cdot \sum k = 216 - 1^k \left(\frac{16}{k} \right) e^{20 \cdot \text{SINR} \left(\frac{1}{k} - 1 \right)} \quad (2.9)$$

$$\text{BER}_{[\text{bps}]} = 10^{\alpha} \cdot e^{\beta \text{SNR}} \quad (2.10)$$

$$\text{PER}_{[\text{pps}]} = 1 - (1 - \text{BER})^{n_{bits}} \quad (2.11)$$

$$\text{PRR} = (1 - \text{BER})^L \quad (2.12)$$

$$(2.13)$$

$$\text{RSSI} = \text{Tx}_{power} \cdot \frac{\text{Rayleigh}_{power}}{\text{PL}} \quad (2.14)$$

$$\text{LoRa} = \frac{2^{\text{SF}}}{\text{BW}} \left((\text{NP} + 4.25) + \left(\text{SW} + \max \left(\left[\frac{8\text{PL} - 4\text{SF} + 28 + 16\text{CRC} - 20\text{IH}}{4(\text{SF} - 2\text{DE})} \right] (\text{CR} + 4), 0 \right) \right) \right) \quad (2.15)$$

$$\text{Lora} = n_s = 8 + \max \left(\left[\frac{8\text{PL} - 4\text{SF} + 8 + \text{CRC} + \text{H}}{4 * (\text{SF} - \text{DE})} \right] * \frac{4}{\text{CR}} \right) \quad (2.16)$$

$$\text{Lora} = \frac{1}{R_s} \left(n_{preamble} + \left(\text{SW} + \max \left(\left[\frac{8\text{PL} - 4\text{SF} + 28 + 16\text{CRC} - 20\text{IH}}{4(\text{SF} - 2\text{DE})} \right] (\text{CR} + 4), 0 \right) \right) \right) \quad (2.17)$$

$$\text{GFSK} = \frac{8}{R_{\text{GFSK}}} \left(L_{preamble} + \text{SW} + \text{PL} + 2\text{CRC} \right) \quad (2.18)$$

$$\text{GFSK} = \frac{8}{\text{DR}} (\text{NP} + \text{SW} + \text{PL} + 2\text{CRC}) \quad (2.19)$$

$$(2.20)$$

Base Station (BS)

[19] Nous avons vu en effet plus haut qu'il a été démontré que la méthode CSMA est plus efficace pour le traitement des faibles trafics, tandis que TDMA est nettement plus appropriée pour supporter les trafics intenses.

SF	BW	07	08	09	10	11	12	07	08	09	10	11	12	07	08	09	10	11	12
125																			
07	125	x								x								x	
08			x							x								x	
09				x							x							x	
10					x							x							
11					x														
12					x														
07	250							x								x			
08									x							x			
09		x							x							x			
10			x							x							x		
11				x							x							x	
12			x								x							x	
07	500								x						x				
08										x					x				
09										x					x				
10											x				x			x	
11		x									x					x			
12			x								x					x			

Table 2.14: uyuyuy

	LoRa[5]	SigFox[6]	NB-IoT [7]	Z-Wave[8]	Wi-Fi[9]
Cost	35e	25e	1020e	812e	< 2e
DR	<50 kbps	<100 bps	<200 kbps	<40 kbps	<300 Mbps
Autonomy	<10 years	<10 years	<10 years	<2 years	<10 days
Range (urban)	<5 km	<10 km	<1 km	<100 m	<40 m
Modulation	CSS	BPSK	QPSK	FSK	BPSK/QAM
BW	125/250 kHz	100 Hz	200 kHz	300 kHz	20/40 MHz
Frequency (EU)	868 MHz	868 MHz	LTE bands	868 MHz	2.4/5.0 GHz
Spectrum Cost	Free	Free	Very High	Free	Free
Max. msg/day	Unlimited	1400, 40	Unlimited	Unlimited	Unlimited
Max. payload	243 bytes	120, 80 bytes	1600 bytes	64 bytes	64 KB

Table 2.15: Wireless technologies commonly used in smart buildings [15]

	LTE Cat.1	LTE Cat.0	LTE Cat.M	EC-GSM	NB-LTE*	NB-CIoT
Spectrum	LTE In-Band, Greenfield			GSM In-Band, Greenfield	Greenfield	Greenfield
Release Date/Commercialization	2009	2014	2015/2016	2016/2017	2016/2017	2016/2017
3GPP Release	Rel-8	Rel-12	Rel-13	Rel-13	Rel-13/14	Rel-13/14
Peak Data Rate	DL: 10Mbps UL: 5Mbps	DL: 1Mbps UL: 1Mbps	DL: 1Mbps UL: 1Mbps	DL: 74kbps UL: 74kbps	DL: 128kbps? UL: 64kbps?	DL: 32kbps UL: 48/14.7kbps
System Bandwidth	20MHz	20MHz	1.4MHz	200kHz	200kHz	200kHz
LPWA Network	No	No	No	Yes	Yes	
Link Budget Target	140dB	140dB	155dB	164dB	164dB	164dB
Network Upgrade	No Need	SW Upgrade	To be determined	Yes (HW/SW?)	Yes (HW/SW?)	New Network Clean Slate overlaid with GSM network

Figure 19: .

2.4.7 Summary and discussion

Characteristics	6LoWPAN	LoRaWAN	SigFox	NB-IoT	INGENU	TE-LENSA
Proprietary Standard	IETF	LoRa Alliance	✓	3GPP		
CF [MHz]	902-929 868-868.6	902-928 863-870 and 434	902 868			
Channels	0016 for 2400 0010 for 915 0001 for 868.3	80 for 915 10 for 868 and 780	25			
BW [MHz]	0005 for 2400 0002 for 915 0600 for 868.3	0.125 and 0.50 for 915 0.125 and 0.25 for 868 and 780	0.0001-0.0012			
DR [kbps]	0250 for 2400 0040 for 915 0020 for 868.3	0.00098-0.0219 for 915 0.250-0.05 for 868 and 780	0.1-0.6			
Modulation	QPSK for 2400 BPSK for 915 BPSK for 868.3	LoRa for 915 LoRa and GFSK for 868 and 780 CSS unslotted ALOHA	BPSK and GFSK unslotted ALOHA	QPSK CDMA-like		
CR [dBm]	-085 for 2400 -092 for 915 -092 for 868.3	-137	-137			
Topology		Star, Stars	Star		Star, Tree	Star
ADR		✓	✗		✓	✗
PL		<250B (depends on SF)	12B(UL),8B(DL)		10KB	
Handover		Multi BS	Multi BS			
Security		AES 128b	✗		16B hash, AES 256b	
LS		✓	✗		✗	✗
FEC		AES 128b	✗		✓	✓
Range	10-100 m	5-15 km	10-50 km	1Km		
Battery lifetime	1-2 years	<10 years	<10 years	<10 years		
Uplink			100bps			
Downlink			8 bytes/msg			
Cost		35e	25e	1020e		
max msg/day		Unlimited	140(UL),4(DL)	Unlimited		
max Payload		243B	12(UL),8(DL)	1600B		

Table 2.16: LPWan Characteristics [16]

	SIGFOX	LORAWAN	INGENU	TELENSA
Modulation	UNB DBPSK(UL), GFSK(DL)	CSS	RPMA-DSSS(UL), CDMA(DL)	UNB 2-FSK
Band	S UB -GH Z ISM:EU (868MHz), US(902MHz)	S UB -GH Z ISM:EU (433MHz 868MHz), US (915MHz), Asia (430MHz)	ISM 2.4GHz	S UB -GH Z bands including ISM:EU (868MHz), US (915MHz), Asia (430MHz)
DR	100 bps(UL), 600 bps(DL)	0.3-37.5 kbps (L O Ra), 50 kbps (FSK)	78kbps (UL), 19.5 kbps(DL)	62.5 bps(UL), 500 bps(DL)
Range	10 km (URBAN), 50 km (RURAL)	5 km(URBAN), 15 km (RURAL)	15 km (URBAN)	1 km (URBAN)
Channels	360 channels	10 in EU, 64+8(UL) and 8(DL) in US plus multiple SFs	40 1MHz channels, up to 1200signals per channel	multiple channels

Table 2.17: [17]

Standard	802.15.4k	802.15.4g	Weightless-W	Weightless-N	Weightless-P	DASH 7 Alliance
Modulation	DSSS, FSK	MR-[FSK, OFDMA, OQPSK]	16-QAM, BPSK, QPSK, DBPSK	UNB DBPSK	GMSK, offset-QPSK	GFSK
BW	ISM S UB -GH Z, 2.4GHz	ISM S UB -GH Z, 2.4GHz	TV white spaces 470-790MHz	ISM S UB -GH Z EU (868MHz), US (915MHz)	S UB -GH Z ISM or licensed	UB -GH Z 433MHz, 868MHz, 915MHz
DR	1.5 bps-128 kbps	4.8 kbps-800 kbps	1 kbps-10 Mbps	30 kbps-100 kbps	200 bps-100kbps	9.6,55.6,166.7 kbps
Range	5 km (URBAN)	up to several kms	5 km (URBAN)	3 km (URBAN)	2 km (URBAN)	0-5 km (URBAN)
MAC	CSMA/CA, CSMA/CA or A LOHA with PCA	CSMA/CA	TD-MA/FDMA	slotted A LOHA	TD-MA/FDMA	CSMA/CA
Topology	star	tar, mesh, peer-to-peer	star	star	star	tree, star
PL	2047B	2047B	>10B	20B	>10B	256B
Security	AES 128b	AES 128b	AES 128b	AES 128b	AES 128/256b	AES 128b
Forward error correction	✓	✓	✓	✗	✓	✓

Table 2.18: [17]

Phy protocol	IEEE 802.15.4	BLE	EPCglobal	Z-Wave	LTE-M	ZigBee
Standard		IEEE 802.15.1				IEEE 802.15.4, ZigBee Alliance
BW(MHz)	868/915/2400	2400	860-960	868/908/2400	700-900	
MAC	TDMA, CSMA/CA	TDMA	ALOHA	CSMA/CA	OFDMA	
DR (bps)	20/40/250 K	1024K	varies 5-640K	40K	1G (up), 500M (down)	
Throughput				9.6, 40, 200kbps		
Scalability	65K nodes	5917 slaves	-	232 nodes	-	
Range	10-20m	10-100m				
Addressing	8 16bit	16bit				

Table 2.19: IoT cloud platforms and their characteristics [7]

	802.15.4	802.15.4e	802.15.4g	802.15.4f
CF	2.4Ghz (DSSS + oQPSK) 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK)	2.4Ghz (DSSS + oQPSK, CSS+DQPSK) 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK)	2.4Ghz (DSSS + oQPSK, CSS+DQPSK) 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK)	2.4Ghz (DSSS + oQPSK,CSS+DQPSK) 868Mhz (DSSS + BPSK) 915Mhz (DSSS + BPSK) 3~10Ghz (BPM+BPSK)
DR Differences	Upto 250kbps -	Upto 800kbps Time sync and channel hopping	Up to 800kbps Phy Enhancements	Mac and Phy Enhancements
PL Range Goals	127 bytes 1~75+ m General Low-power Sensing/Actuating	N/A 1~75+ m Industrial segments	Up to 2047 bytes Upto 1km Smart utilities	N/A N/A Active RFID
Products	Many	Few	Connode (6LoWPAN)	LeanTegra PowerMote

Table 2.20: IEEE 802.15.4 standards [18]

SF	Sensitivity[dBm]	DR[kb/s]		
6	-118	9.38		
7	-123			
8	-126			
9	-129			
10	-132			
11	-134.5			
12	-137			

Table 2.21: hg hg

SF/BW	125kHz		250kHz		500kHz	
	Sensitivity [dBm]	Bit Rate [kb/s]	Sensitivity	Data Rate	Sensitivity	Data Rate
-	Sensitivity [dBm]	Bit Rate [kb/s]				
6	-118		-115		-111	
7	-123	5.468	-120		-116	
8	-126	3.125	-123		-119	
9	-129	1.757	-125		-122	
10	-132	0.976	-128		-125	
11	-133	0.537	-130		-128	
12	-136	0.293	-133		-130	

Table 2.22: Receiver sensitivity [dBm]

DR	Modulation			Max transmission unit		BR x kbit/s
	SF	BW [kHz]	CR	Total [B]	Payload [B]	
0	12	125	4/6	64	51	0.25
1	11	125	4/6	64	51	0.44
2	10	125	4/5	64	51	0.98
3	9	125	4/5	128	115	1.76
4	8	125	4/5	255	242	3.125
5	7	125	4/5	255	242	5.47
6	7	125	4/5	255	242	11
7		125	4/5	255	242	50

Table 2.23: oioioi

Feature	Wi-Fi	802.11p	UMTS	LTE	LTE-A
Channel MHz	20	10	5	1.4, 3, 5, 10, 15, 20	<100
Frequency band(s) GHz	2.4 , 5.2	5.86-5.92	0.7-2.6	0.7-2.69	0.45-4.99
BR Mb/s	6-54	327	2	<300	<1000
Range km	<0.1	<1	<10	<30	<30
Capacity	Medium	Medium	X	✓	✓
Coverage	Intermittent	Intermittent	Ubiquitous	Ubiquitous	Ubiquitous
Mobility support km/h	X	Medium	✓	<350	<350
QoS support	EDCA Enhanced Distributed Channel Access	EDCA Enhanced Distributed Channel Access	QoS classes and bearer selection	QCI and bearer selection	QCI and bearer selection
Broadcast/multicast support	Native broadcast	Native broadcast	Through MBMS	Through eMBMS	Through eMBMS
V2I support	✓	✓	✓	✓	✓
V2V support	Native (ad hoc)	Native (ad hoc)	X	X	Through D2D
Market penetration	✓	X	✓	✓	✓
DR	<640 kbps	250 kbps	106424 kbps	✓	✓

Table 2.24: An example table.

2.5 IoT SDN

SDN is the equipment virtualisation, NFV is the function virtualisation

2.5.1 Application

2.5.2 Control

2.5.3 Data

2.5.4 Summary and discussion

2.6 IoT Security

2.6.1 Application

2.6.1.1 MQTT

2.6.1.2 Blockchain

Blockchain Layers

- ➡ Transaction & contract layer
- ➡ Validation layer (forward validation request)
- ➡ Block Generation Layer (PoW, PoC, PoA, PoS, PBFT)
- ➡ Distribution Layer
 - Consensus algorithms
 - ➡ Proof of Work (PoW)
 - ➡ Proof of Capacity (PoC)
 - ➡ Proof of Authority (PoA)
 - ➡ Proof of Stake (PoS)
 - ➡ Proof of Byzantine Fault Tolerant (PBFT)

2.6.2 Network

2.6.3 MAC/Radio

2.6.3.1 LoRaWan

2.6.3.2 Zigbee

2.6.3.3 802.15.4

2.6.4 Summary and discussion

3 | 1 [1]

1 [1]

Contents

3.1	Introduction	44
3.1.1	problem statement	44
3.1.2	Background	44
3.1.3	Purpose (Goal)	45
3.1.4	Limitations	45
3.1.5	Method	45
3.2	Related work	46
3.3	Background	46
3.3.1	Hardware	46
3.3.2	Operating system	47
3.3.3	Communication protocol	48
3.3.4	Workspace and tools	48
3.4	Proposed	48
3.4.1	Drivers and firmware	49
3.4.2	CoAP server	49
3.4.2.1	Testing	50
3.4.2.2	Final prototype	50
3.5	Experimentation	50
3.5.1	Range	50
3.5.2	Response time	51
3.5.3	Connection speed	52
3.5.4	Power consumption	53
3.6	Results	55
3.6.1	Range	55
3.6.2	Response time	55
3.6.3	Connection speed	56
3.6.4	Power consumption	56
3.6.5	Project execution	56

Abstract

The exponential usage of messaging services for communication raises many questions in privacy fields. Privacy issues in such services strongly depend on the graph-theoretical properties of users' interactions representing the real friendships between users. One of the most important issues of privacy is that users may disclose information of other users beyond the scope of the interaction, without realizing that such information could be aggregated to reveal sensitive information. Determining vulnerable interactions from non-vulnerable ones is difficult due to the lack of awareness mechanisms.

To address this problem, we analyze the topological relationships with the level of trust between users to notify each of them about their vulnerable social interactions. Particularly, we analyze the impact of trusting vulnerable friends in infecting other users' privacy concerns by modeling a new vulnerability contagion process. Simulation results show that over-trusting vulnerable users speeds the vulnerability diffusion process through the network. Furthermore, vulnerable users with high reputation level lead to a high convergence level of infection, this means that the vulnerability contagion process infects the biggest number of users when vulnerable users get a high level of trust from their interlocutors. This work contributes to the development of privacy awareness framework that can alert users of the potential private information leakages in their communications.

3.1 Introduction

3.1.1 problem statement

3.1.2 Background

Internet of Things (IoT) is a concept aiming at connecting all things to the Internet [1]. The different kinds of devices range from simple sensor devices to complex machines such as industry robots. Home automation has been available for a few years in the forms of timers and remotely controlled devices, such as lights, garage door, and climate control equipment. Also in the industry and workplace, there are current systems that have some of the functionality of IoT, e.g., sensors in robots and machines which keep track of the system status so that maintenance can be scheduled at the right time. However, these systems or sensors rarely communicate with each other or make decisions based on other sensor values; instead they depend on input from a user. In the same way cellphones connected people and made them constantly connected to the Internet, IoT will connect devices and make them constantly connected to the Internet [2]. In theory, this could lead to a future with autonomous technology all around us. The benefits could be huge as it

would save time and energy for both the individual at home and for the industry [3]. IoT could be used in industry to automate power-heavy tasks to run when the electricity price is low. This principle can also be applied for the home user with laundry machines and charging of e.g. electric cars. This practice would lead to reduced energy consumption and thus a reduced environmental footprint. i3tex AB wants to investigate potential fields of applicability of this upcoming technology. i3tex AB has customers in the automotive, communication, and pulp industries; those customers have made inquiries on how to integrate IoT and sensor networks into production. As technology evolves, size and energy consumption of the IoT devices will decrease and computation power will increase [4]. This reduction in size and energy consumption, together with the increased computing power, will open up new fields for IoT. Thus, i3tex AB want to have an IoT platform to present to their current and potential customers. The interest in IoT is rapidly increasing, and thus, in the near future, the number of devices connected to the Internet is expected to increase rapidly. To support this huge increase in both number of connected devices and the sheer amount of data that will be sent over both wired and wireless networks, the communication technology must be ready [5].

3.1.3 Purpose (Goal)

The purpose of this project is to find and examine a communication method for devices that are made to be a part of IoT. This will be done by examining the available technologies and then developing a prototype based on the findings, which will be used for examining the communication method. This project will examine the physical, link, and network layers [6, 7] of the Open Systems Interconnection model (OSI model) [8], in order to find suitable technologies on the market. As IoT is still only defined as a concept, there are several technologies to take into consideration and examine in further detail. The prototype will be delivered to i3tex AB together with appropriate documentation, e.g. technical specification, hardware manual, software manual, and API specification.

3.1.4 Limitations

To be able to achieve the project goal within the available time, limitations need to be defined in the three main areas of: Operating System (OS), hardware, and communication method. The OS will not be custom-made, but rather selected amongst those already on the market. Thus, to simplify the hardware selection, only those OSs which already have hardware support that meets the requirements will be taken into consideration. Furthermore, support for either 6LoWPAN, ZigBee, or Bluetooth Low Energy (BLE) as communication method is required, since development to make those standards available is outside the scope of the project. On the hardware side, the limitations will be to only use existing devices and parts as there will be no time for developing hardware or Printed Circuit Boards (PCBs). However, the hardware does not need to have an integrated radio transceiver, but needs to support at least one transceiver supporting IEEE802.15.4 [6]. Thus, communication methods will be primarily selected from specifications building on the IEEE 802.15.4.

