The impact of IoT communication protocols on efficient mobile crowdsensing

Nikolaos-Stylianos Loumis

Submitted for the Degree of Doctor of Philosophy from the University of Surrey

Institute for Communication Systems Faculty of Engineering and Physical Sciences University of Surrey Guildford, Surrey GU2 7XH, UK

November 2018

© Nikolaos-Stylianos Loumis 2018
Summary

The Internet of Things is not just a theoretical topic anymore, but it has become a reality that affects everyday life in many aspects. Users and devices can now be connected to the Internet whenever, wherever and interact with each other in revolutionary ways. Mobile Crowdsensing (MCS) is a collaborative demonstration of this interaction. However, MCS introduced a series of challenges surrounding the areas of power management, privacy preservation, and data quality. These may result in compromised user experience, limited user acceptance, and minimise the potential this area has to offer. This study addresses the discussed problems by examining the impact Internet of Things (IoT) communication protocols have on modern mobile crowdsensing systems. To do so, an end-to-end crowdsensing system is designed to evaluate the most popular IoT protocols. Simulation and off-the-shelf-device-based experiment runs provided an insight on the performance of the said protocols. Based on the findings, a new sensing approach is introduced aiming to improve system’s robustness and minimise energy requirements. Finally, a user-preference efficient crowdsensing algorithm that determines the most efficient communication protocol based on user input and experiment parameters is proposed.

Key words: Internet of Things, Mobile crowdsensing, Power consumption, Smartphones, Efficient.
Email: n.loumis@surrey.ac.uk
WWW: http://www.surrey.ac.uk/
Acknowledgments

First and foremost, I want to use this space to show my sincere appreciation to Ioannis Karavokyros M.D., PhD for his kindness, care, and sympathy demonstrated back in 2013. His selflessness throughout the following years was memorable. It is not an exaggeration to say that I would not be around to present this study, if it were not for him. These words might not be enough to fully reflect my gratitude.

I would like to thank my supervisor Prof Klaus Moessner not only for his motivation, knowledge, and guidance throughout my research, but also for providing me the opportunity to join his team as an intern 5 years ago. Unbeknownst to him, this opportunity triggered a domino effect that changed – for the better – the professional course of many people. Furthermore, I would like to express my appreciation to my co-supervisor Dr Francois Carrez for his key support in crucial times of this study. I also thank Dr Stylianos Georgoulas for his continuous help and guidance since the first day I arrived at the University of Surrey. I wish him all the best with his new chapter of life that started recently.

Besides my advisors, I need to give my appreciation to all the new colleagues and friends I made, Dr Niklas Palaghias, Dr Massimiliano Lehnus, Dr Konstantinos Katsaros, and the soon-to-be-Doctors Giannis Selinis, and Nikolaos Papachristou for their constant support and all the happy memories made. Thanks to my brother George, my cousin Constantinos, and to all my friends back home Kyriakos, Thanos, Akis, Christos, and Antonis for the countless laughs and stories we shared.

Furthermore, I need to thank my uncle Vassilis Loumis for giving away my first computer back in the mid-90s. Little did he know that this Intel 486 machine would be the spark that would ignite my passion for the computer science.

I cannot thank enough Dr Sylvia Agathou for everything she has done all these years. She has stood by me in countless occasions and helped me go through some really hard times. Her tolerance, support, spirit, and love are truly remarkable. I look forward to seeing where her new endeavour leads us to.

Last but definitely not least, I would like to acknowledge the unconditional support of my parents Lakis and Lena. The thought of them on “cloudy days” always feels like a crepuscular ray that brings warmth to my heart. Being apart is a constant reminder that one cannot have everything in life…
To all the missed-out memories...
# Table of Contents

**TABLE OF CONTENTS** ........................................................................................................ VI

**LIST OF FIGURES** ........................................................................................................... IX

**LIST OF TABLES** ............................................................................................................ XI

**GLOSSARY OF TERMS** ..................................................................................................... XII

## 1 INTRODUCTION ............................................................................................................. 1

1.1 MOTIVATION .................................................................................................................. 1

1.2 CHALLENGES ................................................................................................................. 3

1.3 RESEARCH PROBLEM ...................................................................................................... 5

1.4 SUMMARY OF CONTRIBUTIONS ..................................................................................... 6

1.5 PUBLICATIONS ................................................................................................................. 7

1.6 OUTLINE ........................................................................................................................... 8

1.7 CONCLUSION .................................................................................................................... 9

## 2 LITERATURE REVIEW .................................................................................................. 10

2.1 MOBILE CROWDSENSING ............................................................................................ 10

2.1.1 CURRENT STATE OF MOBILE SENSING ................................................................... 11

2.1.1.1 Full stack applications .......................................................................................... 14

2.1.1.2 Middleware applications ....................................................................................... 18