3.1.5 Method

To ensure that the right technologies were selected and investigated, the first phase of the project was a literature study. The study served as a foundation when developing and

performing the evaluation of the communication methods. At the end of the phase, a requirements specification was formulated to serve as a platform for the next phase. After the literature study, a selection process was performed, where the most promising technologies that met the requirements were examined in further detail and brought into the development phase. This process included the selection of development tools and other decisions bound to the product development. In the development phase, the chosen set-up was configured and assembled to prepare for testing; it was then tested according to throughput, range, latency, and energy consumption. Throughput was measured in kilobyte per second (KB/s) and tested by transferring data of different sizes in both congested and uncongested network set-ups to simulate real world and lab environments. The same set-up was used to measure the latency of a transmission, which was measured in microseconds (ms). Range was calculated instead of measured, with meters (m) as the unit. The power consumption was measured in watts (W). Each week a meeting with the company supervisors was performed, to keep the work on the right track. Here, feedback was given and other issues and questions handled.

3.2 Related work

3.3 Background

The goal of the implementation phase is to have a working prototype for future assessment. To make the process of implementing the prototype possible, the first part of the implementation process will be to create a set of requirements. When these are set, the process will continue by comparing the data from chapter 2 to find the candidates that fulfil the requirements. After the technologies are selected, the process will continue with setting up the workspace, which includes the platform for development and the required tools to build, debug and test the prototype. Finally, when these three steps have been performed, the next step will be to start with the actual prototype development.

Requirements

As the time dedicated for development is limited, the requirements have to make sure that the development process does not run into any major obstacles. All parts of the total prototype need to fit together seamlessly. However, the hardware platform and the operating system are tied most closely together. Therefore, they need requirements that complement each other, so that they can act as a platform for software development. Naturally, both the hardware and the operating system requirements might have to be altered slightly to enable the best match.

3.3.1 Hardware

Each platform examined in chapter 2 has different strengths and weaknesses. When looking at the MCU, OpenMote has a ARM Cortex-M3 which is more powerful compared to the other two alternatives: it features a 32bit 32MHz core with 32KB RAM and 512KB flash memory compared to the MSP430 16bit 25MHz core with 10KB/100KB memory configuration. In terms of peripherals all three platforms are comparable, with similar amount of DAC, GPIO pins, and external busses. All of the platforms have a temperature sensor and an accelerometer, but OpenMote also features an light/uv-light sensor and a voltage

sensor built into the MCUs ADC. The MSB430 platform has a somewhat lower power consumption in active RF mode thanks to the less power-hungry sub-GHz transceiver. On the other hand, the less powerful MSP430 MCU has a better deep sleep power consumption, but as the radio is not integrated in the chip as it is in the CC2538 SoC, that advantage is offset by the external transceiver. Comparing the transceivers, there are two 2.4GHz models and one subGHz model; the sub-GHz CC1100 has a higher transmit power of 10dBm compared to 0dBm for CC2420 and 7dBm for CC2538. Also the sensitivity is similar for all the alternatives but gives a slight advantage to CC2538 with -97dBm compared to -95dBm and -93dBm for CC2420 and CC1100. Using these numbers and Friis range equation (equation 4.1) the range of each transceiver with a Fade margin (FM) of 20dB can be seen in figure 3.1. The benefits of working with lower frequencies can clearly be seen as the theoretical range of CC1100 is almost 3 times longer than the 2.4GHz transceivers.

Adding all this information together, the choice of platform will land on OpenMote with the CC2538SoC. It both has a MCU with more memory and better performance, and a transceiver with really good characteristics both in terms of energy consumption and range. Also OpenMote is the only option that can act as a border router using OpenBase; it lets the SoC interface with USB, UART, JTAG, and Ethernet, which enables the standalone border router mode without the need to be connected to a computer or other hardware. The OpenBattery extension lets the SoC operate as a node in a mesh network and provides a dual AAA battery slot connected to the PCB.

Resume

Transceiver with IEEE 802.15.4 or IEEE 802.15.1 Integrated sensor/sensors MCU with low power mode under 5MicroA Wakeup from low power mode with timer Border router ability Joint Test Action Group (JTAG) support

3.3.2 Operating system

As a modern operating system can be compiled to match almost any hardware, the most important thing to have in mind is the out-of-the-box hardware support. Only RIOT and Contiki have full support for the ARM Cortex-M3 of the considered operating systems and thus both TinyOS and freeRTOS are directly eliminated as developing the support would take too much time. Compared to RIOT, Contiki also has full driver support for the sensors and transceiver, which should decrease the implementation time significantly. When compiled into binary form, RIOT uses less RAM and ROM and thus probably is a bit faster compared to Contiki, which could be important if the application consumes much resources. The lower memory usage might also give RIOT an advantage in being future-proof. Contiki also has support for soft real-time scheduling compared to the hard real-time scheduling of RIOT; this is however not crucial, as the software that will be running on the OS does not have any hard real-time constraints. Both RIOT and Contiki have support for 6LoWPAN but no support for either ZigBee or BLE; this is due to the fact that these are proprietary stacks. Support could be added but would take some time to customize for the given OS. What gives Contiki the largest advantage is that it also have border router software ready for deployment, which in the RIOT case would have to be developed. All in all, as the project has such a limited time frame, Contiki will be selected as the OS; this, mainly because Contiki comes with most advantages time-wise; this choice means that the focus of the software development will be the creation of a test and evaluation system.

Resume

Support for the SoC/MCU and transceiver 6LoWPAN, ZigBee or BLE stack Soft real-time RAM and ROM footprint matching the hardware

3.3.3 Communication protocol

The OpenMote platform has a IEEE 802.15.4 transceiver and thus supports both ZigBee and 6LoWPAN; this means that BLE is not an option. As ZigBee does not have full IPv6 support yet and is not integrated into Contiki, the natural choice is 6LoWPAN. This choice will not only save some development time but also enables evaluation of the header compression. As seen in figure 3.2, the 6LoWPAN stack in Contiki will replace the IP stack while maintaining the same functionality. As the functionality is the same, TCP and HTTP will work with 6LoWPAN, but including them in the source increases the OS build size considerably. On top of the UDP layer, Contiki also has a working implementation of CoAP that can be used for retrieving data from the nodes in a power efficient manner. CoAP is a stateless protocol that uses the HTTP response headers to achieve a very low overhead in transmissions while using application level reliability methods to ensure packet delivery.

Resume

IPv6 addressing support Existing OS support Network type is mesh UDP

3.3.4 Workspace and tools

The ARM Cortex-M3 chip that OpenMote and CC2538 is built upon requires the GCC ARM Embedded compiler. This tool-chain is free and runs on both Linux, OSX, and Windows; however, there is no bundled development application so a secondary application for programming is needed. In Windows, there are several Integrated Development Environments (IDEs) such as IAR Workbench ARM [30], Code Composer Studio [31] and the Eclipse plug-in ARM DS-5 [32, 33]; these IDEs use various proprietary toolchains and have a price tag ranging from free to several thousand SEK. Most of the IDEs also have a code size restriction for the free versions. To minimize the costs, the development machine development machine used in this project will run Ubuntu 14.04 LTS, the used tool-chain is GCC ARM Embedded, and Geany is used as the code development application. To analyse the network traffic in real-time, the open source tool Wireshark is used together with a IEEE 802.15.4 packet sniffer. Together with a laptop, the packet sniffer will grant the ability to traverse the mesh network and analyse the network in real-time as it is seen by the nodes.

3.4 Proposed ...

The goal of the development process is to have a functional border router and at least two nodes to be able to test how response time and throughput differs with each hop in a mesh network. To be able to measure response time and throughput, each node needs to have a CoAP server which can respond to ping and also receive an arbitrary amount of data for throughput measurement. It is desirable for each node to be able to send information about each sensor so the project can be used as a tech-demo. The first part of the development was to set-up of the workspace and tools mentioned in section 3.1.5.

Ubuntu OS was installed in a VirtualBox Virtual Machine to make it easier to duplicate and backup; this procedure gave a noticeable decrease in performance and it is recommended to have a dedicated native Ubuntu machine for this type of development. Even though Ubuntu uses an easy-to-use package system, there were some problems in finding a version of GCC ARM Embedded tool-chain that was compatible with Contiki's built-in simulator Cooja [34, 35]; eventually, version 4.82 was used to successfully build Contiki. Cooja is a useful tool for testing and debugging network configurations but does not have support for the CC2538 MCU; instead, nodes called Cooja Motes are simulated with generic hardware. As Cooja is written in Java and runs in a JVM, Oracle Java 1.8 was also installed.

3.4.1 Drivers and firmware

Figure 3.3 shows an overview of the full system. The foundation is the SoC with the MCU, transceiver, and sensors. The Contiki operating system implements the soft real-time kernel together with the firmware for the SoC/MCU and the drivers for the peripherals and sensors. The last part is the communication stack, which provides TCP and UDP connectivity over 6LoWPAN. On top of the TCP and UDP protocols, HTTP and/or CoAP can be implemented. The firmware required for the OS to work properly on the hardware platform was already implemented. However, the drivers for the I²C bus and the sensors were not implemented. The I²C driver is required for the sensor drivers which in turn enables the MCU to communicate with the sensors on the OpenBattery platform.

3.4.2 CoAP server

In order to make each node's sensor data accessible, a CoAP sever was implemented as an application running on top of Contiki. A CoAP server in general can handle any number of resources; in this implementation, one resource was made for each sensor value i.e. temperature, light, humidity, and core voltage. The temperature, light, and humidity sensors all work in a similar fashion. When their value is requested, the I²C bus is initialized and then a request is sent over the bus. When the response with the value arrives, that data is put into either a plain-text or JavaScript Object Notation (JSON) formatted message depending on the request and then sent back to the requester. As the core voltage sensor is part of the MCUs ADC, that value is retrieved by simply getting data from a register (a somewhat faster operation). As a buffer for testing throughput speed was also needed, a resource with a circular buffer was implemented. This resource is configured with CoAP's block-wise transfer functionality for arbitrary data size; however, the buffer in itself is only 1024Kb to allow the program variables to fit into the ultra low leakage SRAM. For testing purposes, the data could have been discarded instead of actually saved into the buffer, but then the transfer can not be verified. Resources are defined by paths as CoAP works in a very similar way as HTTP.

Each resource is registered in the server with its path, media type, and content type. When a package arrives on the CoAP port, the server starts to break down the package to be able to direct it to the right resource. It starts with verifying that the package is actually a CoAP package, and then it checks the path and sends it to the correct resource. The resource then inspects the method field in the package header to direct the incoming data to the right function. CoAP package method can be either GET, PUT, POST or DELETE. This function then inspects the request media type and answer content type so that the function can parse the request and send a correctly formatted answer. If the resource does not

implement the received method, the server responds with 405 Method not allowed and if the content/media type is not supported the answer is 415 Unsupported Media Type. The content/media types are text/plain, application/json, application/exi, and application/xml.

3.4.2.1 Testing

Contiki is shipped with a simulation tool called Cooja which is written in Java; it can simulate an arbitrary number of nodes with different roles and configurations. All simulation data, such as radio packages and node serial output may be viewed through different windows and exported to various formats. Unfortunately Cooja did not have support for ARM Cortex-M3, but the general set-up was still tested by using Cooja Motes, which are nodes without specified hardware, and MSP430 nodes such as Wismote or Skymotes. With this simulator the basic understanding of the communication between nodes was gained; also, before the hardware arrived, early testing was performed to test the OS and application software.

3.4.2.2 Final prototype

The final prototype consists of four OpenBattery nodes and one OpenBase border router. Both the nodes and the border router are deployed with Contiki. Each node runs a CoAP server, described in section 3.2.2, on top of the OS in its own thread. The border router runs a router software called 6lbr that acts as translator between Ethernet and IEEE 802.15.4 [36]. Both types of hardware are configured with a 8Hz Radio Duty Cycle (RDC) driver to keep the power consumption to a minimum. RDC is a OS driver that cycles the listening mode of the transceiver to reduce power consumption. As Contiki puts the MCU into Low Power Mode (LPM) when no function is running and the transceiver is off, the RDC driver indirectly controls when the MCU is in LPM. When using the RDC protocol, the nodes repeatedly send messages until the target node wakes up and sends an Acknowledge packet (ACK); this makes communication seamless, even though most of the time the nodes transceivers are not active. Also, an always-on RDC driver, where the transceiver is constantly listening, will be used to be able to look at the performance impact of the 8Hz RDC.

3.5 Experimentation

In this chapter the results from each type of assessment are presented. The first assessment is range, followed by response time, after that connection speed, and finally the power consumption. The only assessment that is not performed on the prototype is the range assessment.

3.5.1 Range

Range is very hard to measure without advanced equipment and isolated rooms but can be roughly estimated with equation 4.1 called Friis range equation [37]. P_t is the sender transmit power, P_r the receiver sensitivity, d is the distance between the antennas in meters, f is the signal frequency in hertz, and λ is the wavelength. G_t and G_r is the antenna gain for the transmitter and the receiver. The last term in equation 4.1, when inverted, is the Free-space path loss (FSPL) and can be expanded as shown in equation 4.3. P_r (dB) =

$$P_t + G_t + G_r + 20 \log_{10} () 4d c f FSPL(dB) = 20 \log_{10} (d) + 20 \log_{10} (f) 147.56 = (4.1) (4.2)$$

(4.3) Unlike Friis range equation, the Link budget equation 4.4 also takes external loss like FM into account [38]. This is needed to make a correct estimation of the actual range as there are several things in the environment that obstructs and distorts the signal. $P_r = P_t + G_t + G_r + FM$ FSPL (4.4) Combining equation 4.1, 4.3 and 4.4 gives us the equation for the estimated distance as seen in equation 4.5. $d = 10 \times P_t + G_t + G_r + P_r + FM + 147.56 - 20 \log_{10} (f)$

With this equation an estimation of the transceiver range can be made for different FMs and transmit powers. When deployed, the transceiver is configured to only accept packages with a signal strength of -70dBm and above to minimize packet loss and corruption. The antenna gain for OpenMote is 0dBi and can thus be omitted. Figure 4.1 shows a comparison between three different levels of FM: 0dB, 10dB, and 20dB. A FM of 0dB means that there is no signal loss except the FSPL and this is very hard to achieve outside of a lab environment. When increasing the FM to 10dB, which corresponds to a normal home environment, the maximum range drops to 22m. However, in these kind of environments the desired range is usually around 10m which would let the device reduce the transmit power to around 0dBm. Finally, the FM is increased to 20dB which is roughly what it would be in a office or industrial environment. The maximum range in this environment is now reduced to only 7m when transmitting at maximum power.

Response time Before measuring the response time, some theoretical estimations are needed to be able to evaluate the real values. The theoretical values are based upon the radio duty cycle (RDC) and the average response time to reach a node can thus be derived from equation 4.6, 4.8, 4.9 and 4.10. As each node only

3.5.2 Response time

Before measuring the response time, some theoretical estimations are needed to be able to evaluate the real values. The theoretical values are based upon the radio duty cycle (RDC) and the average response time to reach a node can thus be derived from equation 4.6, 4.8, 4.9 and 4.10. As each node only checks the radio every 125ms, this duration combined with the data packet send time of 4ms (equation 4.7.) and ACK send time corresponds to the worst case delivery, as the node needs to wait a whole cycle before being able to send the package the desired node. When the target node is already listening, the best case delivery time is 5ms. Thus, the average theoretical delivery time to reach any adjacent node is 67.5ms. Radio duty cycle: Transfer time: $1s = 0.125s = 125ms$ 8 (4.6) 133B + 4B 4ms 31.25KB/s (4.7) Worst case delivery: $125ms + 4ms + 1ms$ (4.8) Best case delivery: 4ms + 1ms (4.9) $130ms + 5ms = 67.5ms$ (4.10) 2 The delivery time is only calculating the time to send a packet over a link, but when calculating the response time, the acknowledge (ACK) response has to be included in the calculation. Each ACK also needs to wait for the target node to be awake, adding one more instance of average delivery time, resulting in 125ms in average response time. This time will multiply with each hop, resulting in equation 4.11, 4.12 and 4.13. Avg. delivery: Avg. response time: $(2 \times 67.5ms) \times \text{hops}$ (4.11) Best case response time: $(2 \times 5ms) \times \text{hops}$ (4.12) Worst case response time: $(2 \times 130ms) \times \text{hops}$ (4.13) After doing a test with real nodes set-up with a 8Hz RDC with three hops, as seen in figure 4.2, the values in table 4.1 were obtained. Each node was pinged 200 times at a one minute interval to simulate some traffic on the network. What can clearly be seen in the average field of the table is that the average of 765ms is much higher than the expected average of 135ms; the difference is mainly due to the worst-case pings that in some cases had response times up to 30 seconds. However, when looking at the geo-

metrical mean which is better at smoothing out big spikes seen in figure 4.3, the observed response time is still 265ms which is a bit longer than the expected worst case response time for one hop. Also, for two and three hops the observed average is high, but the geometrical mean shows that this is due to the spikes. The estimated response time for two hops is 270ms which as seen in the geometrical mean table 4.1 is way off by 500ms. The same observation goes for three hops where the observed geometrical mean response time is 1181ms which compared to the estimated response time of 405ms is significantly higher.

With these observations in mind, the estimation could be described much better with equation 4.14. which would result in an average response time of 266, 532 and 1064 ms for one, two and tree hops. However, this would mean that the response time is doubled for one hop and then doubled for each consequent hop making the response time exponential which should not be the case.

With the RDC disabled, i.e. the transceiver is always listening and the MCU does not go into sleep mode, the response time is completely different. As seen in table 4.2 the average response time is around 12ms per hop and the spikes seen in the response time for 8Hz RDC is gone. Furthermore, the estimated best case response time of 10ms is very close to the observed average response time. The response time also scales to the number of hops as expected and is roughly 12ms per hop. It appears there might be a problem in

3.5.3 Connection speed

Connection speeds can be measured in several ways each with their own different pros and cons. One of the most popular ways is throughput, i.e. the amount of data over the link is divided by the time it took to reach the target. However, this gives a false picture of how fast the connection actually is from the developers point of view, as the measured data does not only contain application data but also headers and checksums. IEEE 802.15.4 has a theoretical data rate of 250kb/s as seen in equation 4.15. but this is only a measure of how many bits per second the transceiver is able to output. The application data part, when using no header compression, is only 41% of the total transfer. Thus, resulting in a theoretical application data rate, also called goodput, of only 12.81KB/s. Data rate: $250\text{kb/s} = 31.25\text{KB/s}$ $133\text{B} / 54\text{B} = 0.59$ $133\text{B} \times 0.59 = 78.67\text{B}$ Theoretical goodput: $31.25\text{KB/s} \times 0.59 = 12.81\text{KB/s}$ Overhead: (4.15) (4.16) (4.17) When using CoAP as the application level protocol, each package can carry either 32 or 64 bytes of application data. In practice, the 64B mode is only applicable when sending packages between nodes on the same mesh network, as the addressing fields then can be fully compressed. When using applications outside the mesh network, each package can only carry 32B of data, resulting in a packet size of 111B as shown in equation 4.18; this does not affect the theoretical data rate but has a noticeable impact on the goodput due to the large overhead of 71as shown in equation 4.19. To be able to use the full data rate, the application needs to use a protocol without handshakes, i.e. UDP, as the transceiver then can send the packets as fast as physically possible. CoAP is implemented on top of UDP and thus has a low transport layer overhead, but uses its own mechanism for handshaking, delivery and ordering. The theoretical CoAP application throughput can be estimated by looking at the average response time of the node and then add that time to the the data delivery time. Each package needs to be acknowledged before the next package is sent, and thus the node response time needs to be taken into consideration. When doing so, the throughput as calculated in equation 4.20. is only 1.64KB/s and thus the theoretical goodput is reduced from 12.81KB/s down to 0.48KB/s as shown in equation 4.21. Packet size: $133\text{B} + 54\text{B} = 187\text{B}$

$$= 111B \text{ Actual overhead: } 79B = 0.71 \cdot 111B \quad (4.18) \quad (4.19)$$

Using the packet size of 111B together with the theoretical response time from equation 4.11 would give the results shown in figure 4.4. To verify these calculations, the same test set-up as shown in figure 4.2, which also was used in section 4.2, was used to test throughput and goodput at different number of hops. Each node was sent 1KB data each minute for 200 minutes; the time from the first package sent to the final acknowledge packet received was measured for each 1KB transmission. The first test was performed with an RDC of 8Hz and resulted in the values shown in figure 4.5. As the chart shows, the theoretical throughput and goodput is much higher than the observed values, but this is due to the fact that the actual average response time is higher than the theoretical one. With some calculations made the observed throughput and goodput are within range of what is expected, given the observed response times in table 4.1. Equation 4.22 uses the observed values to calculate the average response time, given the values in figure 4.5.

3.5.4 Power consumption

To measure power on devices that use very low power and also changes the power consumption very rapidly and frequently is not an easy task. According to the currency specification from the CC2538, the different power modes have the consumption seen in table 4.3 using the built in voltage regulator TSP6750 that switches the input voltage down from the 3V to 2.2V. The components on the OpenBattery supplied directly by the 3V batteries have the current and power consumption specifications as seen in table 4.4.

Given these power profiles, combined with the time it takes to receive and transmit packages, and retrieve a measurement, the theoretical power for one RDC cycle results in the chart seen in figure 4.7. The node starts in sleep mode using 112.81W and after 109ms wakes up and goes into RX mode where a request for a sensor value is received. The node then switches off the radio and fetches the sensor value. After the value is retrieved from the sensor, the radio is once again put in to RX mode for a Channel Clear Assessment (CCA) before entering TX mode and sending the payload. The transmission is successful and the node goes into RX mode to listen for the ACK, when it is received the node enters sleep mode again. For this cycle the average power consumption is 4.8mW which would drain the 2250mWh batteries in 19 days.

However, as the nodes have a RDC running at 8Mz, most of the time there will be no package for the node to receive and thus no measuring and transmitting, as seen in figure 4.8. This cycling reduces the average power consumption to 0.47mW, which would make the batteries last for ca 200 days. The goal is to have a node that can run for one year without having to change the batteries and to be able to do this on 2xAAA batteries with 750mAh the average consumption has to be under 257W as calculated in

To verify these assumptions, we used a Keithley 2280S power supply [39] to measure the total current draw of the prototype. The node was connected to the power supply, which was set up to make 277 measures each second with a supply voltage of 3V. Several measurements were performed. One of the most interesting ones can be seen in figure 4.9. In this picture, we can clearly see the different operating modes, as the node performs 3 transmissions during the interval. In the first transmission at the 1.6s mark, the strobing feature of the RDC protocol is seen as the package is sent 5 times before the receiving node is awake and can receive the package. In the two following transmissions, the package is delivered on the first try. As our measurement is limited to 277Hz, the current peaks when only waking up to listen for traffic are sometimes missed, and the peak value is hard to extract; but the 8Hz RDC cycle is still visible. The average power consumption for these

cycles is 8mW, which would make the batteries only last for 11 days. However, when taking the average of a measuring series without any transmissions, the average goes down to 4mW, which increases the battery time to 23 days. The theoretical sleep power of 0.11mW compared to the measured of 3mW is what makes the average power consumption that high.

Reducing this power consumption by a tenfold would result in an average consumption of 0.39mW, which is closer to the theoretical average power consumption. A discovery made when measuring the power was that the nodes consumed less power when supplied with a lower input voltage. Simply by reducing the voltage from 3V to 2.6V reduced the power consumption in LPM by 15%. However, this reduction could affect the range of the nodes.

Internet of Things can be realised in several ways as there are still many viable options on the market, mainly in terms of hardware, operating systems, and communication standards. Given the recent development in the field, Thingsquare recently released a technology demo using the same practices as used in this thesis; the choices taken are on track with the latest development [40]. Also, both Google and Microsoft have announced that they are developing IoT OSs. When these products are released, it would be very interesting to compare them with Contiki. It would be exciting to see if an open-source project can surpass the commercial offerings in terms of speed, RAM and ROM footprint, and device support. Furthermore, an in-depth comparison between RIOT and Contiki would give much insight into the kind of OS practices that benefit IoT development the most. Google have also started to develop a substitute for 6LoWPAN and UDP that they have named Thread [41]. As 6LoWPAN and ZigBee, it runs on top of IEEE 802.15.4 and thus might be able to out-compete the existing implementations. Google promises lower latencies and power consumption compared to the existing technologies.

The prototype

The prototype development took more time than initially planned; mostly because of the complexity of the OS, but also due to bugs in the untested drivers. The prototype combines the technology from each field, i.e. hardware, OS, and communication protocol, and fulfils the requirements set in section 3.1.1. Even though the OS is relatively simple, compared to Linux, Windows, and OS X, understanding the mechanics of the RDC driver and the LPM driver was difficult, but necessary to be able to interpret the test results. The prototype worked very well during most of the testing, with only a few unforeseen deviations. One occurred during the power measurement, where the power consumption in low power mode tripled in one of the test series; this behaviour could not be reproduced and is therefore not included in the results. Also, in the early stages when working with the 8Hz RDC driver, packet losses over 50% were recorded for packets with more than one hop; this problem was solved, when a new version of radio driver was released by the OS development team. Selecting OpenMote to be the hardware platform together with Contiki as the OS, was a very good choice as companies are starting to build their IoT solutions around Contiki and similar hardware platforms [42, 43]. Already in the beginning of the development, several benefits were noticed; new drivers and bug-fixes were released increasing the stability and functionality of the OS. The active community around the combination of OpenMote and Contiki was really helpful when developing the drivers for the I²C and sensor drivers. Example projects for other platforms could be used as references, giving much insight to how the programming for this type of OS worked. It would have been interesting to examine the differences between two operating systems; not only to

test which one has the better performance, but also to compare which one that has the more favourable code structure and development procedure.

3.6 Results

Collecting the data went well and were reasonably straight forward; it was easy to transition between the two different test set-ups and thus making several test scenarios. Assessments were made in the areas of range, response time, connection speed, and power consumption. In each area, the theoretical values were first calculated and then compared to the retrieved measurements; except in the range case, as the required equipment for measuring was not economically justifiable to purchase.

3.6.1 Range

The theoretical range for OpenMote when transmitting at full power in an office environment is only 7m. As measuring the range was not a viable option due to the cost of measuring equipment, only distance estimations from the placement of the nodes when maintaining a stable connection can be used as a reference. Using a map of the office and the position of the nodes the range seems to be around 10m, which would mean that the effective FM of the office is around 16dB using the always-on RDC. The FM changed a bit when using the 8Hz RDC as more packages congested the air and the range dropped to somewhere around 5m; resulting in an effective FM of 23dB. To increase the range of the transceiver, a switch to the 860MHz frequency band would be the most effective solution; with a FM of 23dB, the theoretical range would increase to 14m with the same transceiver properties, and with a FM of 16dB the range would be 31m. Usually, transceivers with a lower frequency output also have a lower power consumption while transmitting. Working in sub-GHz also gives the benefit of less interference as fewer other devices uses those frequencies. Changing to a sub-GHz band would thus decrease the power consumption and increase the range, without changing the functionality of the nodes.

3.6.2 Response time

Initially when measuring the response time the always-on RDC was used and the measured response time was very close to the theoretical value. However, when using the 8Hz RDC protocol the values started to drastically differ from the theory. This behaviour is likely to originate from the way the RDC driver predicts the next time when the target node should be awake. The procedure is called phase optimization; when enabled, the node saves the time when the node was last seen, it then uses this value to predict the next time the node should be awake based on the RDC cycle. However, this prediction is based on the nodes internal clock. As the clock can differ from those of the other nodes, misalignments seem to occur, resulting in misses when trying to reach the target node. Each misalignment increases the time it takes to reach the target node as the node then needs to strobe the package until the target nodes wakes up again. In theory, when sending strobos the target node should wake up and receive the package within one cycle (125ms); however, this is not guaranteed as other transmissions might occupy the air, further increasing the response time. If the phase optimization could be improved to guarantee the alignment between the nodes, the response time should get much closer to the theoretical value; as the time to reach the node would be maximum one cycle and the air would not be as congested by nodes sending strobos.

3.6.3 Connection speed

The connection speed, when using CoAP or any other protocol with perpacket ACK, is directly bound to the response time. IEEE 802.15.4 has a relatively low data-rate, only 250kbps, compared to other solutions, e.g. BLE (1Mbps) and WLAN (>54Mbps). As throughput is based on datarate over a longer period of time, both the overhead and the response time is needed to make a good estimation. CoAP has a very low header size compared to many other communication protocols, but due to the very small frame size, the overhead is still relatively high. As of now, the results clearly show that when a reliable transfer is desired the connection speed of IEEE 802.15.4 and CoAP is only sufficient for data exchanges around 32 bytes. When the nodes use the always-on RDC, the goodput is less than 3KB/s for one hop and is halved for every hop; however, when the 8Hz RDC is enabled, the goodput is reduced to under 0.1KB/s. Using messages without per-packet ACK, thus removing the response time from the equation, would let the nodes transfer real-time audio and maybe even highly compressed video. However, using messages without the per-packet ACK disables the reliable transmission guarantee, and thus it can only be used with data streams where packet loss is acceptable.

3.6.4 Power consumption

Making a rough estimation of the power consumption of the platform was straight forward task and so was measuring the actual consumption. When comparing, the two the values differed by a factor of 30, which was not expected. The reason probably originates from the clock interrupt which is triggered every 8ms. Initially, this interrupt was assumed to be disabled when the system entered the lower power modes, but this was not the case. As the interrupt fires at 125Hz and the time to wake up and go back to LPM is only 272s, the power spikes from these interrupts were not seen on the measuring instruments. As seen in figure 4.9, even the peaks from the listening cycles were hard to record and those lasted for at least 4ms; instead, the power consumption from the clock timer looks like an increased LPM power consumption. At the time this was discovered there was no time to fix it, but doing so should decrease the average power consumption to within the limits, granting the nodes the ability to run on battery power for a year. As no delays from calculation could be observed, the clock speed on MCU could, in all probability, have been reduced to save power on the nodes. However, this reduction would only have affected the consumption when the node was in active mode, which is only a few percent of the total cycle time. The OpenMote chip has a step-down DC-DC converter for this purpose which is switched off in LPM mode to reduce quiescent currents; however, as most of the time is spent in LPM, reducing the input voltage to 2.1V by changing battery type and removing the step-down converter would be preferable as it would reduce the power consumption. These changes could affect the range of the device, but this has to be assessed.

3.6.5 Project execution

Looking at the time plan and the milestones, as seen in Appendix A and B, each milestone matches a task or transition in the time plan. The planning report was not submitted to the examiner until the 6/2-15, which is two weeks behind schedule, exceeding the time planned for milestone M1. The first draft was submitted before deadline, but several revisions were necessary. In retrospect, the litterature study should probably have been planned in parallel with the planning report, as the information from the study helped with the report. Milestone M2 marks the switch from the literature study and selection of

technology to the development phase. This milestone was met and development could begin in the following week. As seen in Appendix C, the development phase have several risks to consider. The only risk encountered in this phase was R4, as one of the hardware platforms was delivered with a broken sensor. However, this malfunction did not affect the time plan as the development could continue regardless of the malfunction. The end of the development phase was defined by milestone M3, approval of prototype, which was completed ahead of schedule granting an early transition into the assessment phase. In the assessment phase, it could be argued that risk R9 was encountered when measuring the power consumption, as the results from those measurements did not properly show the wake-ups from the clock timer. This phase contained milestone M4 and M5, of which only M5 was done in time. The Half-time presentation, milestone M4, was performed on the 8/4-15 in the form of a meeting, where the progress, results and continuation plan were discussed. Also, a half-time version of the report was sent the 17/4-15 and approved by the examiner. Milestone M6, deliver the final prototype, was completed a few days before the set deadline which eliminated risk R11 and gave more time to work on the writing and the presentation. Both of the oral presentations were attended on the 1/6-15 to grant some experience in how the presentation and opposition are carried out, thus now following the time plan. However, there were not many presentations to watch during the planned weeks, as the presentation schedule follow the academic semesters. The presentation for this thesis was not performed until the 3/6-15, thus being two weeks behind schedule. However, it was scheduled on the first available date suggested by the institution. The final version of the report will be submitted to the examiner before the 19/6-15, thus successfully completing milestone M7.

3.7 Discussion

The purpose of the project was to find and examine a communication protocol that could be suitable for IoT applications, by investigating the current hardware, OS, and communication protocols and building a prototype from the selected choices. What can be said about the investigation is that it is difficult to examine all candidates in detail; this means that a rough selection has to be made based on initial knowledge potentially discarding good options. The general feeling is, however, that all of the examined candidates in this project were relevant and added valuable insights to the current technology status. The assessment gave relevant and interesting results that improved the understanding in what IoT can be used for, and what further areas of investigation could be. One of the most interesting areas of further investigation would be the RDC driver, as it directly affects the response time and thus also the connection speed. Even though the power consumption was not in line with the expectations, the reason has been found and can be resolved. Another conclusion is that IoT is not ready for real-time applications as the latency is much higher than expected, for the technologies assessed in this thesis, and also has a high spread. As the latency increases for each subsequent network hop and the minimum observed latency per hop is 11ms, when using the always-on RDC, this type of communication will probably only be used for applications where response time can vary greatly, without affecting the functionality. CoAP as a communication protocol shows a lot of promise when combined with 6LoWPAN and IEEE 802.15.4. It performs well given its simplicity but has one disadvantage: the large overhead which comes from the MAC addressing fields in the IEEE 802.15.4 frame. If this overhead could be reduced from the current 71% to only 30% the goodput would double. A solution would be to use a similar mechanism as BLE where the packet size varies depending on application. Each node also

has computing time left as the MCU is more powerful than needed for the given application; an improvement would be to use a less powerful MCU, like the ARM Cortex-M0+, to reduce the clock speed as suggested in the discussion. When looking at the future-proof aspect the later suggestion is probably the better, as the clock then could be increased if more computing power is needed. In the future, batteries will hopefully be able to store more energy, thus increasing the time between battery changes or reducing the battery size.

4 | 2 SDN: Sentilo [20]

2 SDN: *Sentilo* [20]

Contents

4.1	Introduction	61
4.1.1	Problem Statement	61
4.1.2	Background	61
4.1.3	Purpose (Goal)	61
4.1.4	Limitations	61
4.1.5	Method	61
4.2	Related work	61
4.3	Background	61
4.3.1	Hardware: Zolertia Z1 Motes	62
4.3.1.1	Peripherals ports	62
4.3.1.2	Internal sensors	62
1)	Temperature Sensor	62
2)	Accelerometer	62
4.3.1.3	External Sensors	62
1)	Analog sensors	62
4.3.1.4	Relay actuator	63
4.3.1.5	Distance sensor	63
4.3.2	Operating systems	63
4.3.3	Communication protocol	63
4.3.3.1	Composition	64
4.3.3.2	RIME	64
4.4	Proposed	65
4.4.1	Definitions	65
4.4.2	Sentilo Architecture	65
4.4.2.1	PubSub Server	65
4.4.2.2	Web Catalog Application	66
4.4.2.3	Extensions (Agents)	66
4.4.3	Sentilo structure	67

4.4.4	Sentilo API	67
4.5	Experimentation	67
4.5.1	Sensor Network	68
4.5.1.1	Border Router	68
4.5.1.2	Nodes	68
4.5.2	Network connector	68
4.5.2.1	Application workflow	68
4.5.2.2	Sensor registration	68
4.5.2.3	Sensor data publish	69
4.6	Results	69
4.6.1	Range	69
4.6.2	Response time	69
4.6.3	Connection speed	69
4.6.4	Power consumption	69
4.7	Discussion	69
4.7.1	Future lines of work	69

Abstract

The exponential usage of messaging services for communication raises many questions in privacy fields. Privacy issues in such services strongly depend on the graph-theoretical properties of users' interactions representing the real friendships between users. One of the most important issues of privacy is that users may disclose information of other users beyond the scope of the interaction, without realizing that such information could be aggregated to reveal sensitive information. Determining vulnerable interactions from non-vulnerable ones is difficult due to the lack of awareness mechanisms.

To address this problem, we analyze the topological relationships with the level of trust between users to notify each of them about their vulnerable social interactions. Particularly, we analyze the impact of trusting vulnerable friends in infecting other users' privacy concerns by modeling a new vulnerability contagion process. Simulation results show that over-trusting vulnerable users speeds the vulnerability diffusion process through the network. Furthermore, vulnerable users with high reputation level lead to a high convergence level of infection, this means that the vulnerability contagion process infects the biggest number of users when vulnerable users get a high level of trust from their interlocutors. This work contributes to the development of privacy awareness framework that can alert users of the potential private information leakages in their communications.

4.1 Introduction

4.1.1 Problem Statement

From the start of the computer networks, to the mobile applications nowadays, the amount of information shared has been constantly increasing. We have all kind of devices, from the big servers in datacenters, the TVs at home, mobile phones, car sensors, ... The Internet Of Things (IOT) refers to the idea of connecting all the things to the Internet. By things, it refers to any ordinary object that can be useful getting information. These things should be connected by an embedded device, capable to connect to the Internet in one side, and get information from the thing on the other.

4.1.2 Background

4.1.3 Purpose (Goal)

The first objective of this thesis is to document the capabilities of the Z1 motes and the Contiki OS for the IOT, by building applications to gather data from sensors and the network capabilities both from the motes and the OS. Secondly, to build an application using the Z1 motes, and the Contiki OS, using COAP(Constrained Application Protocol) and 6LoWPan(IPv6 over Low power Wireless Personal Area Networks) to retrieve information from the motes, and connect them to Sentilo, an open source sensor and actuator platform.

4.1.4 Limitations

There has been an increased research and development for the Smart Cities. The smart cities objective is to gather information from the city, to enhance quality and performance of urban services, to reduce costs and resource consumption, and to engage more effectively and actively with its citizens. This project is intended to approach two goals, to be used as a starting point for anyone who wants to use the Contiki OS with the Z1 motes, and to build a simple application for the Smart Cities, to collect data from sensors, and sending it to an information center.

4.1.5 Method

This document is divided in two parts. First, the description of the main tools used to create the experimental environments. A description of the Contiki OS, the Z1 motes, and the protocols used to communicate, IEEE 802.14.5, 6LowPAN and CoAP. The a brief description of the sensor data collector Sentilo Secondly, a description of the environment setup, and an explanation of how it works. Finally, conclusions and future work is presented.

4.2 Related work

4.3 Background

4.3.1 Hardware: Zolertia Z1 Motes

The Z1 is a low power wireless module compliant with IEEE 802.15.4 and Zigbee protocols intended to be used for Wireless Sensor Networks. This mote has support for Tiny OS, Contiki OS, OpenWSN and RIOT. The MCU architecture is based upon the MSP430 and the radio transceiver on CC2420 architecture, both from Texas Instruments.

4.3.1.1 Peripherals ports

North Port

East Port

South Port

West Port A Z1 mote has 2 internal sensors, and using the external ports, can be connected to a variety of external sensors. The main issue about collecting data with Contiki is the lack of support for floating point numbers in the stdio library, because of the large amount of code it requires. It has floating point numbers, but those are only useful for internal operations. If a program needs to send the decimal data to an external source, has to use integers in the stdio functions, to write into the buffers.

4.3.1.2 Internal sensors

1) Temperature Sensor The internal temperature sensor in the Z1 mote is the tmp102 sensor from Texas Instruments. This sensor is integrated with the z1 motes using the I2C interface. It can read the temperature range of -40°C to +125°C. The Contiki OS has its own library of functions that can read the sensor data, located in "platform/z1/dev/tmp102.h". To use it in a program, it has to include the library dev/tmp102.h

2) Accelerometer The internal accelerometer in the Z1 motes is the adxl345 from Analog Devices Inc. This sensor is integrated with the z1 motes using the I2C interface. The Contiki OS has its own library of functions that can read the sensor data, located in "platform/z1/dev/adxl345.h". To use it in a program, it has to include the library dev/adxl345.h. The sensor has 8 different interrupts to enable and 2 pins for mapping the interrupts.