2.1.1.3 End nodes applications ......................................................................................... 18

2.1.1.4 Findings ................................................................................................................... 22

2.2 INTERNET OF THINGS (IOT) PROTOCOLS................................................................. 22

2.2.1 MESSAGE QUEUING TELEMETRY TRANSPORT ......................................................... 22

2.2.2 ADVANCED MESSAGE QUEUING PROTOCOL (AMQP) ........................................... 23

2.2.3 EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL (XMPP) ......................... 23

2.2.4 REPRESENTATIONAL STATE TRANSFER (REST) .................................................... 24

2.2.4.1 Hypertext Transfer Protocol (HTTP) ..................................................................... 24

2.2.4.2 Constrained Application Protocol (CoAP) ............................................................... 24

2.2.5 COMPARISON OF IOT PROTOCOLS .......................................................................... 25

2.3 DEFINITION OF KEY PERFORMANCE INDICATORS .................................................... 28

2.4 GAP ANALYSIS ................................................................................................................. 29

2.4.1 SAMPLING ..................................................................................................................... 29
2.4.2 Transmission ................................................................. 30
2.4.3 Storage ........................................................................ 30
2.5 Next steps ........................................................................ 31
2.6 Conclusion ........................................................................ 32

3 Overall System Definition .................................................. 33
3.1 System Components ............................................................ 33
3.1.1 Data Aggregation and Consumption Server ..................... 34
3.1.2 Connected Devices .......................................................... 35
3.1.3 Components Coordination ............................................... 35
3.1.4 System Resources ........................................................... 37
3.2 Experimental Set-up ............................................................. 38
3.3 Data Transmitted ................................................................. 39
3.4 Experiment Duration ............................................................ 39
3.5 Experiment Sequence .......................................................... 39
3.6 Conclusion ......................................................................... 41

4 Simulation-Based Evaluation ................................................ 42
4.1 Internet of Things (IoT) Simulators ....................................... 42
4.2 Simulator Adjustments .......................................................... 45
4.3 Sensing Parameters .............................................................. 46
4.4 Simulator Results ................................................................. 46
4.5 Discussion ......................................................................... 48
4.6 Conclusion ......................................................................... 49

5 Implementation-Based Evaluation .......................................... 50
5.1 Crowdsourcing Devices in Your Pocket ................................. 50
5.1.1 Crowdsourcing Software ................................................. 51
5.1.1.1 User Profile ............................................................... 51
5.1.1.2 Google Cloud Messaging .......................................... 54
5.1.1.3 A Hybrid Protocol ..................................................... 55
5.1.1.4 Privacy by Design ...................................................... 55
5.1.1.5 Additional Features ................................................... 58
5.1.1.6 Dissemination of the Crowdsourcing Tool in the World 62
5.1.2 Assumptions ................................................................. 63
5.2 Monitoring Energy Consumption ......................................... 64
5.2.1 Hardware Solutions ....................................................... 64
5.2.2 Software Solutions ....................................................... 64
5.3 Conclusion ......................................................................... 67
6 RESULTS AND DISCUSSION ................................................................. 68

6.1 SUMMARISED EVALUATION REVIEW ........................................... 68
  6.1.1 IDLE STATE .................................................................................. 69
  6.1.2 GCM ......................................................................................... 73
  6.1.3 MQTT ....................................................................................... 75
  6.1.4 REST ......................................................................................... 76
  6.1.5 RABBITMQ ............................................................................... 77
  6.1.6 RABBITMQ + GCM ................................................................. 78
6.2 A NEW SENSING APPROACH ....................................................... 80
  6.2.1 EVALUATION OF THE NEW APPROACH ............................... 82
6.3 DISCUSSION .................................................................................. 92
  6.3.1 POWER TRENDS JUSTIFICATION ........................................... 92
  6.3.2 BATTERY LIFE ......................................................................... 94
6.4 IMPACT AND CONCLUSION ....................................................... 96

7 OPTIMAL PROTOCOL SELECTION ON MCS SYSTEMS ......................... 99

7.1 DYNAMIC CROWDSENSING EXPERIMENTS .................................. 99
7.2 MULTI-OBJECTIVE OPTIMISATION .............................................. 100
  7.2.1 WEIGHT ASSIGNMENT ......................................................... 101
  7.2.2 USER-PREFERENCE EFFICIENT CROWDSENSING ALGORITHM ..... 103
  7.2.3 ADAPTING USER-PREFERENCE EFFICIENT CROWDSENSING ALGORITHM 104
7.3 ARTIFICIAL PROBLEMS .................................................................. 106
  7.3.1 ARTIFICIAL PROBLEMS COMPOSITION ............................... 106
  7.3.2 EVALUATION OF UPECA ....................................................... 110
    7.3.2.1 DURATION D = 5 MINUTES .............................................. 111
    7.3.2.2 DURATION D = 10 MINUTES ............................................ 113
    7.3.2.3 DURATION D = 20 MINUTES ............................................ 115
  7.3.3 RESULTS AND DISCUSSION ................................................. 117
7.4 CONCLUSION ................................................................................. 117