4.3.1.3 External Sensors

The Z1 motes have several ways to connect sensors. In the next chapters, there are some examples of sensors, and how to read the data.

1) Analog sensors To read the analog sensors, there is a Contiki library in platform/z1/dev/z1phidgets.h. This library reads the values of 4 of the pins of the north ports, and returns a 16 bit register, representing the value. It uses a 12bits A/D converter, so the min value is 0 and the max in 4095.

1.1) Precision Light Sensor The precision light sensor used as an example is the Phidget P/N 1127 sensor. This sensor is an analog sensor that measures light intensities of up to 1000 lux. It is a non-radiometric sensor. The output value does not depend on the input voltage, but the input voltage will limit the maximum measurement value. The sensor can be connected to the north port of the Z1 motes, into the 3V port or the 5V port.

In Fig.60, to read the value of the sensor, the phidgets library from Contiki is used. After a raw read, the value is transformed to lux, knowing the maximum value of the A/D converter is 4095, and the maximum value the sensor can give is 1000 lux. (In this case is connected to 5V)

1.2) Force Sensor The force sensor used used as an example is the Phidget P/N 1106 sensor. This force sensor can be used as a button for human input or to sense the presence of a small object. It is a radiometric sensor. The output value depends on the input voltage. It measures the same force value with 3V or 5V.

In Fig. 64, to read the value of the sensor, the phidgets library from Contiki is used. Once the raw value is read, it is transformed it to Newtons, knowing the maximum value of the A/D converter is 4095, and the maximum value the sensor can give is 39.2 Newtons.

4.3.1.4 Relay actuator

The relay used as an example is the Electronic brick 5V Relay form seeedstudio.

This actuator, works as a switch, when a signal is sent through the signal pin. It has a library for the Z1 motes in platform/z1/dev/relay-phidget.h".

This library conflicts with the phidgets library, because it turns the selected pin from the north port as an output, and the phidgets functions as an input. In this configuration, the switch is powered with 5V supplied by the Z1 in the ON port, and with ground in the OFF port. It toggles the led on and off, each time the signal is triggered.

In Fig.68 example, the main loop waits for a specified time, and then toggles the relay.

4.3.1.5 Distance sensor

The distance sensor used as an example is the SEN-12784 from SparkFun. It has an VL6180 digital sensor integrated, that can read light and distance. It uses the an I2C interface to extract the values from the sensor registers.

Contiki has a I2C interface library adapter for the Z1 motes in platform/z1/dev/i2cmater.h To use it in a program, it has to include the library dev/i2cmaster.h

The function in Fig.71, shows how to read the distance from the device. It calculates the distance by sending a pulse of light, and retrieving it back, the doing an internal calculation with the difference between the power of the signal sent and the received. Between the activation and the collection of the value, there is some time waiting for the light to travel. The functions in Fig.71 and Fig.73, show how to set and get a register from the sensor, using the I2C interface.

4.3.2 Operating systems

4.3.3 Communication protocol

Wireless sensor networks combines 3 concepts together: sensor + CPU + radio. However, combining sensors, radio and CPU's together requires an extensive understanding of the hardware components as well as modern networking technologies to connect the devices.

Each node needs to have the necessary tools to send data over the radio channel, while meeting the requirements of size, cost and power consumption. The research and development of this kind of devices, has been increased over the last years. There are a number of operating systems focused on providing communications stacks and at the same time focused on saving power. On the other hand, the devices integrating a CPU and a radio transceiver have become more available and efficient.

4.3.3.1 Composition

There are four main types of nodes in a WSN structure. Sensor nodes: These nodes are in charge of collecting data, and sending it to the network. These nodes have 2 parts, the sensors board and the mote. The sensor board, contains the sensor to acquire data (light, temperature, humidity,...) The mote integrates the CPU and the radio transceiver. Route nodes: Nodes with the only purpose of making possible the link between the sensor nodes and the rest of the network. They work as a repeater of the radio signal, and implement routing tasks. Server station: It is the concentrator of the data sent over the network. It is a node itself, or a node attached to a more powerful machine, able to manage lots of data. Gateway: Connects the WSN to an external network, if needed.

The transmission of sensor's data is done by all the nodes of the network. Each data packet, is sent to the server station hop by hop. Reducing the transmission power in the nodes, may reduce the power consumption on it, but it may require a larger number of hops to arrive to the server station

4.3.3.2 RIME

RIME is a communication stack designed for Contiki. It provides a hierarchical set of wireless network protocols. This protocol stack can send data over the standard IEEE 802.14.5 with very few transmissions and less overhead than an IP based protocol, saving energy in the devices involved in the connection. Implementing a complex protocol (say the multi-hop mesh routing) is split into several parts, where the more complex modules make use of the simpler ones.

These are some of the different modules of Rime:

abc: the anonymous broadcast, it just sends a packet via the radio driver, receives all packets from the radio driver and passes them to the upper layer; **broadcast:** the identified broadcast, it adds the sender address to the outgoing packet and passes it to the abc module;

unicast: this module adds a destination address to the passed packets to the broadcast block. On the receiver side, if the packet's destination address doesn't match the node's address, the packet is discarded;

stunicast: the stubborn unicast, when asked to send a packet to a node, it sends it repeatedly with a given time period until asked to stop. This module is usually not used as is, but is used by the next one.

runicast: the reliable unicast, it sends a packet using the stunicast module waiting for an acknowledgement packet. When it is received it stops the continuous transmission of the packet. A maximum retransmission number must be specified, in order to avoid infinite sending.

polite and ipolite: these two modules are almost identical, when a packet has to be sent in a given time frame, the module waits for half of the time, checking if it has received the same packet it is about to send. If it has, the packet is not sent, otherwise it sends the packet. This is useful for flooding techniques to avoid unnecessary retransmissions.

multipath: this module requires a route table function, and when it is about to send a packet it asks the route table for the next hop and sends the packet to it using unicast. When it receives a packet, if the node is the destination then the packet is passed to the upper layer, otherwise it asks again the route table for the next hop and relays the packet to it.

4.4 Proposed

Sentilo is an open source platform to store sensor and actuators information. This platform is designed for the smart cities environment, to be used as a sensor data server that stores the data from different providers and different components within the providers.

4.4.1 Definitions

Provider: A Sentilo account in the server. It stores the published data, and sends the data to his subscribers.

Publisher: A device that sends data to the server. It publish the data into a provider account.

Subscriber: A device that receives data. It is subscribed to a certain data from a provider

Worker: A threat in the server that executes a programed task

Redis: A in-memory data structure store. It is used as a Publisher/Subscriber implementation to store the data in the memory of the server.

MongoDB: A database that stores the data as 'documents'. A 'document' is a JSON object.

4.4.2 Sentilo Architecture

The platform has 3 distinct parts:

PubSub Server (Core)

Web Catalog Application (A web interface to check the information of the PubSub Server)

Extensions (Also called Agents, they extend the capabilities of the PubSub Server)

The core platform, listens and responds to requests specified in the API. By default, it listens the TCP port 8081

- For a publisher, it registers the data sent, in one of the platform items.

- For subscribers, it responds with a JSON with the requested data of an item.

The web catalog, is a web interface to manage and see the information on the PubSub Server. It listens the TCP port 8080. The platform supports some extensions in order to extend the base functionalities such as alerts or data storage.

4.4.2.1 PubSub Server

The Core of the platform is a running process, that listens to the requests and creates workers (Threads) to do the tasks. There are 2 requesters:

Publishers: Send data from sensors, and alerts.

Subscribers: First, they request a subscription. Then waits for the data they are subscribed is sent. The platform is separated in 2 different layers: Transport and Service. The

transport layer manages the incoming requests (as published data, data requests or subscription requests) and generates a queue with tasks containing the information of the request. Then, a limited pool of workers handles the requests, every time each finishes the previous task.

When a client sends an Http request to the platform, the process is: (Fig. 76). 1.The server accepts the request. 2.Queues the request on the list of pending requests. 3.When a Worker is available, a pending task is assigned to it for processing(removeing it from the queue) (a) delegates the request to an element of the service layer (b) constructs the HTTP response from the information received. 4.Sends the response to the client's request

The service layer manages the workers information and processes it and registers the data or delivers the data depending on the request. (Fig. 77)

1.The Worker delegates the request to the associated handler depending on the type of request (data, order, alarm, ...)

2.The following validations are performed on each request: (a) Integrity of credential: checks the received token sent in the header using the internal database in memory containing all active credentials in the system. (b) Authorization to carry out the request: validate that the requested action can be done according to the permission database.

3.Stores the data in Redis (in memory), and depending on the type of data (a) Publish the data through publish mechanism (b) Register of the subscription in the ListenerMessageContainer (A list of all subscribers) and into Redis as a subscriber.

4. If any new data is received, Redis publish the data to the subscribers, otherwise this step is skipped.

5. The container notifies the event to each subscriber associated with it by sending an HTTP Request to them.

4.4.2.2 Web Catalog Application

The catalog application platform is a web application that uses MongoDB as data storage database. The Web App has 2 parts:

- A public console for displaying public data of components and sensors and their data

- A secured part for resources administration: providers, client apps, sensors, components, alerts, permissions, ...

It is fully integrated with the Publish/Subscribe platform for data synchronization:

- Permission and authentication data

- Register statistical data and the latest data received for showing it in different graphs of the Web application.

4.4.2.3 Extensions (Agents)

The extensions of Sentilo add functionalities to the Core application. The extensions are subscribed to the Redis module for all the incoming notifications.

When Redis receives a publication of data, sends a message to all subscribers, including all the agents. The agent gets the data, and carries out his task. Currently there are 3 Sentilo agents:

- Relational database agent

- Stores all the incoming data in a external database Alarm agent

- Manages the internal alerts defined into the Web Catalog and published an alert if the condition is met.

- Location updater agent

Is responsible of updating automatically the component location according to the location of the published observations.

4.4.3 Sentilo structure

The platform has 5 main items:

- Component
- Sensor
- Alert
- Alarm
- Order

A component is the item where a set of sensors is attached. A sensor is a representation of a physical sensor, it is attached to a component. The data published is sent for a specific sensor. An alert is a trigger registered in Sentilo when an event happens. There are 2 types of alerts: internal and external. The internal alerts are related to specific sensors and its logic is defined using basic math rules or configuring an inactivity time. The external alerts are defined by third party entities, which will be the responsible of calculating their logic and throw the related alarms when applies. An alarm is the message sent to the subscribers of an alert when it is triggered. Must be attached to an alert. An order is a message registered for a specific sensor or component. It is received by the subscribers of the sensor or component orders.

4.4.4 Sentilo API

The Application Programming Interface (API) define a set of commands, functions and protocols that must be followed by who wants to interact with the platform from external systems, like sensors/actuators or applications. The requests are HTTP requests with 3 fields in the header:

- The Request Method: GET, POST, PUT
- IDENTITYKEY: The authentication token
- Content-Type: application/json

The platform has 3 operations for publishers:

- Retrieve data: Using the GET method, any kind of data can be consulted, the response is in JSON format

- Register data: Using the POST method, can be registered components, sensors alerts, alarms or orders.

- Update data: Using the PUT method, components, sensors alerts, alarms and orders data can be updated. Also sensor data can be published. It also has 3 kind of subscriptions:

- To sensor data - To orders - To alerts

All the documentation of the Application Programming Interface can be found in:

- <http://www.sentilo.io/xwiki/bin/view/APIDocs/WebHome>

4.5 Experimentation

The objective of this scenario is to connect a Wireless Sensor Network to a running Sentilo server. There are 2 sides of the network, with the border router in the middle of both. The WSN uses CoAP to extract the sensors information, and the sensor data. The Sentilo server uses HTTP requests, with JSON objects. The JSON (JavaScript Object Notation) is a text format transmit data objects consisting of attributevalue pairs. It is one of most widely used by programming languages to send data over HTTP.

4.5.1 Sensor Network

The wireless sensor network is composed by Z1 motes connected by a border-router.

4.5.1.1 Border Router

The Border Router manages the RPL (Routing Protocol for Low-Power and Lossy Networks), and is connected to a computer using Tunslip, a tool used to bridge IP traffic between 2 devices, over the serial line. Tunslip creates a virtual network interface (tun) on the host side and uses SLIP (serial line internet protocol) to encapsulate and pass IP traffic to and from the other side of the serial line.

4.5.1.2 Nodes

Each of the motes has a CoAP server running, and has a resource for each sensor attached to the mote. In this environment 2 Sentilo items will be used:

Component: The hardware where a sensor is attached.

Sensor: A physical sensor. It must be attached to a component For the Sentilo server, each component, sensor, and alert must have a unique id. In this setup, each mote is a component in the server, the mote id is used for the unique id in sentilo. For this example, the mote 3 will have the id MOTE03. Each sensor has his unique id too, using the component id and the type of sensor. In this setup the temperature sensor of the mote 3 will have the id MOTE03TMP. Every sensor has a CoAP resource defined in the mote. A location resource is defined to set the mote location

4.5.2 Network connector

In the computer connected with the border-router, there's a Java application that pulls the information in the WSN using CoAP, and communicates with the Sentilo server to register the sensor and send the data. A provider must be registered manually in Sentilo in order to get the authentication token. For every request sent, the authorization token is checked.

4.5.2.1 Application workflow

The Java application that connects the 2 networks, follows 5 steps: 1. Searches for all the Motes of the specified network in the border router, by sending an HTTP GET to the border router. It responds with an XML with the information of all the motes.

Discovers all the sensors in each Mote, by sending a CoAP discover to each mote.

Gets the information of each sensor, by sending a CoAP GET to the resources on the mote.

Registers each sensor in Sentilo, by sending a HTTP POST to the server with the information of the sensor.

Starts collecting data from the sensors, and registers it in Sentilo, by sending a CoAP observe to each Mote resource, and for each observation, sends a HTTP PUT with the data to Sentilo.

4.5.2.2 Sensor registration

Once the application has a list of all the motes and the sensors of each one, sends a GET request every mote for each one of the sensors resources, to get the information of the resource. The sensor resource has defined the information needed to register.

Once the information of the sensor is gathered, it creates a JSON Object to register the sensor into the Sentilo server via the API.

4.5.2.3 Sensor data publish

The application starts an OBSERVE on the mote for each sensor resource. At this point, the application starts to listen for messages from the CoAP resource. The sensor periodic resource sends information of the sensor data periodically. The period of observation is defined in the mote. In every observation, the data is sent to Sentilo in a JSON Object via the API.

The parameters sent in the JSON to the server are:

4.6 Results

4.6.1 Range

4.6.2 Response time

4.6.3 Connection speed

4.6.4 Power consumption

4.7 Discussion

The Contiki OS, collects all the technologies needed for the development of centralized data collectors, for the sensors. This platform combined with Sentilo, creates a real application platform, to be able to deploy in several possible real environments. The main advantages of Contiki, are how easy is to create code, and generate concurrent scenarios inside the same mote, being able to have a web server at the same time a root node of a WSN is running, without complexity. At the same time , the application level library as COAP, with the complete examples of this libraries, makes this system a powerful and versatile tool. A disadvantage of this platform, is the lack of documentation and examples, outside the inner code. There's a lot of time and test to make, for a more complex application. Secondly, the Sentilo platform, is an easy to install, use and program applications with. It has a wide set of options and tools, that need to be understand carefully for a rich application that uses all the functionalities properly. The combinations of both, makes a good, simple and potentially improvable scenario, for centralize data collection.

4.7.1 Future lines of work

There are some future lines of work in this experimental environment:

1. Test the CoAP server in the new release of Contiki. Contiki 3.0 A new release of Contiki was released in September 2015, with some changes and improvements overall, specially with CoAP. The new release supports CoAP 18.
2. A Java connector with a dynamic network. The Java connector finds the motes in a stable WSN, if a node is missing or replaced, it needs a manual interaction to find all the motes again, by erasing all the network, and start to find all the motes again. Besides, the protocol handling the routes, is IP and the protocol handling the links is RPL. The IP routes in the border router expire every certain time, that means that if a mote is missing,

a route is still present for a certain time, even if the RPL is aware of the missing mote. As a possible solution, there are repairing route methods in CoAP that are used to repair the broken links between nodes.

5 | 3 MEC

3 MEC

Contents

5.1	Introduction	72
5.1.1	Problem Statement	72
5.1.2	Background	72
5.1.3	Purpose (Goal)	72
5.1.4	Limitations	72
5.1.5	Method	72
5.2	Related work	72
5.3	Background	72
5.3.1	Hardware	72
5.3.2	Operating system	72
5.3.3	Communication protocol	73
5.3.4	Workspace and tools	73
5.4	Proposed	73
5.5	Experimentation	73
5.6	Results	73
5.6.1	Range	75
5.6.2	Response time	75
5.6.3	Connection speed	75
5.6.4	Power consumption	75
5.7	Discussion	75

Abstract

The exponential usage of messaging services for communication raises many questions in privacy fields. Privacy issues in such services strongly depend on the graph-theoretical properties of users' interactions representing the real friendships between users. One of the most important issues of privacy is that users may disclose information of

other users beyond the scope of the interaction, without realizing that such information could be aggregated to reveal sensitive information. Determining vulnerable interactions from non-vulnerable ones is difficult due to the lack of awareness mechanisms.

To address this problem, we analyze the topological relationships with the level of trust between users to notify each of them about their vulnerable social interactions. Particularly, we analyze the impact of trusting vulnerable friends in infecting other users' privacy concerns by modeling a new vulnerability contagion process. Simulation results show that over-trusting vulnerable users speeds the vulnerability diffusion process through the network. Furthermore, vulnerable users with high reputation level lead to a high convergence level of infection, this means that the vulnerability contagion process infects the biggest number of users when vulnerable users get a high level of trust from their interlocutors. This work contributes to the development of privacy awareness framework that can alert users of the potential private information leakages in their communications.

The article is organized as follows. Section 5.2 elucidates summary of related works, In section 5.4, we propose our ... to Section 5.5 evaluate the performance of our ... in terms of packet delivery ratio, throughput, and power consumption. Section 5.7 concludes the article and gives some ideas for future work.

5.1 Introduction

5.1.1 Problem Statement

5.1.2 Background

5.1.3 Purpose (Goal)

5.1.4 Limitations

5.1.5 Method

5.2 Related work

5.3 Background

Selection of technology

5.3.1 Hardware

5.3.2 Operating system

5.3.3 Communication protocol

5.3.4 Workspace and tools

5.4 Proposed

Naïve modes	Instantaneous Hist. average Clustering
Parametric models	Rarely used Traffic Models Time Series Linear regression ARIMA Kalman filtering ATHENA SETAR Gaussian Maximum Likelihood
Non-Parametric models	k-Nearest Neighbor Locally Weighted Regression Fuzzy Logic Bayes Network Neural Network Include temporal/spatial patterns

Table 5.1: Taxonomy of prediction models [[short_2007](#)]

- ➡ Network selection
 - ➡ MADM
 - * Ranking methods
 - * Ranking & weighted methods
 - ➡ Game theory
 - * Users vs users
 - * Users vs networks
 - * Networks vs network
 - ➡ Fuzzy logic
 - * as a score method
 - * another theory
 - ➡ Utility function
 - * 1
 - * 2

5.5 Experimentation

jhjh

5.6 Results

Signal-to-interference & noise ratio (**SINR**) Signal-to-Interference Ratio (**SIR**) **BR DR BW BER PER PRR** Packet delivery ratio (**PDR**) **SNR** Packet loss rate (**PLR**) Round trip time (**RTT**) Transmission Energy (**Tx**) Payload size (**PS**) Traffic congestion (**TC**) Duty cycle (**DC**) **SR** Sleep time (**SL**) Jitter (**Jit**) Co-channel Interference (**CCI**) Time on Air (**ToA**) **PL** Mobility (**Mob**) Throughput (**Th**) Service Cost (**SC**)

Layer	Maximize (Reward)	Minimize (Cost)
Application	Sec security Ergonomic	SC Service Cost
Network	BW Bandwidth available PRR Packet Reception Ratio Th Throughput Range Network coverage Availability Availability DR Data rate PDR Packet delivery ratio	PS Payload size Jit Jitter TC Traffic congestion PLR Packet loss rate RTT Round time trip \mathcal{O}_{time} Time Complexity \mathcal{O}_{space} Space Complexity
Radio	SNR Signal-to-noise ratio DC Duty cycle SR Symbol rate Mob Mobility BR Bit rate SINR signal-to-interference & noise ratio SIR Signal-to-Interference Ratio	BER Bit error rate SL Sleep time Tx Transmission Energy CCI Co-channel Interference PL Path loss ToA Time on Air

Table 5.2: Network selection inputs and classification of parameters [21] + QoS parameters [22] [23]

[24]

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (p_i - r_i)^2 \quad (5.1)$$

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (p_i - r_i)^2} \quad (5.2)$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |p_i - r_i| \quad (5.3)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (5.4)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (5.5)$$

$$\text{F1_Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{precision} + \text{recall}} \quad (5.6)$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (5.7)$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (5.8)$$

$$\text{ROC} = (\text{TPR}, \text{FPR}) \quad (5.9)$$

$$\text{Novelty} = \sum_{i \in L} \frac{\log_2 P_i}{n} \text{ where } P_i = \frac{n - rank_i}{n - 1} \quad (5.10)$$

$$\text{Serendipity} = \frac{1}{n} \sum_{i \in n} \max(P_{\text{user}} - P_U, 0) \times rel_i \quad (5.11)$$

$$\text{diversity} = \frac{a}{c} \sum_{i=1}^c \frac{1}{n} \sum_{j=1}^n i_j \quad (5.12)$$

$$\text{Coverage} = 100 \times \frac{u}{U} \quad (5.13)$$

$$\text{Stability} = \frac{1}{P_2} \sum_{i \in P_2} |P_{2,i} - P_{1,i}| \quad (5.14)$$

5.6.1 Range

5.6.2 Response time

5.6.3 Connection speed

5.6.4 Power consumption

5.7 Discussion

6 | Short paper

Short paper

Contents

6.1	Introduction	78
6.2	Related work	78
6.3	Background	78
6.4	Prototype Development	78
6.5	Experimentation	78
6.5.1	Range	78
6.5.2	Response time	78
6.5.3	Connection speed	78
6.5.4	Power consumption	78
6.6	Results	78
6.6.1	Range	78
6.6.2	Response time	78
6.6.3	Connection speed	78
6.6.4	Power consumption	78
6.7	Discussion	78

Abstract

The exponential usage of messaging services for communication raises many questions in privacy fields. Privacy issues in messaging services strongly depend on the graph-theoretical properties of users' interactions representing the real friendships between users. One of the most important issues of privacy is that users may disclose information of other users beyond the scope of the interaction, without realizing that such information could be aggregated to reveal sensitive information. Our. Our goal is to. Determining vulnerable interactions from non-vulnerable ones is difficult due to the lack of awareness mechanisms. To address this problem, we analyze the topological relationships with the

level of trust between users to notify each of them about their vulnerable social interactions. Particularly, we analyze the impact of trusting vulnerable friends in infecting other users' privacy concerns by modeling a new vulnerability contagion process. Simulation results show that over-trusting vulnerable users speeds the vulnerability diffusion process through the network. Furthermore, vulnerable users with high reputation level lead to a high convergence level of infection, this means that the vulnerability contagion process infects the biggest number of users when vulnerable users get a high level of trust from their interlocutors. We validate our approach by using real messages from Enron dataset.

6.1 Introduction

6.2 Related work

6.3 Background

6.4 Prototype Development

6.5 Experimentation

6.5.1 Range

6.5.2 Response time

6.5.3 Connection speed

6.5.4 Power consumption

6.6 Results

6.6.1 Range

6.6.2 Response time

6.6.3 Connection speed

6.6.4 Power consumption

6.7 Discussion

7 | Long paper

Long paper

Contents

7.1	Introduction	80
7.2	Related work	81
7.2.1	LoRa Overview	81
7.2.1.1	Bandwidth (BW):	81
7.2.1.2	Carrier Frequency (CF):	81
7.2.1.3	Coding Rate (CR):	81
7.2.1.4	Spreading Factor (SF):	82
7.2.2	From Experimental Measurements to Simulations	82
7.2.3	LoRa Evaluation and Limits	82
7.2.4	LoRa Network Deployment Strategies	83
7.3	Background	83
7.4	Approach	83
7.4.1	Spectrum	84
7.4.2	LoRa Modulation	84
7.4.3	Spreading Factors Orthogonality	84
7.4.4	LoRa SX1276 and SX1301 Transceivers	84
7.4.5	MAC and Application Layers	85
7.4.6	Deployment Scenarios	85
7.5	Experimentation	86
7.6	Results exploitation	87
7.6.1	Deployment Scenarios and Assumptions	87
7.6.2	Homogeneous Deployments Scenarios	87
7.6.3	Heterogeneous vs. Homogeneous Deployment Scenarios	87
7.6.4	Application Period and Packet Duration vs. ERC Regulations	88
7.7	Conclusion	88

Abstract

The need of Low power wide area network (LPWAN) networks increased quickly these last years. The main factor is that IoT devices require low power consumption to transmit data in a wide area. Lora, Sigfox and NB-IoT are the most known technologies that satisfy these requirements. Applications like smart building and smart environment are one of hundreds use cases that need to be deployed with such technologies. Unlike Sigfox and NB-IoT, Lora is more open for academic research because the specification that governs how the network is managed is relatively open. LoRa is a wireless modulation technique that uses Chirp Spread Spectrum (CSS) in combination with Pulse-Position Modulation (PPM). The transmission could be configured with 4 parameters: Spreading factor (SF), Transmission power (Tx), Coding rate (CR) and Bandwidth (Bw), to achieve better performance.

The main LPWAN research directions are about large scale networks to support massive number of devices, interference issues, link optimization and adaptability. Thus heterogeneous network deployments and Spreading Factor (SF) allocation strategies need to be studied. In this paper, we investigate the performance of homogeneous networks (i.e. when all the nodes select the same LoRa configuration) and heterogeneous networks (i.e. when each node selects its LoRa configuration according to its link budget or their needs) for large scale deployments (up to 10000 nodes per gateway). For that purpose we have developed a LoRa Module, based on improved WSNet simulator, including a spectrum usage abstraction, the co-channel rejection due to the quasi-orthogonality of SFs and the gateway capture effect. Simulation results show the performance comparison in terms of reliability, network capacity and power consumption for homogeneous and heterogeneous deployments as a function of the number of nodes and the traffic intensity. The comparison shows the benefits of the heterogeneous deployment where each node selects its configuration according to its link budget.

7.1 Introduction

Over the past few years, new approaches called Low Power Wide Area (LPWA) networking technologies have emerged.

These technologies became a new alternative to current generations of cellular networks (2G, 3G, and 4G), while covering large areas (typical range of 10 km).

Several papers have reviewed the different LPWA technologies. For example, Raza et al. [1] surveyed standardization activities (IEEE, IETF, 3GPP, ETSI), as well as industrial ones around LPWA technologies (LoRa alliance, Weightless-SIG, Dash7 alliance). They identified potential research directions to address limitations and challenges of LPWA technologies such as a massive number of devices, link optimization, and adaptability.

After giving an overview of LPWA and cellular technologies for IoT [2], the authors discussed the capabilities and limitations of LoRaWAN. Nevertheless, the potential of an adaptive LoRa solution in terms of spreading factor, bandwidth, transmission power, and topology is still not well studied or exploited. Thus, new protocols and strategies are required to improve LoRa scalability.

The first step is to understand heterogeneous network deployments when devices use different spreading factor. In this paper, we investigate homogeneous (i.e. when all the nodes select the same LoRa configuration) and heterogeneous deployments (i.e. when each node selects its LoRa configuration according to its link budget or their needs) for a large number of devices (up to 10000 nodes per gateway). In order to evaluate performance, we have developed an accurate model of the PHY/MAC LoRa based on the extended WSNet simulator.

The LoRa model takes into account spectrum usage, co-channel rejection due to quasi-orthogonality of the LoRa spread spectrum modulation, and the gateway capture effect. The simulation results give an insight on reliability, network capacity, and energy consumption for homogeneous and heterogeneous deployments as a function of the number of nodes and traffic intensity.

The article is organized as follows. Section II gives a short overview of the LoRa technology and provides the state of the art on LoRa experimental measurements and simulations, LoRa limitations, and network deployment strategies as well as motivations. Section III presents the developed LoRa WSNet based simulator and the deployment scenarios. Section IV analyzes performance in terms of packet delivery ratio, throughput, and power consumption. Section V concludes the article and gives some ideas for future work.

7.2 Related work

7.2.1 LoRa Overview

In this section, we give a short overview of the LoRa physical layer parameters. LoRa specification [3], [4] and technical documents [5], [6], [7], [8] contain more detailed information.

7.2.1.1 Bandwidth (BW):

is the range of frequencies available for transmission. Larger BW increases the data rate but decreases sensitivity. Typically, BWs are 125 kHz, 250 kHz, and 500 kHz. However, Semtech SX1276 offers BW configurations from 7.8 kHz to 500 kHz.

7.2.1.2 Carrier Frequency (CF):

is the central frequency in a band. For our study, we consider the 868 MHz band.

7.2.1.3 Coding Rate (CR):

characterizes resilience to transmission errors. Higher CRs result in better robustness, but increases the air time. LoRa supports CR of 4/5, 4/6, 4/7, and 4/8.

7.2.1.4 Spreading Factor (SF):

is the ratio between the chip rate and the symbol rate. For a given SF, there are 2 SF chips per symbol. SF can be selected between 6 and 12, where SF12 presents the highest sensitivity and the longest range, but the lowest data rate.

7.2.2 From Experimental Measurements to Simulations

Some authors deployed LoRa networks and experimentally studied its performance [9] [10] [11] [12] [13]. The measurements were done in city centers, tactical troop tracking, and sailing monitoring systems. Nevertheless, experimental results in real life networks are not reproducible and MAC layer optimization is difficult. Blenn et al. [14] performed simulations based on traces from experiments and analyzed results based on real life and large scale measurements from The Things Network but their simulations are limited to the deployed scenario. To and Duda [15] presented LoRa simulations in NS-3 validated in testbed experiments. They considered the capture effect and showed the reduction of the packet drop rate due to collisions with a CSMA approach. In a system level simulator, Haxhibeqiri et al. [16] studied the scalability for LoRaWAN deployments in terms of the number of nodes per gateway. Simulations are performed for a duty cycle of 1% but they are limited to 1000 nodes. We developed a LoRa simulator to compare the performance in different deployment scenarios for large scale networks based on an accurate model of the LoRa PHY/MAC layers. We simulate several deployment scenarios varying traffic intensity and the number of nodes.

7.2.3 LoRa Evaluation and Limits

Several authors evaluated performance and limits of LoRa networks. Reynders et al. [17] evaluated Chirp Spread Spectrum (CSS) and ultra-narrow-band networks. They proposed a heuristic equation that gives Bit Error Rate (BER) for a CSS modulation as a function of SF and Signal to Noise Ratio (SNR). Cattani et al. [18] evaluated the impact of the LoRa physical layer settings on the data rate and energy efficiency. They evaluated the impact of environmental factors such as temperature on the LoRa network performance and showed that high temperatures degrade the Packet Delivery Ratio (PDR) and Received Signal Strength (RSS). Goursaud et al. [19] studied the performance of the CSS modulation. They showed the possibility of interference between different SFs and evaluated co-channel rejection for all combinations of SFs. Feltrin et al. [20] discussed the role of LoRaWAN for IoT and showed its application to many use cases. They considered the effect of non perfect orthogonality of SFs for a link level analysis. Petajajarvi et al. [21] analyzed the scalability of a LoRaWAN wide area network and showed its good coverage (e.g. until 30km on water for SF12 and transmit power of 14 dBm). They also showed the maximum throughput for different duty cycles per node per channel. Mikhaylov et al. [22] discussed LoRa performance under European frequency regulations. They studied the performance metrics of a single end device, then the spatial distribution of several end devices. They showed LoRa strengths (large coverage and good scalability for low uplink traffic) and weaknesses (low reliability, delays, and poor performance of down-link traffic). Bor et al. [23] presented a capability and performance analysis of a LoRa transceiver and proposed LoRaBlink protocol for link-level parameter adaptation. Nunez et al. [24] analytically showed the potential gain of adaptive LoRa solutions that choose suitable radio parameters (i.e., spreading factor, bandwidth, and transmission power) to different deployment topologies (i.e., star and mesh). These studies provide a first view

of LoRa performance and its limitations. As a conclusion we need to take into account the capture effect and imperfect orthogonality of SFs. We contribute with an accurate LoRa simulation model considering the co-channel SF interference and the gateway capture effect, allowing accurate performance analysis in large scale simulations for different deployment scenarios. Our study extends the previous evaluations of LoRa limits with the evaluation of reliability, network throughput, and power consumption from sparse to massive access deployment scenarios.

7.2.4 LoRa Network Deployment Strategies

Some authors studied LoRa network deployments and SF allocation strategies. Bor et al. [25] studied LoRa transceiver capabilities and the limit supported by LoRa system. They showed that LoRa networks can scale if they use dynamic selection of transmission parameters. Georgiou et al. [26] investigated the effects of interference in a network with a single gateway. They studied two link-outage conditions, one based on SNR and the other one based on co-SF interference. They showed, as expected, that performance decreases when the number of nodes increases and highlighted the interest of studying spatially heterogeneous deployments. Croce et al. [27] showed the effect of the quasi-orthogonality of SFs and found that overlapped packet transmissions with different SFs may suffer from losses. They validated the findings by experiments and proposed SIR thresholds for all combinations of SFs. They remarked that LoRa networks cannot be studied as a superposition of independent networks because of imperfect SF orthogonality. Abeele et al. [28] studied the capacity and scalability of LoRaWAN for thousands of nodes per gateway. They showed the importance of considering the capture effect and interference models. They proposed an error model from BER simulations to determine communication ranges and interference. They also analyzed three strategies of network deployments (random SF allocation, a fixed one, and according to related PDR), the last one presenting the best performance. Lim et al. [29] analyzed the LoRa technology to increase packet success probability and proposed three SF allocation schemes (equal interval based, equal area based, and random based). They found that the equal area scheme results in better performance compared with other schemes because of the reduced influence of SFs. The state of the art indicates the interest in heterogeneous deployments and SF allocation strategies. Thus, we analyze homogeneous and heterogeneous deployments with different SF allocations as a function of the number of nodes and traffic intensity in order to show network performance and the benefits of heterogeneity for large scale networks.

7.3 Background

7.4 Approach

We use an accurate and realistic WSNet-based simulator [30] written in C/Modern C++ under CeCILL free license. WSNet is a modular event-driven wireless network simulator that implements the required communication protocol layers and simulates the network behavior with a high level of accuracy. We have extended the simulator in several aspects (e.g., spectrum use, interference, capture effect) to take into account flexibility and specificity of the LoRa PHY/MAC layers.

7.4.1 Spectrum

To support PHY layer heterogeneity and flexibility, we have modified the core of the WS-Net simulator by including a spectrum model. This new core is inspired by Baldo et al. [31] to provide a support for modeling the frequency-dependent aspects of communications. Moreover, it exploits the spectrum in terms of spectral resources instead of logical channels. It provides more accurate PHY models (several waveforms with different configurations) and a more accurate interference model for heterogeneous simulations. Thus, we can evaluate different configurations of LoRa network (e.g., SF, BW, channel) for different homogeneous and heterogeneous deployments. Furthermore, it permits the evaluation of inter-technology interference (from a non LoRa technology), not presented in this article for space reasons.

7.4.2 LoRa Modulation

The LoRa modulation is an adaptation of the CSS modulation. Its advantages are low power transmissions and channel robustness. LoRa features seven orthogonal SFs from SF6 to SF12. Symbol duration T s and bit rate R b are defined as follows:

We have developed a LoRa modulation module inspired by the Equation 3 presented in [17]. This equation shows BER as a function of SF and the energy per bit to noise ratio N

where $Q(x)$ is the Q-function.

7.4.3 Spreading Factors Orthogonality

We have developed an interference module taking into account the effect of the quasi-orthogonality of SFs. It uses the values for co-channel rejection for all combinations of the desired signal SF d and the interferer signal SF i [19] presented in Table I.

Table 7.1

7.4.4 LoRa SX1276 and SX1301 Transceivers

The Semtech SX1276 transceiver provides high interference immunity while minimizing energy consumption [7]. Sensitivity dBm is defined according to the Semtech designer guide [5] as:

where 174 accounts for the thermal noise effect, BW is the receiver bandwidth, N F is the receiver noise factor for a given hardware implementation, and SN R is the minimum ratio of the desired signal power to noise that can be demodulated. Table II shows sensitivity and the data rate for the SX1276 transceiver.

The Semtech SX1301 offers breakthrough gateway capabilities with a multi-channel high performance transmitter/receiver designed to simultaneously receive several LoRa packets with different SFs and up to 8 channels [8]. It enables robust communications for a large number of nodes spread over a wide range. In our simulator, we consider the gateway capture effect based on SX1301. The gateway may receive, depending on the

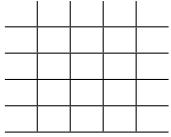


Table 7.2

reception power, several packets with different SFs because they are quasi-orthogonal with respect to each other. If the gateway receives two packets overlapped with the same SF, the gateway will rec

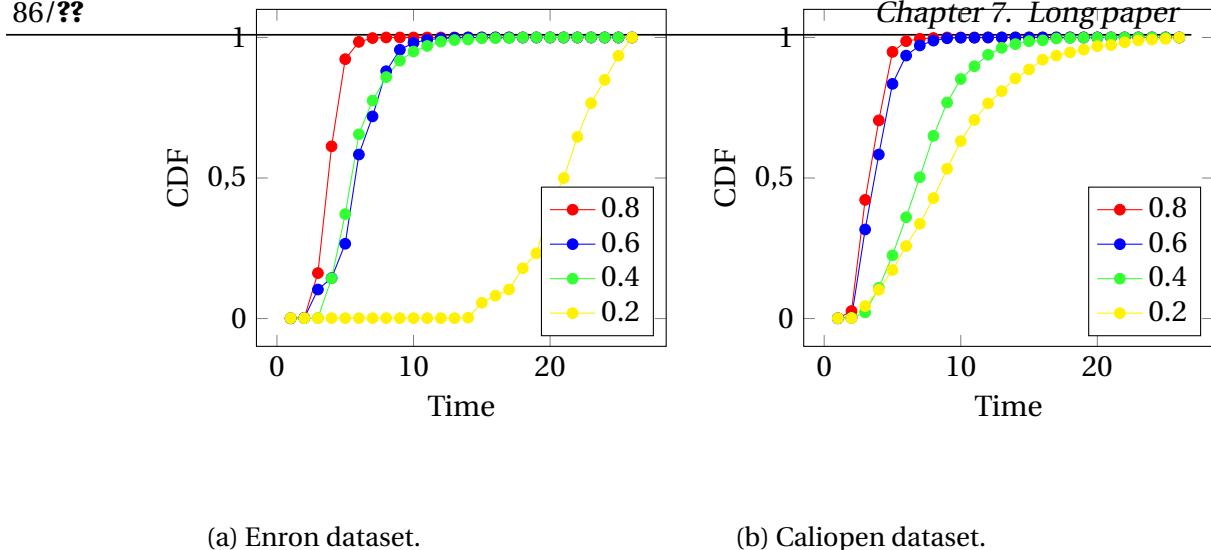
7.4.5 MAC and Application Layers

Our simulator considers a LoRa random access method that basically behaves like ALOHA. The application layer manages the data traffic generation at each node inside a time window Application Period (AP). At every AP occurrence, each node picks randomly a time instant inside the current window, and a data packet is generated and sent to the lower layer (network or MAC). Hence, each node can generate only one data packet per AP. We vary AP between 60 s and 40 min to evaluate performance for different application use cases.