8 CONCLUSION .................................................................................. 119

8.1 CLOSING REMARKS .................................................................... 119
8.2 FUTURE WORK ............................................................................ 121

9 REFERENCES .................................................................................. 122

10 APPENDIX ...................................................................................... 132
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Current challenges in Mobile Crowdsensing</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>The steps of a crowdsensing experiment</td>
<td>29</td>
</tr>
<tr>
<td>3</td>
<td>Generic architecture of IoT experiments</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>Architecture of IoT experiments using AMQP, MQTT, and XMPP</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>Architecture of IoT experiments using REST</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>Setting experiment parameters using DAC server’s UI</td>
<td>39</td>
</tr>
<tr>
<td>7</td>
<td>Message sequence during XMPP, AMQP, and MQTT experiments</td>
<td>40</td>
</tr>
<tr>
<td>8</td>
<td>Message sequence during REST experiments</td>
<td>41</td>
</tr>
<tr>
<td>9</td>
<td>REST Message delivery trend for devices in use d=50 and d=100</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>Exporting a socio-economic profile using IoTLab app.</td>
<td>52</td>
</tr>
<tr>
<td>11</td>
<td>Confirmation dialog</td>
<td>56</td>
</tr>
<tr>
<td>12</td>
<td>Initial state of the app</td>
<td>56</td>
</tr>
<tr>
<td>13</td>
<td>User data mapped to a region of 500m radius</td>
<td>56</td>
</tr>
<tr>
<td>14</td>
<td>Service notification</td>
<td>57</td>
</tr>
<tr>
<td>15</td>
<td>Service notification expanded</td>
<td>57</td>
</tr>
<tr>
<td>16</td>
<td>Active location services on Android OS</td>
<td>57</td>
</tr>
<tr>
<td>17</td>
<td>User profile</td>
<td>58</td>
</tr>
<tr>
<td>18</td>
<td>Research Idea Proposal</td>
<td>59</td>
</tr>
<tr>
<td>19</td>
<td>List of available ideas</td>
<td>60</td>
</tr>
<tr>
<td>20</td>
<td>Research Idea Details and Ranking</td>
<td>60</td>
</tr>
<tr>
<td>21</td>
<td>System notifications during experiments</td>
<td>61</td>
</tr>
<tr>
<td>22</td>
<td>Resources projected on a Map</td>
<td>62</td>
</tr>
<tr>
<td>23</td>
<td>Global outreach of our crowdsourcing tool</td>
<td>62</td>
</tr>
<tr>
<td>24</td>
<td>Energy consumption per protocol during idle state</td>
<td>70</td>
</tr>
<tr>
<td>25</td>
<td>Energy consumption per protocol during light-load sensing</td>
<td>70</td>
</tr>
<tr>
<td>26</td>
<td>Message delivery per protocol during light-load sensing</td>
<td>71</td>
</tr>
<tr>
<td>27</td>
<td>Energy consumption per protocol during medium-load sensing</td>
<td>71</td>
</tr>
<tr>
<td>28</td>
<td>Message delivery per protocol during medium-load sensing</td>
<td>72</td>
</tr>
<tr>
<td>29</td>
<td>Energy consumption per protocol during heavy-load sensing</td>
<td>72</td>
</tr>
<tr>
<td>30</td>
<td>Message delivery per protocol during heavy-load sensing</td>
<td>73</td>
</tr>
<tr>
<td>31</td>
<td>GCM message delivery trend</td>
<td>74</td>
</tr>
<tr>
<td>32</td>
<td>GCM energy consumption trend</td>
<td>74</td>
</tr>
<tr>
<td>33</td>
<td>MQTT message delivery trend</td>
<td>75</td>
</tr>
</tbody>
</table>
Figure 34: MQTT energy consumption trend ................................................................. 75
Figure 35: REST message delivery trend ................................................................. 76
Figure 36: REST energy consumption trend .............................................................. 77
Figure 37: RabbitMQ message delivery trend ............................................................ 77
Figure 38: RabbitMQ energy consumption trend ....................................................... 78
Figure 39: RabbitMQ + GCM message delivery trend ................................................ 79
Figure 40: RabbitMQ + GCM energy consumption trend .......................................... 79
Figure 41: Message sequence during triggered XMPP, AMQP, and MQTT experiments .... 80
Figure 42: Message sequence during triggered REST experiments ............................ 81
Figure 43: Energy consumption per protocol during light-load triggered sensing .......... 82
Figure 44: Message delivery per protocol during light-load triggered sensing .............. 83
Figure 45: Energy consumption per protocol during medium-load triggered sensing ...... 83
Figure 46: Message delivery per protocol during medium-load triggered sensing .......... 84
Figure 47: Energy consumption per protocol during heavy-load triggered sensing ......... 85
Figure 48: Message delivery per protocol during heavy-load triggered sensing .......... 85
Figure 49: GCM message delivery trend (triggered sensing) ...................................... 86
Figure 50: GCM energy consumption trend (triggered sensing) .................................. 86
Figure 51: MQTT message delivery trend (triggered sensing) .................................... 87
Figure 52: MQTT energy consumption trend (triggered sensing) ............................... 87
Figure 53: REST message delivery trend (triggered sensing) ...................................... 88
Figure 54: REST energy consumption trend (triggered sensing) .................................. 89
Figure 55: RabbitMQ message delivery trend (triggered sensing) ............................... 89
Figure 56: RabbitMQ energy consumption trend (triggered sensing) .......................... 90
Figure 57: RabbitMQ + FC message delivery trend (triggered sensing) ..................... 91
Figure 58: RabbitMQ + GCM energy consumption trend (triggered sensing) ............ 91
Figure 59: Power trend per protocol on burst scenario .............................................. 92
Figure 60: Power trend per protocol on triggered scenario ........................................ 93
Figure 61: Idle windows between triggers during (a) medium and (b) high-load scenarios .... 93
Figure 62: Discharge time per sensing-load and protocol (burst type) ......................... 95
Figure 63: Discharge time per sensing-load and protocol (triggered type) .................... 96
Figure 64: Pseudocode for UPECA .......................................................................... 104
Figure 65: Pseudocode for the adapting UPECA ...................................................... 105
Figure 66: Pseudocode for the Artificial problems engine .......................................... 106
List of Tables

Table 1: State-of-the-art in Mobile Crowdsensing ........................................................... 12
Table 2: Comparison of IoT protocols ........................................................................... 25
Table 3: Registering a new crowdsourcing resource ..................................................... 37
Table 4: Representation of a sensor in the resource index ........................................... 38
Table 5: Simulation-based evaluation parameters ......................................................... 46
Table 6: Message delivery percentages for devices d=1 .............................................. 47
Table 7: Message delivery percentages for devices d=50 ............................................ 47
Table 8: Message delivery percentages for devices d=100 ............................................. 48
Table 9: Smart mobile devices used ............................................................................. 50
Table 10: Registration of user's socio-economic profile .............................................. 52
Table 11: Updating of user's socio-economic profile .................................................... 53
Table 12: Median Energy consumption during preliminary and generalisation runs ........ 69
Table 13: Maximum continuous sensing per sensing-load and protocol (burst type) ........ 98
Table 14: Maximum continuous sensing per sensing-load and protocol (triggered type) .... 98
Table 15: Output of the Artificial Problems Engine for d= {5, 10, 20} ......................... 107
Table 16: Generated weights mapped to the artificial problems .................................... 110
Glossary of Terms

ACK Acknowledge
ADB Android Debug Bridge
AMQP Advanced Message Queuing Protocol
APE Artificial Problems Engine
API Application Programming Interface
BLE Bluetooth Low Energy
CoAP Constrained Application Protocol
CPU Central Processing Unit
cURL Client URL
DAC Data Aggregation and Consumption
DAS Direct Assignment Strategy
DTLS Datagram Transport Layer Security
EU European Union
FP7 7th Framework Programme for Research and Technological Development
GCM Google Cloud Messaging
GDPR General Data Protection Regulation
GPS Global Positioning System
GPU Graphics Processing Unit
HTTP Hypertext Transfer Protocol
IDE Integrated Development Environment
IoT Internet of Things
IP Internet Protocol
IPSO Internet Protocol for the Networking of Smart Objects
JSON JavaScript Object Notation
KPI Key Performance Indicator
LoRa Long Range
LTE Long-Term Evolution
LWM2M Lightweight M2M
M2M Machine-to-Machine
MCDM Multiple-Criteria Decision-Making
MCS Mobile Crowdsensing
MEMS Micro-Electro-Mechanical Systems
MGRS Military Grid Reference System
MQTT Message Queuing Telemetry Transport
OS Operating System
OSI Open Systems Interconnection
PUBACK Publish Acknowledgment
<table>
<thead>
<tr>
<th><strong>Abbr</strong></th>
<th><strong>Full Form</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>QoI</td>
<td>Quality of Information</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAS</td>
<td>Random Assignment Strategy</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio-Frequency IDentification</td>
</tr>
<tr>
<td>RSSI</td>
<td>Received Signal Strength Indication</td>
</tr>
<tr>
<td>SASL</td>
<td>Simple Authentication and Security Layer</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UniS</td>
<td>University of Surrey</td>
</tr>
<tr>
<td>UPECA</td>
<td>User-Preferece Efficient Crowdsensing Algorithm</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>V2V</td>
<td>Vehicle-to-vehicle</td>
</tr>
<tr>
<td>V2X</td>
<td>Vehicle-to-everything</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity, <em>IEEE 802.11x</em></td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Network</td>
</tr>
<tr>
<td>XMPP</td>
<td>eXtensible Messaging and Presence Protocol</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