7.4.6 Deployment Scenarios

In this subsection, we present the network deployment scenarios and the SF allocation strategy in each scenario.

- In SF i Homogeneous, nodes are uniformly deployed in a disk of radius equal to $D_{\max}(SF_i)$, the maximum transmission range of SF i (Figure 1(a)), and select the same LoRa configuration SF i .
- Multi-Homogeneous is the superposition of independent SF i homogeneous deployments (i from 6 to 12 and the corresponding radius $D_{\max}(SF_i)$) with a single gateway at the center. The number of nodes are equally distributed between homogeneous networks. As each SF i homogeneous deployment is independent and uniform, we can see in Figure 1(b) higher density close to the gateway.
- In Heterogeneous f(D_{\max}), nodes are uniformly deployed in a disk of radius equal to $D_{\max}(SF_{12})$, the maximum transmission range of SF12 and then, each node selects its configuration according to its link budget (i.e., with the maximum data rate or the minimum SF). As shown in Figure 1(d), some rings naturally appear. Nodes in the farthest ring are configured with SF12, in the next ring with SF11, and so on until the last disk with SF6.
- In Heterogeneous Random, nodes are still uniformly deployed in a disk of radius equal to $D_{\max}(SF_{12})$ but they randomly select their configuration among the ones available according to their link budget. Hence, each node selects its LoRa configuration according to its budget link and its needs. For example, nodes in the third ring can randomly select their SF between SF10 and SF12 depending on their optimization criteria (e.g., energy consumption, data rate, reliability).



(a) Enron dataset. (b) Caliopen dataset.

Figure 1: Cumulative distribution function of infected users.

7.5 Experimentation

To evaluate our approach, we used Enron dataset in our experimentation. There are many reasons for using Enron dataset to evaluate our vulnerability measure techniques. First of all, it is probably the only actual corporate messaging service dataset available to the public. Second, it contains all kind of emails "personal and official", so email logs can reveal the level of trust between users by studying the information flow in the network. Finally, this dataset is similar to the kind of data collected for fraud detection or counter-terrorism, hence, it is a perfect test bed for testing the effectiveness of new vulnerability measure techniques.

The properties of the Enron dataset used for our experimentation are presented in table 7.3.

Parameter	Enron dataset	Caliopen dataset
Nodes	958	5885
Edges	6966	26547
Diameter	958	2096
Mean degree	2.413361	9.02192
Edge density	0.00252	0.001533
Modularity	0.654600	0.86526
Mean distance	3.042114	3.914097

Table 7.3: Datasets properties.

Due to visibility issues, we extract only important nodes given in [25]. Our substantive interest in this experimentation is in how vulnerability moves through the network. The inputs of our experimentation consist of a set of individual vulnerabilities of users in the network. This set is generated randomly and is represented in the graph of Figure ??, values of this set are between 0 and 1 and represent the extent to which users are exposed to different kind of attacks such as (phishing, spam, etc). Unlike social vulnerability values, these values didn't take into account the vulnerability contagion process between users. To evaluate how trust coefficient affects our outputs i.e., social privacy vulnerabilities, we variate the trust coefficient through 4 symmetric values 0.2, 0.4, 0.6 and 0.8.

7.6 Results exploitation

7.6.1 Deployment Scenarios and Assumptions

In this study, we have simulated the deployment scenarios described in Section III.F with a SX1301 gateway, at the center, configured with one single channel of 125 kHz and up to 10000 SX1276 nodes. Hence, the gateway can simultaneously receive up to 7 packets configured with different SF each (SF6 to SF12). Packets are dropped according to the modulation and interference models. Table III presents the simulation parameters. For sake of simplicity, we have considered the 868 MHz band and the Okumura Hata pathloss model.

7.6.2 Homogeneous Deployments Scenarios

In this subsection, we present the results of simulations for homogeneous deployments with different SFs as a function of the number of nodes (up to 10000) with AP set to 60 s (i.e., each node transmits a packet every minute). We have to emphasize that nodes are deployed in a disk of radius equal to the maximum transmission range of SF i for each scenario, i.e. the Homogeneous SF6 in a disk of radius D max (SF 6) and so on. Figure 2 shows PDR for homogeneous deployments with different SFs from 6 to 12. We can see that PDR decreases when the number of nodes increases. The SF6 Homogeneous deployment has better PDR compared with the others due to its short air time but its range is reduced

Figure 3 shows the throughput in packets per second for homogeneous deployments with different SFs. We can see that when the number of nodes increases, the throughput becomes saturated. The SF12 Homogeneous deployment converges faster and presents the lowest throughput due to its low data rate. Nevertheless, it presents the longest range. We have extended the previous analysis by varying AP up to 40 min and we set the number of nodes to 10000. As shown in Figure 4, increasing the Application Period reduces the probability of collision, then PDR increases. Figure 5 shows the throughput as a function of the Application Period. As we can observe, the throughput is higher when nodes use lower SF. Thus, in dense networks with high intensity traffic, we have to prioritize the use of the SF6 configuration. Nevertheless, it reduces the range and therefore, multi-hop communications or denser deployment with several gateways may be necessary to cover the same area. Figure 6 compares the average power consumption per node for the homogeneous deployments as a function of the Application Period for a simulation time offers 100*AP. Increasing traffic intensity (e.g. AP from 40 min to 1 min) results in increased power consumption. SF6 configuration presents low consumption due to its shorter time on air. Higher consumption for the SF12 configuration due to its longer time on air.

7.6.3 Heterogeneous vs. Homogeneous Deployment Scenarios

In this subsection, we compare the simulation results of the heterogeneous strategies with Homogeneous SF12 and Multi-Homogeneous deployments. Each deployment covers the same area (disk of radius D max (SF 12)). For homogeneous deployments, only Homogeneous SF12 is used for comparisons in order to keep the same cell coverage. We simulate the heterogeneous deployments with up to 10000 nodes and AP is fixed to 60 s. Figure 7 compares PDR of homogeneous and heterogeneous deployments. The Heterogeneous f(Dmax) deployment presents the best performance. This is because in heterogeneous deployments we reduce packetloss taking advantage of the quasi-orthogonality of SFs

and the deployment strategy. For 100 nodes, the gain in terms of PDR for the Heterogeneous f(Dmax) deployment is 300% compared to the Homogeneous SF12 deployment. For the Multi-Homogeneous and Heterogeneous Random deployments, the gains are respectively 214% and 200%. A reliability of 20% is reached at 170, 360, 400 and 2600 nodes for the Homogeneous SF12, Heterogeneous Random, Multi-Homogeneous and Heterogeneous f(Dmax), respectively. Figure 8 compares the network throughput for the homogeneous and heterogeneous deployments in received packets per second. When the number of nodes increases, the throughput saturates. The Heterogeneous f(Dmax) deployment presents better performance compared with others deployments up to 10000 nodes. Figure 9 compares PDR as a function of traffic intensity (i.e. with AP from 1 min to 40 min). Decreasing traffic intensity reduces the packet loss, then PDR increases. The Heterogeneous f(Dmax) deployment presents better PDR because it takes advantage of the orthogonality of SFs reducing the interference, e.g., all nodes set up to SF12 are far from the gateway, then the interference to nodes close to the gateway set up to SF6 are reduced. The Homogeneous SF12 deployment presents lower PDR due to its long packet duration and the low spectral efficiency. Figure 10 compares the throughput for homogeneous and heterogeneous deployments. When traffic intensity decreases, the throughput decreases. For high traffic intensity (e.g., 1 min Application Period), the Heterogeneous f(Dmax) deployment presents a throughput of 10 packets per second whereas other strategies have a throughput less than 2 packets per second. Figure 11 compares average power consumption for the homogeneous and heterogeneous deployments for a simulation time of 100*AP. When increasing traffic intensity, power consumption increases. Taking the Homogeneous SF12 deployment as reference, the comparison shows that power consumption for the Heterogeneous f(Dmax) deployment increases smoothly compared with the exponential increasing of the Homogeneous SF12 deployment.

7.6.4 Application Period and Packet Duration vs. ERC Regulations

In Europe, ERC regulates access to radio frequency bands. For most of the sub-bands in the 868 MHz band, the duty cycle must be lower than 1%. We analyze this constraint for a 50 Bytes packet configured with different SFs. Table IV shows the packet duration and the Application Period required to respect the regulation for different values of SF. For an AP of 1 min, the packet duration must be lower than 600 ms to respect the duty cycle of 1%. Table IV shows that the constraint can be satisfied with all SFs except SF11 and SF12. APs for SF12 and SF11 must be longer than 2.6 min and 1.4 min, respectively, in order to respect the regulations.

7.7 Conclusion

The main challenge of this work is to investigate the possibility of using genetic algorithm to model the selection of lora configuration that satisfy the applications requirements. Our main contribution was to develope 3 applications that requires different level of QoS. such as text transmition, sounds transmission and image transmission. We used a low cost lora gateway on raspberrypi and builded 2 arduino boards equiped with 2 lora Transceivers. The gateway capture effect is based on the SX1276 transceiver. To measure the accuracy of applying genetic algorithm in the edge computing to select the best lora configuration we used both simulation and real enviroment tesbeds. Our simulations compare the performance of each configuration selection

homogeneous and heterogeneous deployments as a function of the number of nodes and traffic intensity. First, we have analyzed homogeneous deployments for different SFs from SF6 to SF12. Simulations show better performance for the SF6 deployment but reduced cell coverage. Second, we have compared heterogeneous and homogeneous deployments: the Heterogeneous deployment that selects its LoRa configuration according to its link budget results in the best PDR and throughput, as well as the lowest average power consumption compared to other deployments, for a different number of nodes and different traffic intensity. The results clearly show the benefits of heterogeneity for large scale network deployments and the need for adaptive SF allocation strategies. As a future work, we plan to validate our approach by using both simulation and real environment testbeds. We plane to study the performance of applying our approach in terms of PDR, throughput, and power consumption. The module will optimally select the configuration according to the scenario criteria (e.g., high data rate, energy efficiency, or network congestion) and the radio environment (e.g., link budget, level of interference, device mobility).

Contents

8.1	Introduction	91
8.2	Related work	91
8.3	Related work	91
8.4	Approach	91
8.5	Experimentation	91
8.6	Results exploitation	91
8.7	Conclusion	92

Abstract

8.1 Introduction

The difficulty to build such system is

In this work we

The article is organized as follows. Section 8.2 elucidates summary of related works, In section 8.4, we propose our ... to Section 8.5 evaluate the performance of our ... in terms of packet delivery ratio, throughput, and power consumption. Section ?? concludes the article and gives some ideas for future work.

8.2 Related work

8.3 Approach

8.4 Experimentation

8.5 Results exploitation

8.6 Conclusion



9 | UTLC

UTLC

Contents

9.1	Introduction	94
9.2	Related work	95
9.3	Related work	96
9.4	Use Case and Model Design	96
9.4.1	Use case	96
9.4.2	Design Model	97
9.5	Prototyping	100
9.5.1	6LoWPAN, Contiki OS, Re-Mote and Border Router	100
9.5.2	MQTT and UBIODOTS	101
9.6	Results	102
9.7	Conclusion	104

Abstract

Most traffic light's control systems in smart cities are wired and have a semi-static behavior. They are time-based, with pre-configured pattern and expensive cameras. Although traffic lights can communicate wirelessly with incoming vehicles, they are less adapted to an urban environment. If we consider light signs as an Internet of Things (IoT) network, one issue is to model thoroughly the change of signs' states and the Quality of Service (QoS) of this network. In this paper, we propose a new architecture of Urban Traffic Light Control based on an IoT network (IoT-UTLC). The objective is to interconnect both roads' infrastructures and traffic lights through an IoT platform. We designed our IoT-UTLC by selecting motes and protocols of wireless sensor network (WSN). Message Queuing Telemetry Transport (MQTT) protocol has been integrated to manage QoS. It enables lights to adapt remotely to any situation and smoothly interrupt traffic light's classic cycles. Our experimental results show that the MQTT protocol is efficient when

the packets rate exceeds 35% of traffic flow, it reduces traffic delay up to 0.05s at 90% of congestion. After verification and validation of our solution using a UPPAAL model checker, our system has been prototyped. Motes' functions have been implemented on Contiki OS and connected through a 6LoWPAN/IEEE 802.15.4 network. Time-stamping messages have been performed throughout the system to evaluate the MQTT protocol with different QoS levels. In our experiments, we measured the Round-trip delay time (RTT) of messages exchanged between the WSN and IoT Cloud. The results show that MQTT decreases the RTT when the Cumulative Distributed Function (CDF) of generated messages exceeds 35%.

9.1 Introduction

According to the French agency of statistics on roads' accidents, 12% of them happen in intersections caused by the non-compliance to traffic rules. 23% of them led to hospitalization in which 14% are fatal [Routiere2015]. Urban Traffic Light Control (UTLC) are one possible solution to regulate vehicle flows at intersections. However, a static cycle of traffic lights (or lights signs) has a direct impact on traffic jams, particularly when an emergency vehicle must cross through as quickly as possible. A long period at red or green light might impact the fluidity of the city traffic.

The Internet of Things (IoT) and Everything (IoE) can be a solution to adapt traffic light control to traffic density. It allows heterogeneous connected objects, *e.g.* Zigbee, LoRa, SigFox, ITS-G5, to interact and exchange sensed data on roads, vehicles, pedestrians presence, time of leaving house, etc [Perera2014]. Therefore, connecting heterogeneous infrastructures following a Device-to-Device approach is possible through upper layers or an intermediate Cloud platform. Wireless Sensor Networks (WSNs) are the source of these data, and the Cloud is the remote entity that collects them. Fog and Edge computing have been proposed to address the low latency of IoT applications by efficient resource distribution and a local processing. Fog computing leverages Edge devices and remote and private Cloud resources with distributed data processing. It provides the advantage to process data closer to the source and thus mitigate latency issues and reduce network congestion. However, constructing a real IoT Fog Computing is costly for evaluation while the environment has to be controllable for experiments [Dastjerdi2016]. However, in our work we would implement our Edge computing with a remote Cloud data gathering and deal with latency through QoS protocol. Remote Cloud offers the possibility to integrate new services. For example, we can deploy sensors to measure noise or air pollution via traffic signs or roads.

Our objective is to model, prototype and evaluate the Quality of Service (QoS) of an IoT solution for a traffic light control system. Modeling of traffic light states control is essential to avoid incoherent situations. Number of models based on Petri Nets (PN) have been proposed in [difebrero_trafficresponsive_2006] and [26]. Their main drawbacks are the limitation to the structural analysis of state transitions and the lack of verification. However, our design is based on UPPAAL (UPPSala and AALborg Universities) [27] timed automata for design and verification of coherent state of cross road's traffic light. It specifies a graph of states with clocks and data variables. To implement our Urban Traffic Light Control based on an IoT network architecture (IoT-UTLC), we setup a real IEEE 802.15.4

WSN with devices that can act as actuators and sensors. Small traffic light signs are driven by a Border Router (BR) device to a sink node which is a gateway to the Internet. This BR is connected to a host computer (or sink) also connected to an IoT Cloud platform. WSN devices forward their data to the IoT Cloud through this sink which defines required levels of QoS based on Message Queuing Telemetry Transport (MQTT) protocol [Al-Fuqaha2015]. The collected data can be transmitted to different devices such as sink, BR or wireless sensor/actuator devices. When WSN devices detect the arrival of high priority vehicles, sensed data are routed to the IoT Cloud. Then, the sink node takes a decision to change the light's state and forward generated messages to actuator devices through BR with a high level of QoS. Thanks to IPv6 over Low power Wireless Personal Area Network (6LoWPAN) [chalappuram_development_2016], our WSN is energy-efficient and IPv6 compatible.

This paper is organized as follows. In Section 9.2, we review related work. Section ?? reports the design of our prototyping solution. We describe the use case defined with the design model. Section ?? defines our prototype (IoT Testbed), giving our specifications and discussing on our choices of technologies and protocols. Finally, Section 9.6 presents the obtained results that show the usefulness and the best practice for MQTT.

9.2 Related work

Petri nets (PNs) are widely used for traffic light modelling and control [26]. In [difebbraro_trafficresponses], deterministic-timed Petri Nets have been used to describe signalized intersections. Undesirable deadlock states might appear when the nets are tested for some use cases. The authors in [28] have modified PNs models including stochastic-time for one single signalized intersection. Dotoli and Fanti [29] have built a colored timed PN with a deterministic modular framework, in which parts of the system, and even parts of the subsystems, can be specified and analyzed separately. Examples using modularity are given in Soares and Vrancken [30], in which a p-timed PN is used for the control of a traffic signal in both main road and side streets. However, formal characteristics of PNs (*e.g.*, deadlock and liveness) haven't been discussed. Moreover, PNs suffer from a lack of analysis and verification tools. To overcome these limits of PNs, we propose UPPAAL timed automata for design and verification of coherent state of cross road's traffic light. UPPAAL is a timed-based modelling software with a graphical user interface. It is the result of the research works of two universities UPPsala University in Sweden (UPP) and AALborg University in Denmark (AAL) [27].

In [Web0], thermal cameras and on-street wired sensors detect vehicles and pedestrians in order to adapt the cycle of traffic light control systems. However, such a solution can be expensive. In addition, the system uses only its local view of the environment to detect the arrival of a vehicle. Other solutions use recent technologies such as wireless sensors devices to limit the cost of thermal cameras and reduce the time needed to deploy sensors. In [tlig_decentralized_2014] and [rose_internet_2015], the authors propose an adaptive system based on local wireless communication between lights and vehicles. But such a solution requires a global interconnection between all road's users and infrastructure. This problem comes from the rigid definition of technologies' standards. Our work is not only limited to establish WSN, but it is scalable to interconnect heterogeneous wireless technologies through the Internet. The obtained WSN intends to meet multiple QoS requirements of IoT applying the MQTT protocol. In [Silva2018], the latency of MQTT has been evaluated by calculating the average round-trip delay between two clients located in two different continents. However, the evaluation has been limited to the impact of mes-

sages' size. In our work, we consider the period of generated messages, and we calculate the RTT delays from WSN to Cloud IoT platform.

In [26] [[difebbraro_trafficresponsive_2006](#)] [28] [30], the authors focus only on the structural analysis of their models and the transitions between colors of traffic lights. However, the implementation of their models as a service in the Internet of smart cities has not been discussed. Moreover, their methodology is not tested with any real traffic lights' Testbed.

9.3 Related work

9.4 Use Case and Model Design

Our objective is to build a robust Testbed that behaves like a real urban traffic light control system. As presented in section 9.2, a crossroad traffic light design has been proposed based on PN [26]. Authors demonstrated the necessity to monitor checkpoints like traffic light transitions from red to green and define critical control points to ensure that the transition model is correct. Authors have identified which signal indication sequence optimizes the overall system performance. We were inspired from this work by adding the vulnerability of wireless network *i.e.* packet loss. Indeed, this model is theoretical and static, and would not model entirely traffic light control system. Thus, we propose a UP-PAAL timed automata for modelling and verification of the system.

9.4.1 Use case

We consider the use case of traffic lights at an intersection crossed by high priority vehicle like ambulances, fire-fighters or public transportation systems. Crossing delays are important in such use cases, when the goal is to travel in the city from two locations without experiencing traffic jams. We consider the case of a high priority vehicle approaching a traffic light sign on red state. The detection of priority vehicles via (RFID or touch sensor) triggers the transmission of notification to road signs' network asking for a switch of traffic light to green. In that situation, it should be possible to interrupt the usual cycle of a crossroad.