In 1965, Dr Gordon Moore stated that for the following ten years -at least- the number of transistors per square inch on integrated circuits will double yearly [1]. Little did he know that more than 50 years later the Moore’s law would still dictate the growth of the state-of-the-art microchips. This exponential technological advancement that has been characterising the past decades led naturally to the present day, where multi-core devices such as smartphones, tablets, wearables, and cameras offer ubiquitous computing capabilities to the end users, with continuously decreasing cost. Additionally, the endless evolution of micro-electro-mechanical systems (MEMS) have drastically transformed the dimension and accuracy of modern sensors and consequently, their integration potential. Nowadays, complex sensors are embedded in common devices and enhance user experience. Cutting-edge portable devices are equipped with large and vivid panels, staggering photographic sensors, integrated global positioning systems (GPS) receivers, multiple communication interfaces, and a plethora of context-aware sensors.

Meanwhile, broadband Internet was an additional field which was advanced thanks to the aforementioned technological advancements. According to the Internet World Stats and the International Tele-communication Union, there are countries or regions in the Western world that enjoy up to 97% broadband internet penetration [2]. Furthermore, in the majority of developing countries, mobile broadband is now less expensive that the fixed one, as a result of the price-cut noted in the time between 2013 and 2016 [3]. Thus, users and devices can be connected ubiquitously, leveraging on the accessibility and affordability that portray modern broadband connections. Internet of Things (IoT) is a topic that came naturally from this seamless connectivity and has gained a lot of traction during the past years. By generating, sharing, and accessing information provided by others, smart and dumb devices, actuators, and sensors become part of intelligence-enhanced networks. These networks paved the way to develop situation-aware
applications and services. It comes as no surprise that the IoT has already been adopted in a wide spectrum of areas including:

- **Smart home**: This umbrella term includes everything from colour-changing light bulbs and wireless-triggered switches all the way to smart energy meters and thermostats. Smart home is the most adopted application of the IoT and technology giants such as Google, Amazon, and Apple have released a suite of products.

- **Wearables**: Smart watches, activity trackers, smart glasses, and on-body sensors becoming increasingly popular, thanks to the evolution of their capabilities.

- **Smart City**: Numerous cities in the world have leveraged on the power of IoT to upgrade the quality of life of their citizens. Common applications include urban security, traffic management, telematics, and noise and pollution reduction.

- **Smart grid**: by installing smart meters and sensors, energy providers can now monitor and predict the power demands of the public. In that way, they can improve the efficiency of their network, while providing more reliable services.

- **Connected vehicles**: Recent vehicles come with a plethora of sensors and actuators that not only enhance the driver’s experience, but also provide additional safety layers. Moreover, vehicle-to-vehicle (V2V) and vehicle-to-everything (V2X) communications will be key elements of the upcoming fifth generation (5G) of cellular mobile communications.

- **Connected health**: Internet of Things helps physicians to monitor and treat patients in a faster and more efficient way. Additionally, modern IoT devices facilitate telemedicine more than ever. Despite its great potential, connected health is still hindered by privacy concerns.

- **Smart farming**: the full agricultural cycle can benefit from using smart connected devices and actuators. Soil and air sensors provide information and offer recommendation to farmers concerning appropriate crops and seeds varieties. GPS-enabled ploughing equipment can keep track of planting even in vast areas. Monitoring drones identify in a cost and time-efficient manner the best time to harvest. Finally, smart storage silos offer real-time data on the status of the grain.
1. Introduction

It is undeniable that the IoT has helped businesses, organisations, and entities to minimise their costs while augmenting automation and productivity. Similarly, end users are offered novel ways to interact with their surrounding environment and people.

Those aspects were not left unnoticed by the research community, who made the most of the novel interaction techniques and the advantages that come with the mobility offered by modern IoT devices. Researchers do not need to rely on proprietary equipment to perform sensing experiments in wide areas, as they can be adequately replaced by smart devices owned by the crowd.

Mobile crowdsensing (MCS) is a term that was introduced by Ganti [4] which defines a technique where a large group of users collect and share data in order to extract information for a common goal, using mobile devices. Over the last years, crowdsensing has been continuously gaining attention and it comes with no surprise that it is currently under the research spotlight. Retrieving and storing information generated from numerous personal devices replaced specialised equipment, minimised cost, and offered access to wider areas for experiments [5], [6].

One way to classify crowdsensing applications is based on their sensing approach to a) opportunistic or b) participatory. On the former, sensing is happening in the background without user’s involvement. This includes data generated by embedded sensors and stats concerning the use of a device, amongst others. On the latter, one must be active and provide the necessary information or data required, such as photographs, annotations, and surveys. A famous manifestation of both types of sensing is Waze¹, which is a smartphone application that provides real-time traffic maps that are built using data captured by user devices. Collecting sensor data such as GPS coordinates, acceleration, and orientation the application can infer the traffic at a specific segment of a road. Furthermore, users can provide additional information (e.g. accidents, roadblocks) to further support the generated traffic maps.

1.2 Challenges

Even though mobile sensing is without a doubt promising, it is facing a series of obstacles that arise from its two main parameters: devices and users. Mobile energy efficiency is a challenge associated with the increasing sensing capabilities, computational power, and use of mobile devices. Seamless sensing and background data processing added additional energy burdens to the already energy-constrained mobile devices. Consequently, smart devices are not exploited at their full potential by

¹ https://www.waze.com/en-GB/
end users, as there is a constant struggle to extend battery life between charging cycles. Additionally, processes that engage the central processing unit (CPU), graphics processing unit (GPU), and embedded sensors raise the temperature of the system and can lead to serious implications to the battery stability. Portable IoT devices are not designed to sustain such demanding tasks, hence they resort to throttling to tackle the heat issue. All these result in compromised user experience [7]–[9].

On the other hand, since crowdsensing experiments by definition engage users to generate and collect data, privacy concerns emerge [10]. As discussed, modern devices are equipped with a plethora of ambient, environmental, and inertial sensors that could generate data and assess a user’s context [11], [12]. Individuals can collect these sensitive data and extract knowledge in order to exploit them in malicious ways. For example, one can monitor data generated from GPS sensors to infer mobility patterns and orchestrate burglaries. Likewise, health status can be easily concluded by combining wearable sensors together with data gathered from applications that track fitness level and dietary habits.

In some cases, however, issues during sensing scenarios appear as a combination of the performance of both users and devices [13]. More specifically, manufacturers, in order to minimise the production cost, do not equip low and medium-end IoT devices with expensive top-tier hardware. Meanwhile, embedded sensors can be called to participate to crowdsourcing experiments at any time and consequently provide data of mediocre quality for context reasons. For instance, a scenario might require gathering audio level data from a smartphone that is placed inside a pocket. Another root of quality problems are malicious users who provide wrong or fabricated data to either contaminate the results of an experiment or, because their sole purpose is to successfully complete a crowdsourcing task for its incentives [10], [14].

User acceptance and engagement on mobile crowdsensing experiments is closely associated with the three main challenges illustrated in Figure 1 [8], [15]–[17]. For that reason, researchers and businesses have been trying to tackle them. Nonetheless, one must not forget that despite that crowdsensing experimenters and participants are part of the same system, their priorities and goals are not aligned. The former are attempting to satisfy specific criteria in terms of number of participants, type of data captured, and data quality. Additionally, they are constantly struggling to keep system’s monetary and energy cost low, while preserving the level of privacy and security set either by legislation or by stakeholders.
On the other hand, participants might have no interest in the greater picture of a crowdsensing experiment, but rather in the aspects that are closely related to them. Despite that, there are ways to make the two ends of this system come together. One approach includes offering incentives to participants to take place in available experiments. Those could be in the form of money, discount coupons, experiences, or providing to charities. Moreover, increased incentives can also persuade participants to provide data of high quality or, compensate them for the compromised user experience. However, the overall cost of a crowdsensing experiment escalates by granting extensive rewards to its participants. Another technique used to motivate people to contribute is gamification. This includes a set of virtual goals that a user can achieve based on her participation frequency and the quality of data provided. These goals can be virtual badges, rankings, and score points.

1.3 Research problem

It is evident that for mobile crowdsensing to grow and become more efficient, it has to be set free from the trammels presented in the previous section. For this to happen, one must inspect closely the troubling aspects before addressing them. When doing so, the following questions arise:

- Can mobile devices perform reliable long-term crowdsensing experiments given their constrained nature?

- How can crowdsensing applications be optimised to alleviate the battery effects to the devices used?
- To what extent are mobile devices capable of dealing with the computational burden and energy consumption required by monitoring applications in a seamless manner?

- What improvements must be applied to current crowdsourcing tools in order to enhance user experience?

- What is the impact of communication protocols during crowdsensing experiments? If it is significant, which one should be used? Under what circumstances?

### 1.4 Summary of contributions

The main objective of this work is to investigate the impact that mobile crowdsensing experiments have on devices and users. Based on this investigation, one can propose and evaluate innovative mechanisms that will eventually render MCS more efficient and raise user acceptance. The contributions of this study are aligned with the research challenges of the area and can be listed as follows:

- Development and evaluation of a system that is able to simulate crowdsensing experiments using a variety of IoT communication protocols (Chapter 3).