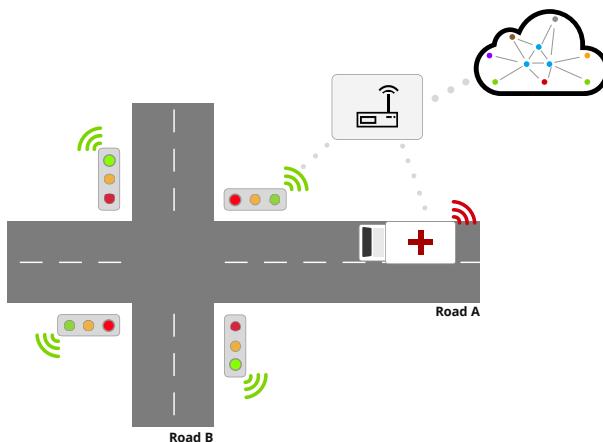


Figure 1: Use case illustration

In order to be as close as possible to a real urban traffic light, we prototyped the closest

Testbed of a crossroad in Paris. We took an actual crossing point with its dimension and static timers. Fig. 1 shows IoT interconnection of four traffic lights, high priority vehicles and roads. Our system is described by one intersection (or crossing) of two roads A and B, with two traffic lights by road. The signs and roads can use heterogeneous technologies as presented with different colors.

The roads are connected to the Internet through sensors, which are able to detect the arrival of high priority vehicles. The detection is performed by Touch sensors driven by WSN nodes. Note that signals on each road should always have different colors (or states). Of course, when the road A traffic light is on green, the state of the traffic light on the opposite road, must be on red and vice versa. To access the Internet, all messages sent by those objects will go through a Border Router (BR) device and a Middleware or Edge computer. The Middleware is connected to the Internet and forwards all packets to a Cloud. Therefore, the collected data in the Cloud allow a Middleware to decide for the future state of lights. Then, BR disseminates this decision through the network to the actuators.

9.4.2 Design Model

Our design is based on UPPAAL model checker software. It specifies a graph of states with clocks and data variables. It allows us to model how our system works and simulates all possible traffic lights states. The modelling allows us to cover all possible cases of lights change of our IoT-UTLC. It is also a tool for verifying formally the consistency of traffic light changes: GREEN to YELLOW states, YELLOW to RED states and RED to GREEN states. We simulated our system by three automata shown in Fig. 3, Fig. 2 and Fig. 4 and available at (<https://github.com/IoT-UTLC/Resources>). We proved that our model worked without deadlock and starvation. It means that in our system, there is always a transition to go to the next state. It proves that the system will not stop functioning over time. Incoherent situations, like four signals on GREEN, must not happen.

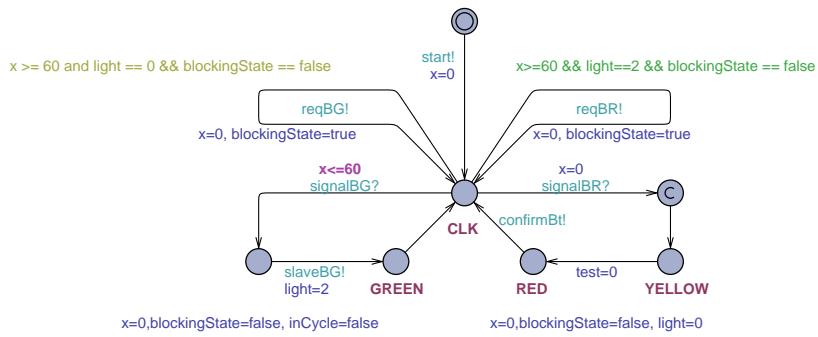


Figure 2: Model of our Traffic Lights in UPPAAL

Fig. 2 highlights the model of traffic lights and describes their behavior. The change of light's state is based on requests and confirmation exchange between lights and the Middleware. We defined two roles for the traffic lights: one is set to master mode and the second one is set to slave mode. Masters send requests every 60 seconds to change their states while considering the current ones. Line 6 of Algorithm 1 shows the condition when a light should change its state. CLK state represents clock or time progress. *reqAG* and *reqAR* are respectively the requests to ask for GREEN and RED states on road A. The same is specified for the road B. When it receives a confirmation from the Middleware, a cycle is started to send the signal with the desired state. Every 30 seconds, the traffic lights can change their states, starting with GREEN light and then to YELLOW light for 3 seconds

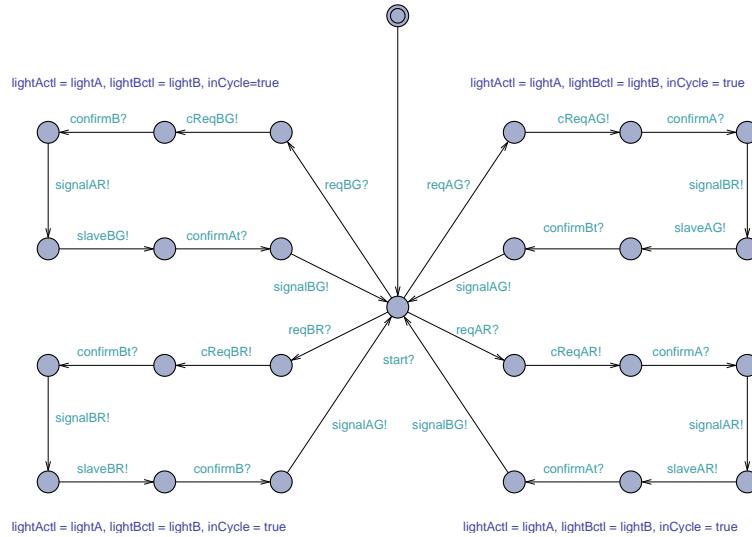


Figure 3: Model of our Middleware in UPPAAL

after that switching to RED. They can also start from RED and then change to GREEN state after additional 3 seconds. We add this extra delay to avoid a dangerous situation when a GREEN state is on the two roads A and B at the same time. Even if we have messages lost due to the wireless nature of the network, our UPPAAL models ensure that this situation doesn't arise. It has been introduced after experiencing a latency between the WSN and the Cloud platform (see section 9.6). When GREEN or RED states are actuated, confirmation messages are sent to the Middleware. The pseudo Algorithm 1 summarizes this mechanism.

Algorithm 1: Traffic light

```

1 init_60s_timer(); while true do
2   if end_timer() then
3     send_request_new_state();
4     reset_timer();
5   end
6   if msg_received_red() and my_state is green then
7     change_state(yellow);
8     wait();
9     change_state(red);
10    send_confirmation_to_middleware();
11  end
12  if msg_received_green() and my_state is red then
13    change_state(green);
14    send_confirmation_to_middleware();
15  end
16 end

```

The model in Fig. 3 defines the different possibilities in terms of internal cycles depending on the request made by traffic lights (*reqAG*, *reqBG*, *reqAR* *reqBR*). Mainly, the Middleware sends the message to the Cloud and waits for its response. Then, it sends messages to traffic lights master and slave to change their state following this order: every light go to RED before setting GREEN signals. It also uses acknowledgments from the traf-

fic lights to ensure that the new state has been set. In order to ensure these two features, we used a system to retain messages if the IoT Cloud Platform send GREEN states before RED states (see line 3 of Algorithm. 2). Algorithm 2 shows a simple description showing how the Middleware confirms the order of traffic lights changes.

Algorithm 2: Middleware confirmation

```

1 if message_received() then
2   if is_green() then
3     | retain_msg(); //green then red
4   end
5   else
6     | if retained_msg_exist() then
7       |   update_to_red(); //green then red
8     end
9     | else
10      |   update_to_red(); //red then green
11    end
12  end
13 end
14 while true do
15   confirmation_red_lights();
16   update_to_green();
17   confirmation_green_lights();
18 end

```

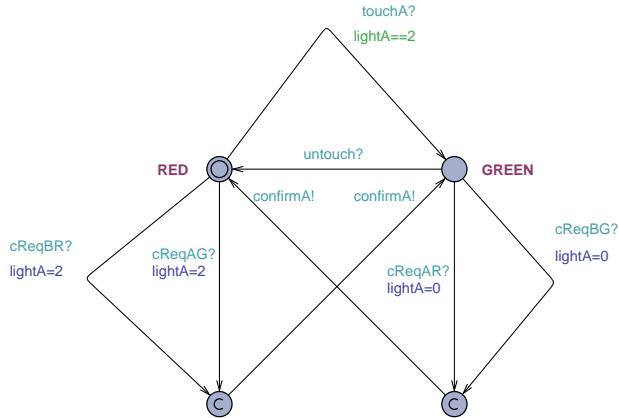


Figure 4: Model of our Cloud variables in UPPAAL

We introduced the IoT Cloud Platform model shown in Fig. 4. It simulates the subscription mechanism according to messages sent by the Middleware to update collected data. We defined two states RED and GREEN without a transition state like YELLOW state defined for the traffic lights. The condition *touchA* indicates if the road A detects the high priority vehicle. The name *touch* is related to the type of sensor integrated in our prototype (see next section). The Cloud confirms to the Middleware that the state is changed by sending a message *confirmA*.

Exchanged messages within our WSN are based on IEEE 802.15.4 stack. And our Middleware defines QoS levels of exchanged messages via MQTT protocol.

9.5 Prototyping

We have prototyped the wireless sensors and actuator's network of traffic lights and roads on a mockup¹ of real intersection in Paris with a scale of 1:68. Our specifications have been defined considering the low-cost and energy efficiency of the solution. This Testbed is a proof of concept of not limited to our use case as it is scalable for other applications. For example, additional sensors of fine particules could be implanted bringing correlation between traffic jam and pollution.

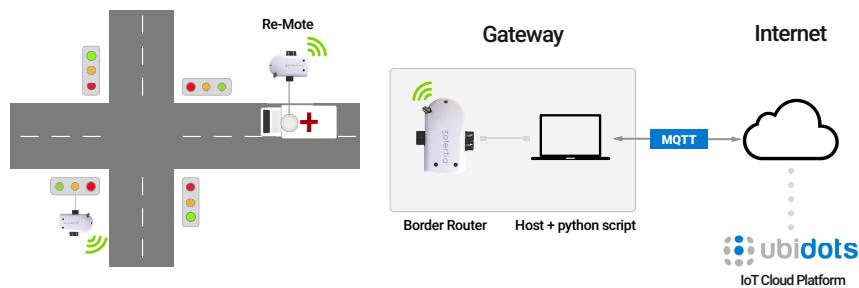


Figure 5: Architecture of IoT-UTLC

Fig. 5 shows the architecture of our IoT-UTLC with three layers. From left to right, we have the WSN layer with connected traffic lights actuators, sensors and IEEE 802.15.4 transceivers. The second part is the gateway of the WSN ensured by the BR and the Middleware *i.e.* Python script launched by host computer. The last layer is the Ubidots IoT Cloud Platform. It is an open source solution used to collect and analyze WSN data.

9.5.1 6LoWPAN, Contiki OS, Re-Mote and Border Router

Our WSN is an IPv6 LowPower Wireless Personal Area Network (6LoWPAN) based on IEEE 802.15.4 stack. It is well adapted to embedded wireless devices with energy aware constraint and for its capabilities to define a mesh topology. Contiki Os² has been used to implement networks' functions such as send, receive and data processing. It is an embedded operating system with large open source community. It supports Zolertia's Re-Motes³ and implements recent IEEE 802.15.4 standard specifications. It also includes protocols such as RPL, CoAP and MQTT. Furthermore, developer community is active and makes available source codes examples in order to help developing quickly new applications.

Re-motes are compatible with our WSN specifications and our design model. They are wireless devices with ultra-low power operation mode. This choice has been motivated by long radio range of its IEEE 802.15.4 CC1200 transceiver, which transmits in the frequency band of 868-915 MHz. In addition, each Re-Mote has analog and digital ports with a possibility to connect several analog sensors and actuators. A Re-mote can be driven by a computer and become a sink and/or BR as well as a gateway between the 6LoWPAN network and the computer.

To implement the previous model described in Section ??, we used six Re-motes: one for the BR, four to control the traffic lights and one Re-Mote to detect the arrival of a high

¹<https://github.com/IoT-UTLC/Resources/wiki>

²<http://www.contiki-os.org/>

³<https://github.com/Zolertia/Resources/wiki/BE-Mote>

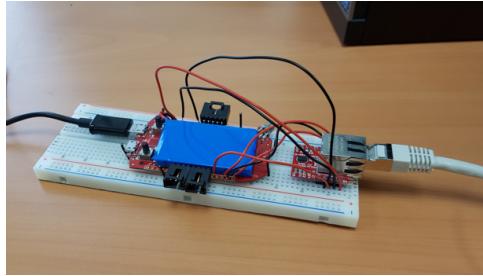


Figure 6: BR and sink combined on one board

priority vehicle near a crossing point. For simplicity, we choose a touch sensor as a detecting device of priority vehicle. We have developed four types of programs running on a Re-mote: traffic lights signs, sensors, high priority vehicle detecting device and BR function. Sensors send periodically information to the IoT Cloud Platform with temperature, pressure or any relevant information that can be sensed. As mentioned in the previous section, traffic lights are sub-divided into two modes: slaves and masters. Masters nodes are the only ones to request the Middleware to change its lights state and slaves simply change its state depending on received packets. These roles are defined to reduce the overhead of network, redundancy and collisions, for instance. Masters send periodically packets to request a change of state to the Middleware which forwards them to a Cloud platform.

BR node behaves differently compared to the other Re-motes. The entries of its routing table are the list of Re-motes that pass through it. It reroutes every packet it receives from its neighboring to host computer (or sink), which creates a connection to the IoT Cloud platform. Two options are possible to create our BR: i) separate the BR and sink and ii) combine both on the same device. In our development, we worked on how to implement the sink and the BR nodes on the same Re-Mote board. Fig. 6 shows a prototype of the combined BR and sink, both connected to an ethernet interface. Indeed, if the border router becomes an Ethernet router, there will no longer be any connection between the host/sink machine and the IoT Cloud platform. Every Re-mote is able to connect independently to the IoT Cloud platform. This approach has some advantages, such as the autonomy of the devices, but it generates an overhead requiring extra synchronization packets' exchange. Therefore, we separate the sink and BR, since this solution is more flexible and resilient for our Testbed.

9.5.2 MQTT and UBIDOTS

Fig. 7 presents the layers of our UTLC network. From bottom to up, the WSN network sense and/or detect, process and actuates traffic lights. The second layer manages the 6 LowPan addressing and routing of packets throughout an IEEE 802.15.4 network. The Edge Computing is the Middleware between the WSN and the Cloud platform. For the setup of our UTLC, we start by establishing the access network of WSN. The next step is to connect this network to Core network. MQTT protocol controls three levels of QoS of exchanged packets from the WSN to the chosen Ubidots⁴ Cloud platform. It adopts IntServ approach for supporting quality of service in the network, it tags incoming packets in the border routers with different levels of priority. Core routers read incoming packets headers and queue them according to their priority, packets with a high priority are sent faster compared to low priority ones.

⁴<https://ubidots.com/>

MQTT ensures the QoS and publish/subscribe mechanisms through a broker. The broker behaves as a server by filtering messages and organizing them in topics, which are strings used to filter messages and define the hierarchy of our data structure. They allow us to organize how to receive multiple data from sensors such as temperature, up time, battery status and how to display them and obtain a real-time glance of our system. It gets its messages from publishers and sends any modifications to entities, which that subscribed to the updated topics. We used this mechanism with the Middleware in order to publish messages to the broker and get from the main topic the new values of the subscriber.

The QoS feature of MQTT protocol manages network resources by handling retransmissions and guarantees the delivery of messages. It allows more control on messages by defining the level of guarantee. By default, the QoS is defined by three levels. The first one, level 0, is 'At most one'. Level 1 is 'At least one' where there is an acknowledgment to let the sender know that its packet has been received. Finally, level 2 'Exactly once' is the highest verification level with a request/response flows to ensure that only one message will be delivered and processed by the receiver. In our case, we applied levels 1 and 2 using *paho.mqtt.client* Python library. For example, publisher of high priority data such as touch sensor has to indicate the highest level of QoS by the code shown bellow. We shared our implementation and its source codes at <https://github.com/IoT-UTLC/contiki>.

```
payload = json.dumps({"RoadA": data, "RoadB": 0})
res, mid = conn.publish(MQTT_URL_PUB, payload,
    qos=int(QoS)) # QoS is QoS level to use
```

We experienced significant latency of high priority messages when we tested of IoT-UTLC mockup. Therefore, assessments of the MQTT protocol in our case provided significant information about its efficiency.

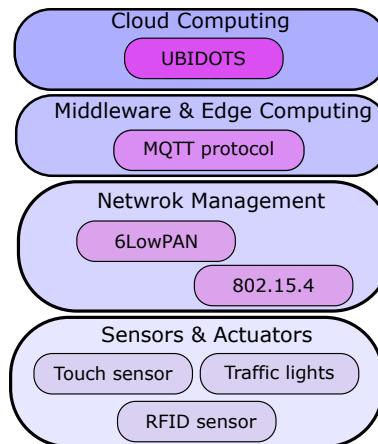


Figure 7: UTLC network layers

9.6 Results

In this section, we report experimental results of urban traffic-light control system based on MQTT protocol. We report the limitations of our solution. To test the efficiency of MQTT protocol, we made 2 scenarios, in the first scenario, the frequency packets sending is 1 packet per 10s, the second scenario, the frequency packets sending is 1 packet per 1s. The measured delays have been taken into account between the Middleware (Edge) to

the Cloud platform (Ubidots) throughout the Internet. Note that we don't know the routes and the routers that our packets will go through. We measured the Round Trip Time (RTT) for the packets exchanged between the two sides. In each scenario, more than 100 values have been taken. Tests have been reproduced for two different hours and days.

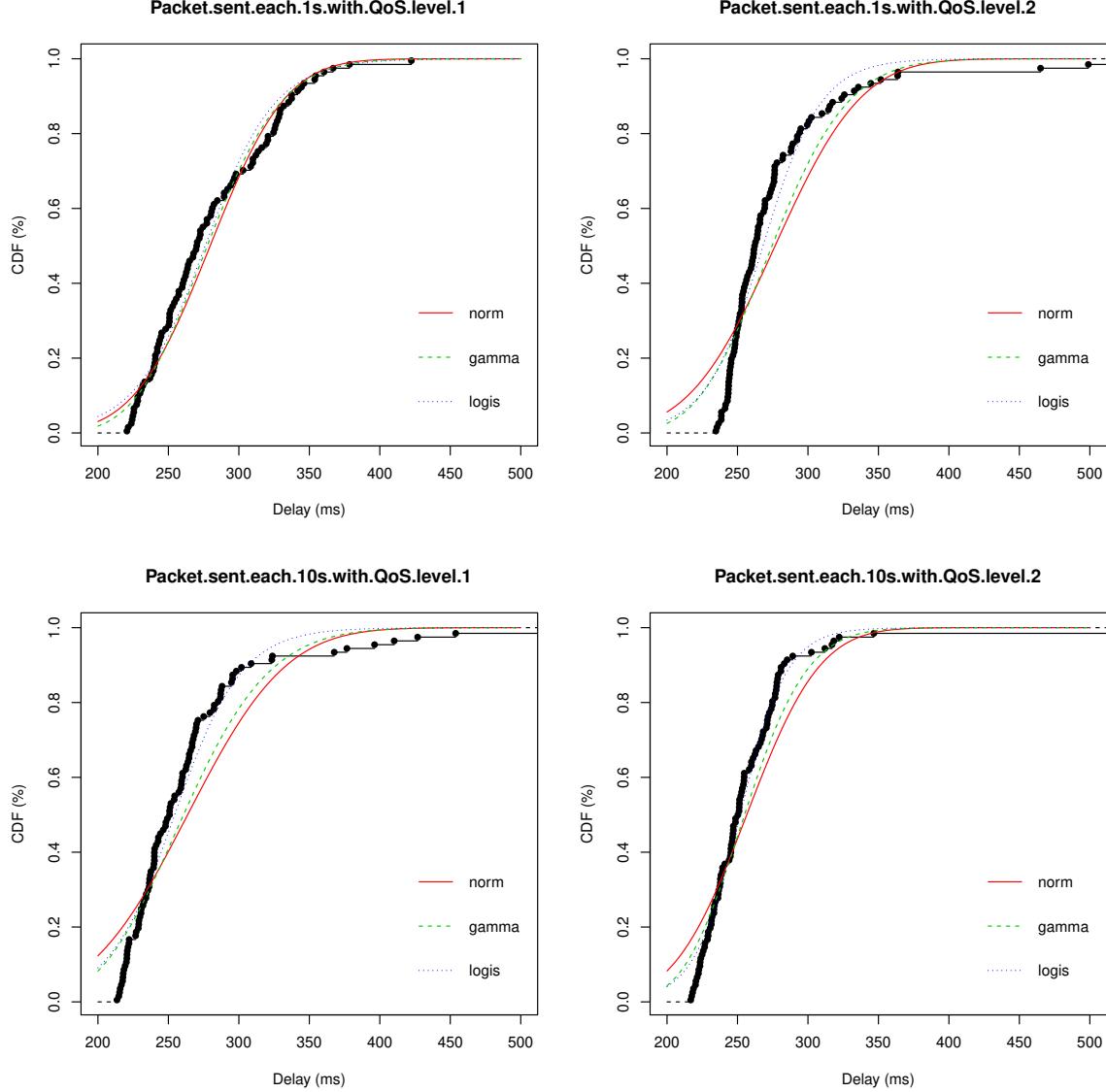


Figure 8: Normal, Gamma and Logistic distribution

Fig. 8 shows the measured Cumulative Distribution Function (CDF) of RTT delays for the two levels of QoS, levels 1 and 2. We consider three probabilistic distribution functions (Normal, Gamma and Logistic) in the experimental results in order to characterize MQTT performance. The obtained correlation allows us to define a representative empirical model. Table 9.1 details the results of a correlation matrix between distributions and experimental results. We can see that logistic distribution [STEPHENSON1979] fits better with our measured RTT values for the two scenarios. The standard logistic law is of parameters 0 and 1. Its distribution function of a random variable x is the sigmoid following the expression:

$$F(x) = \frac{1}{1 + e^{-x}}, \text{ where } x \in [-\infty, +\infty] \quad (9.1)$$

Table 9.1: Correlation between distributions and empirical results

	norm	gamma	logis
1s with QoS level 1	172.12074	175.2950	185.4433
1s with QoS level 2	159.59630	172.8193	189.7002
10s with QoS level 1	146.85668	161.2369	175.3682
10s with QoS level 2	176.28502	192.6108	204.3235

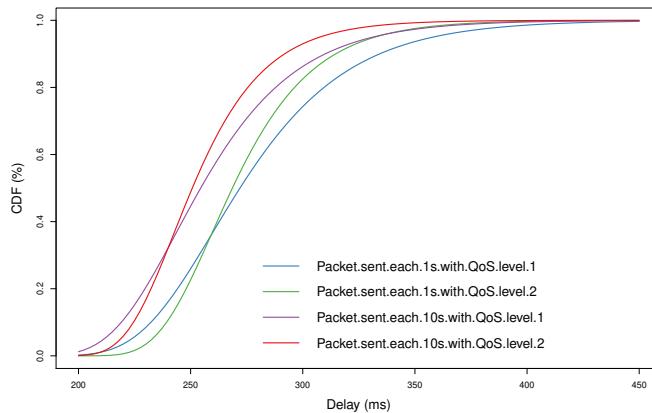


Figure 9: Cumulative distribution function of RTT delay for two QoS levels

Fig. 9 highlights the empirical model of CDF featuring the RTT of MQTT protocol. As can be seen, RTT protocol is more efficient when the number of packets is greater than 35% of the total number of packets sent. This can be explained by the fact that priority queues are useful when queues of QoS are full. In that case, the packets with a highest priority level will reach their destination with low latency. Furthermore, when we increase the number of packets sent to Ubidots at one per second, the MQTT still offers the same efficiency with suitable delays. The found latency of up to 400 ms would be problematic for real world safety applications which require at most 100 ms [Chen2017].

Although our Mockup is innovative by combining IEEE 802.15.4, 6LowPAN, MQTT protocol and Edge Computing, the performances of our solution depend on external parameters. These parameters are related to Internet Service Provider and all packet's routes throughout Internet network from the Middleware to Ubidots. Thus, real implementation of such a system should be done by introducing a private Cloud near data sources.

9.7 Conclusion

In this paper, we proposed a Urban Traffic Light Control (IoT-UTLC), considering its architectures elements and tools used to build an IoT mockup. We reported three main contributions in this work: i) Modelling through UPPAAL of crossing's traffic lights, ii) Prototyping of IoT Edge Computing, iii) Performance assessment of MQTT protocol. Traffic lights control has been taken as an IoT network. WSN has been deployed on motes running Contiki Os and exchanging IEEE802.15.4/6LowPAN packets. MQTT was the QoS protocol between our WSN and Ubidots Cloud platform. UPPAAL model checker design ensured that lights' colors change is adaptive to the arriving of a priority vehicle.

Our experiments have investigated the relationship between the MQTT protocol and the traffic flow congestion. Our results showed that the MQTT is efficient when the num-

ber of packets sent exceeds 35% of the total number. The packets with highest level of QoS has low latency than other packets. The protocol remains efficient since the delay of priority packets decreases when the network overhead increases. However, found latencies of up to 400ms is higher than the expected one for vehicular safety application. Thus, the proposed IoT architecture and protocols must be improved to consider the safety requirements.

While we are still developing our prototyping, we plane to integrate other use cases such as smart buildings and industrial IoT. As a near future work, we plane to extend our experiments with private Cloud towards a real Fog Computing.

Acknowledgement

We would like to thank our colleague Sebti Mouelhi, associate professor at ECE Paris, who provided us insight and expertise on UPPAAL.



10 | Conclusion

"Everything that has a beginning has an ending. Make your peace with that and all will be well" - Jack Kornfield

Contents

10.1	Conclusion	107
10.2	Perspectives	107

10.1 Conclusion

10.2 Perspectives

A | Appendix A

Appendix A

Contents

A.1	Introduction	109
A.2	Related work	109
A.3	Related work	109
A.4	Approach	109
A.5	Experimentation	109
A.6	Results exploitation	109
A.7	Conclusion	109

Abstract

A.1 Introduction

A.2 Related work

A.3 Related work

A.4 Approach

A.5 Experimentation

A.6 Results exploitation

A.7 Conclusion

B | Appendix B

Appendix B

Contents

B.0.0.1	CoAP	113
B.0.0.2	MQTT	113
B.0.0.3	XMPP	114
B.0.0.4	AMQP	114
B.0.0.5	DDS	114
B.0.0.6	mDNS	115

- ADR Network server will change the data rate through appropriate MAC commands
 - Modulation :
 - * Lora: 8 Symbols, 0x34 (Sync Word)
 - * FSK: 5 Bytes, 0xC194C1 (Sync Word)
 - Length :
 - * The Payload length (Bytes)
 - * Optional 16bit CRC for payload
 - Header: It contains:
 - * MAC Header
 - * MTType: is the message type (uplink or a downlink)
 - > whether or not it is a confirmed message (reqst ack)
 - = 000 Join Request
 - = 001 Join Accept
 - = 010 Unconfirmed Data Up
 - = 011 Unconfirmed Data Down
 - = 100 Confirmed Data Up
 - = 101 Confirmed Data Down
 - = 110 RFU
 - = 111 Proprietary
 - * RFU Reserved for Future Use
 - * Major: is the LoRaWAN version; currently, only a value of zero is valid
 - = 00 LoRaWAN RI
 - = 01-11 RFU
 - * MAC Payload :
 - * Frame Header
 - > DevAddr: the short address of the device
 - > NwkID (Network ID): 31th to 25th
 - > NwkAddr (Network Address): 24th to 0th
 - > Encr

0	1	2	3	4	5	6	7
CJP	Type	Dest	Source	Message ID	Length	RFU	MPayload

Frame Payload:

- ▶ CJP
 - ▶ Type
 - 1. Data
 - 2. Transfer
 - 3. Ack
 - 4. Search
 - 5. Inform
 - ▶ Dest
 - ▶ Source
 - ▶ Message ID
 - ▶ Length
 - ▶ RFU
 - ▶ Message Payload

B.0.0.1 CoAP

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Ver	T	TKL	Code	Message ID	
Token					
Options					
11111111		Payload			

CoAP Header

Ver: is the version of CoAP

T: is the type of Transaction

TKL: Token length

Code: represents the request method (1-10) or response code (40-255).

- Ex: the code for GET, POST, PUT, and DELETE is 1, 2, 3, and 4, respectively.

Message ID: is a unique identifier for matching the response.

Token: Optional response matching token.

B.0.0.2 MQTT

0 **1** **2** **3** **4** **5** **6** **7**

Message Type	UDP	QoS Level	Retain	
Remaining length				
Variable length header				
Variable length message payload				CoAP Header

Message type: CONNECT (1), CONNACK (2), PUBLISH (3), SUBSCRIBE (8) and so on

DUP flag: indicates that the message is duplicated

QoS Level: identify the three levels of QoS for delivery assurance of Publish messages

Retain field: retain the last received Publish message and submit it to new subscribers as a first message

B.0.0.3 XMPP

- ▶ Extensible Messaging and Presence Protocol
- ▶ Developed by the Jabber open source community
- ▶ An IETF instant messaging standard used for:
 - ▶ multi-party chatting, voice and telepresence
- ▶ Connects a client to a server using a XML stanzas
- ▶ An XML stanza is divided into 3 components:
 - ▶ message: fills the subject and body fields
 - ▶ presence: notifies customers of status updates
 - ▶ iq (info/query): pairs message senders and receivers
- ▶ Message stanzas identify:
 - ▶ the source (from) and destination (to) addresses
 - ▶ types, and IDs of XMPP entities

B.0.0.4 AMQP

Size the frame size.

DOFF the position of the body inside the frame.

Type the format and purpose of the frame.

- ▶ Ex: 0x00 show that the frame is an AMQP frame
- ▶ Ex: 0x01 represents a SASL frame.

B.0.0.5 DDS

- ▶ Data Distribution Service
- ▶ Developed by Object Management Group (OMG)
- ▶ Supports 23 QoS policies:
 - ▶ like security, urgency, priority, durability, reliability, etc
- ▶ Relies on a broker-less architecture
 - ▶ uses multicasting to bring excellent Quality of Service
 - ▶ real-time constraints
- ▶ DDS architecture defines two layers:
 - DLRL Data-Local Reconstruction Layer
 - * serves as the interface to the DCPS functionalities
 - DCPS Data-Centric Publish/Subscribe
 - * delivering the information to the subscribers
- ▶ 5 entities are involved with the data flow in the DCPS layer:
 - ▶ Publisher: disseminates data
 - ▶ DataWriter: used by app to interact with the publisher
 - ▶ Subscriber: receives published data and delivers them to app
 - ▶ DataReader: employed by Subscriber to access received data
 - ▶ Topic: relate DataWriters to DataReaders
- ▶ No need for manual reconfiguration or extra administration

- ➡ It is able to run without infrastructure
- ➡ It is able to continue working if failure happens.
- ➡ It inquires names by sending an IP multicast message to all the nodes in the local domain
 - ➡ Clients ask devices that have the given name to reply back
 - ➡ the target machine receives its name and multicasts its IP @
 - ➡ Devices update their cache with the given name and IP @

B.0.0.6 mDNS

- ➡ Requires zero configuration aids to connect machine
- ➡ It uses mDNS to send DNS packets to specific multicast addresses through UDP
- ➡ There are two main steps to process Service Discovery:
 - ➡ finding host names of required services such as printers
 - ➡ pairing IP addresses with their host names using mDNS
- ➡ Advantages
 - ➡ IoT needs an architecture without dependency on a configuration mechanism
 - ➡ smart devices can join the platform or leave it without affecting the behavior of the whole system
- ➡ Drawbacks
 - ➡ Need for caching DNS entries

Year	Factors	Computation Model	Results interpretation
2018 jhjhjhjhjhjhjhjhj [31]	-Closeness Centrality -Degree Centrality	Estimation	Closeness have a high degree correlation with privacy scc

Table B.1: Social metrics

Bibliography

Others

- [1] Johan Bregell. “ Hardware and Software Platform for Internet of Things ”. In: *Master of Science Thesis in Embedded Electronic System Design* (2015). 00002 (p. 3, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42–44, 46, 48, 50, 52, 54, 56, 58).
- [2] Djamel Eddine Kouicem, Abdelmadjid Bouabdallah, and Hicham Lakhlef. “ Internet of Things Security: A Top-down Survey ”. In: *Computer Networks* 141 (Aug. 4, 2018). 00029, pp. 199–221 (p. 7).
- [3] V. Prasanna Venkatesan, C. Punitha Devi, and M. Sivarajanji. “ Design of a Smart Gateway Solution Based on the Exploration of Specific Challenges in IoT ”. In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC). 00004. Palladam, Tamilnadu, India: Feb. 2017, pp. 22–31 (p. 7, 9).
- [4] Mattia Rizzi et al. “ Evaluation of the IoT LoRaWAN Solution for Distributed Measurement Applications ”. In: *IEEE Transactions on Instrumentation and Measurement* 66.12 (Dec. 2017). 00000, pp. 3340–3349 (p. 7).
- [5] Luca Feltrin et al. “ LoRaWAN: Evaluation of Link- and System-Level Performance ”. In: *IEEE Internet of Things Journal* 5.3 (June 2018). 00000, pp. 2249–2258 (p. 9).
- [6] Enrique Alba. “ Intelligent Systems for Smart Cities ”. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO ’16 Companion*. The 2016. 00004. Denver, Colorado, USA: ACM Press, 2016, pp. 823–839 (p. 7).
- [7] Ala Al-Fuqaha et al. “ Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications ”. In: *IEEE Communications Surveys & Tutorials* 17.4 (24–2015). 02482, pp. 2347–2376 (p. 10, 14, 38).
- [8] Zhijing Qin et al. “ A Software Defined Networking Architecture for the Internet-of-Things ”. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. NOMS 2014 - 2014 IEEE/IFIP Network Operations and Management Symposium. 00258. Krakow, Poland: May 2014, pp. 1–9 (p. 13).
- [9] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke. “ A Survey on Software-Defined Wireless Sensor Networks: Challenges and Design Requirements ”. In: *IEEE Access* 5 (2017). 00135, pp. 1872–1899 (p. 13).
- [10] Musa Ndiaye, Gerhard Hancke, and Adnan Abu-Mahfouz. “ Software Defined Networking for Improved Wireless Sensor Network Management: A Survey ”. In: 17.5 (May 4, 2017). 00058, p. 1031 (p. 13, 14).
- [11] Samaresh Bera, Sudip Misra, and Athanasios V. Vasilakos. “ Software-Defined Networking for Internet of Things: A Survey ”. In: *IEEE Internet of Things Journal* 4.6 (Dec. 2017). 00067, pp. 1994–2008 (p. 13).

- [12] Tie Luo, Hwee-Pink Tan, and Tony Q. S. Quek. “ Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks ”. In: *IEEE Communications Letters* 16.11 (Nov. 2012). 00356, pp. 1896–1899 (p. 14).
- [13] Salvatore Costanzo et al. “ Software Defined Wireless Networks (SDWN): Unbridling SDNs ”. In: (2012). 00183, p. 25 (p. 14).
- [14] Laura Galluccio et al. “ SDN-WISE: Design, Prototyping and Experimentation of a Stateful SDN Solution for Wireless SEnsor Networks ”. In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE INFOCOM 2015 - IEEE Conference on Computer Communications. 00173. Kowloon, Hong Kong: Apr. 2015, pp. 513–521 (p. 14).
- [15] Sérgio I. Lopes et al. “ Design of Compact LoRa Devices for Smart Building Applications ”. In: *Green Energy and Networking*. Ed. by João L. Afonso, Vítor Monteiro, and José Gabriel Pinto. Vol. 269. 00000. Cham: Springer International Publishing, 2019, pp. 142–153 (p. 36).
- [16] H. A. A. Al-Kashoash and Andrew H. Kemp. “ Comparison of 6LoWPAN and LPWAN for the Internet of Things ”. In: *Australian Journal of Electrical and Electronics Engineering* 13.4 (Oct. 2016). 00010, pp. 268–274 (p. 37).
- [17] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. “ Low Power Wide Area Networks: An Overview ”. In: *IEEE Communications Surveys & Tutorials* 19.2 (22–2017). 00537, pp. 855–873 (p. 38).
- [18] Usman Sarwar. “ IoT Architecture : Elements of Connectivity Technologies ”. In: (2015). 00000, p. 23 (p. 39).
- [19] *Évaluation et Amélioration Des Plates-Formes Logicielles Pour Réseaux de Capteurs sans-Fil, Pour Optimiser La Qualité de Service et l'énergie*. 00000. URL: http://docnum.univ-lorraine.fr/public/DDOC_T_2016_0051_ROUSSEL.pdf (visited on 04/17/2019) (p. 35).
- [20] *Contiki Applications for Z1 Motes for 6LowPAN*. 00000. 2016. URL: <https://upcommons.upc.edu/bitstream/handle/2117/82767/Master%20Thesis.pdf?sequence=1&isAllowed=y> (visited on 01/14/2019) (p. 59, 60, 62, 64, 66, 68, 70).
- [21] Fayssal Bendaoud, Marwen Abdennabi, and Fedoua Didi. “ *Network Selection in Wireless Heterogeneous Networks: A Survey ”. In: *Journal of Telecommunications and Information Technology* 4 (Jan. 2019). 00000, pp. 64–74 (p. 74).
- [22] Atefeh Meshinchi. “ QOS-Aware and Status-Aware Adaptive Resource Allocation Framework in SDN-Based IOT Middleware ”. 00000. masters. École Polytechnique de Montréal, May 2018 (p. 74).
- [23] Abishi Chowdhury and Shital A. Raut. “ A Survey Study on Internet of Things Resource Management ”. In: *Journal of Network and Computer Applications* 120 (Oct. 15, 2018). 00002, pp. 42–60 (p. 74).
- [24] Rim Fakhfakh, Anis Ben, and Chokri Ben. “ Deep Learning-Based Recommendation: Current Issues and Challenges ”. In: *International Journal of Advanced Computer Science and Applications* 8.12 (2017). 00002 (p. 74).
- [25] Jitesh Shetty and Jafar Adibi. “ Discovering Important Nodes through Graph Entropy the Case of Enron Email Database ”. In: 00257. ACM Press, 2005, pp. 74–81 (p. 86).
- [26] Y. Huang, Y. Weng, and M. Zhou. “ Modular Design of Urban Traffic-Light Control Systems Based on Synchronized Timed Petri Nets ”. In: *IEEE Transactions on Intelligent Transportation Systems* 15.2 (Apr. 2014). 00050, pp. 530–539 (p. 94–96).
- [27] Alexandre David et al. “ Uppaal SMC Tutorial ”. In: *International Journal on Software Tools for Technology Transfer* 17.4 (Aug. 2015). 00176, pp. 397–415 (p. 94, 95).

- [28] A. Di Febbraro, N. Sacco, and D. Giglio. “ [On Using Petri Nets for Representing and Controlling Signalized Urban Areas: New Model and Results](#) ”. In: *2009 12th International IEEE Conference on Intelligent Transportation Systems*. 2009 12th International IEEE Conference on Intelligent Transportation Systems. 00018. Oct. 2009, pp. 1–8 (p. [95](#), [96](#)).
- [29] Mariagrazia Dotoli and Maria Pia Fanti. “ [An Urban Traffic Network Model via Coloured Timed Petri Nets](#) ”. In: *IFAC Proceedings Volumes*. 7th International Workshop on Discrete Event Systems (WODES'04), Reims, France, September 22-24, 2004 37.18 (Sept. 1, 2004). 00101, pp. 207–212 (p. [95](#)).
- [30] Michel dos Santos Soares and Jos Vrancken. “ [A Modular Petri Net to Modeling and Scenario Analysis of a Network of Road Traffic Signals](#) ”. In: *Control Engineering Practice*. Special Section: Wiener-Hammerstein System Identification Benchmark 20.11 (Nov. 1, 2012). 00017, pp. 1183–1194 (p. [95](#), [96](#)).
- [31] J. Alemany et al. “ [Estimation of Privacy Risk through Centrality Metrics](#) ”. In: *Future Generation Computer Systems* 82 (May 1, 2018). 00000, pp. 63–76 (p. [116](#)).