- Performance assessment of IoT communication protocols under various experimental circumstances and scalability levels using the aforementioned system (Chapter 4).

- Evaluation of IoT communication protocols using off-the-shelf devices and comparison between the results acquired from simulation and experiments (Chapter 5).

- Design and development of a mobile tool that ensures user’s privacy and enables researchers to conduct complex crowdsensing experiments, while offering monitoring capabilities (Chapter 5).

- Development and evaluation of a novel technique for conducting mobile crowdsensing experiments, which alleviates the reliability issues that govern current state-of-the-art systems (Chapter 6).

- Design, development, and evaluation of a novel mechanism for mobile crowdsensing devices that determines the most efficient IoT communication protocol based on a series of criteria (Chapter 7).
1.5 Publications

Book chapters


Conferences


Part of this study was motivated by the author’s engagement to the European Union (EU) 7th Framework Programme for Research and Technological Development (FP7) funded research project “IoTLab” that run from October 2013 until September 2016. Some parts of the software designed and implemented for IoTLab are used as they are in this study, while others were extended. Publications B.1, B.2, C.1, and C.3 are directly related to this dissertation and more specifically, to Chapters 2, 3, and 5 correspondingly. Publication C. 2 emerged from the author’s partial engagement to the EU FP7 research project “SocIoTal” and is not linked directly to this study.
1.6 Outline

The remainder of this thesis is organised as follows:

Chapter 2 surveys the existing literature in Mobile Sensing and Crowdsensing, including the state-of-the-art power-saving approaches. Furthermore, it identifies the established IoT communication protocols and offers a qualitative comparison.

Chapter 3 introduces and defines the system that used to examine the performance of the selected IoT protocols. Overall architecture, end nodes, implemented server and coordination components are specified. Additionally, this chapter covers all the parameters associated with the experimental set-up.

Chapter 4 consists of the simulation-based evaluation performed on the four IoT communication protocols and variations. It starts with a review of existing IoT simulation tools, followed by an analysis of the design considerations of the selected tool. Furthermore, it presents and discusses the results of the simulations.

Chapter 5 extends the simulations to everyday devices by utilising off-the-shelf smartphones. A novel powerful crowdsourcing tool is introduced and extensively analysed, followed by a review of existing energy consumption monitoring tools.

Chapter 6 presents in scrutiny the outcome of the smartphone-based crowdsensing experiments. The analysis covers both the monitored energy consumption of the devices and the message delivery success, starting from an idle state all the way to the heavy-load sensing scenario. Based on the results obtained, we introduce a new crowdsensing approach that promises enhanced robustness compared with the existing solutions. This chapter continues with a new series of experiment results based on the novel sensing suggestion, followed by a discussion and comparison of the two approaches.

Chapter 7 discusses the dynamic nature of modern crowdsensing experiments. It argues the importance of having adaptable crowdsensing tools to counterbalance the constantly evolving experiment scene. It introduces a method that calculates the best protocol per scenario based on user preferences and system parameters. To do so, a mechanism that maps user preferences to weights is also proposed. This chapter concludes by examining the effect of the user-preference efficient crowdsensing algorithm in real life scenarios.

Chapter 8 provides a conclusion of this research and discusses the proposed future work.
At this point it is important to highlight the use of footnotes and references in throughout this study. More specifically, the former are used when the author wants to provide additional reading material on a topic that might not be widely known to a reader. The latter are used when the author needs to support statements he made or when a specific work is mentioned directly.

1.7 Conclusion

The first chapter of this study covers the technological evolution of the past century and how it led to current advancements. The author introduces the concept of the IoT and a few widely adopted use cases derived from it. Among them is mobile crowdsensing which despite its promising nature it introduces challenges that are still open and trouble users and experimenters. Then, the research problem is composed followed by the contributions that are presented in this document. The next chapter conducts a literature review on the current state of mobile sensing, the most popular IoT communication protocols and discusses the research gaps found from the said review.
Chapter 2

Literature Review

The goal of this chapter is to provide an overview of the state-of-the-art of Mobile Crowd Sensing (MCS). It begins with an analysis on current MCS systems and presents the methods proposed by the researchers in an attempt to tackle the issues affecting the area. Later on, it dives into the more technical parts of a MCS system, by analysing and comparing the most popular Internet of Things communication protocols. Finally, it identifies the research gaps that this research focuses on.

2.1 Mobile Crowdsensing

High-speed reliable broadband Internet connections are nowadays a reality which enables users and devices to be connected at anytime and anyplace. “Internet of Things” (IoT) is a topic that emerged from this seamless connectivity which is offered by current technology. By producing and uploading their data and by accessing information generated by others, devices and things are enhanced with intelligence. This leads to a network of smart devices that communicate with each other and paves the path to develop situation-aware applications and services.

As presented in [12] and [18] with scrutiny, mobile smart devices are currently used in a variety of ways with one target: extraction and observation of user’s context. Benefiting from the appropriate models, a device can recognise if a user is walking, running, or even having social interactions just by exploiting a series of data captured from embedded sensors like compass, accelerometer, microphone, and gyroscope. In recent years, researchers have been more enthusiastic about the emerging opportunistic sensing and context aware systems. Mobile crowdsensing (MCS) is a term that characterises applications which emerged from the aforementioned trend. MCS was defined as a term by Ganti when he surveyed all the available – at the time – systems which extracted information by collecting and sharing data [4]. It is undisputable that mobile crowdsensing offers a novel and interactive way of solving problems, through an open call, leveraging on the power that emerges from a crowd.
Since the dawn of Web 2.0, crowdsourcing has been gaining increasing attention and not only from the research community. Initiatives such as SETI@home [19], Openstreet maps [20], and Open Signal² proved to researchers and businesses that the concept of crowdsourcing/crowdsensing appeals positively to general population and is worth perusing further.

Several researchers have attempted to present, categorise, and compare existing mobile crowdsensing solutions. Studies such as [4], [10], [16], [18], [21]–[25][26] offer a wide spectrum of classifications which are performed from the point of view of:

- **Participation**: participatory or opportunistic
- **Privacy level**: personal, social, public
- **Tasks**: creation, assignment, execution, and upload as the main phases of the experiments
- **Incentives**: entertainment, services, monetary
- **Problem target**: health, environment, crime prevention etc

### 2.1.1 Current state of mobile sensing

Chapter 1 scratches the surface of the challenges that current mobile sensing systems are facing. Hence, the author believes that a detailed survey needs to be conducted on the current applications and solutions. This will allow us to detect research gaps that will alleviate the existing energy, privacy, and data quality problems. This section thoroughly inspects the area of mobile sensing.

Table 1 presents a plethora of modern mobile sensing systems introduced by the research community. These applications can be either targeted to a single area of the sensing lifecycle (i.e. data sampling, data transmission) or introduce an end-to-end solution. Additionally, this Table summarises the type of data that each system can provide (e.g. location, temperature, or generic when data are not sensor-generated), the communication protocol, and the interface used to transmit the said data. Finally, it presents information on whether a system has implemented any measures to mitigate the energy and privacy challenges discussed in Section 1.2.

Following Table 1, this section examines in detail all the presented solutions and groups them based on their scope into: *End node, Middleware, and Full stack* applications.

---

² https://opensignal.com/
### 2. Literature Review

Table 1: State-of-the-art in Mobile Crowdsensing.

<table>
<thead>
<tr>
<th>System</th>
<th>Technology</th>
<th>Type of Sensor</th>
<th>Communication Interface</th>
<th>Energy Efficient</th>
<th>Privacy awareness</th>
</tr>
</thead>
<tbody>
<tr>
<td>AnonySense [6]</td>
<td>HTTP</td>
<td>Microphone, Bluetooth, Wi-Fi</td>
<td>Wi-Fi</td>
<td>Low footprint language</td>
<td>X</td>
</tr>
<tr>
<td>BX Tracker [27]</td>
<td>HTTP</td>
<td>GPS, accelerometer, activity, cellular network</td>
<td>Wi-Fi, Cellular</td>
<td>Minimising GPS use</td>
<td>X</td>
</tr>
<tr>
<td>CARROM [26]</td>
<td>HTTP</td>
<td>Accelerometer, light, magnetometer, temperature, gyroscope, GPS</td>
<td>Wi-Fi, Cellular</td>
<td>Minimising data uploading via data mining</td>
<td></td>
</tr>
<tr>
<td>CenceMe [28]</td>
<td>HTTP</td>
<td>Microphone, accelerometer, GPS, Bluetooth, camera</td>
<td>Wi-Fi, Cellular</td>
<td>Pre-processing of data, power-aware sensing cycles</td>
<td></td>
</tr>
<tr>
<td>Crowd++ [29]</td>
<td>N/A</td>
<td>Microphone</td>
<td>N/A</td>
<td>Adapting sensing time</td>
<td>X</td>
</tr>
<tr>
<td>CrowdSense@Place [30]</td>
<td>N/A</td>
<td>Microphone, camera, GPS, Wi-Fi</td>
<td>Wi-Fi, Cellular</td>
<td>Prioritise uploading via Wi-Fi while line-powered</td>
<td>X</td>
</tr>
<tr>
<td>CUPUS [31]</td>
<td>TCP/IP, GCM</td>
<td>Generic</td>
<td>Wi-Fi, Cellular</td>
<td>Selective sampling</td>
<td>X</td>
</tr>
<tr>
<td>Ear-Phone [32]</td>
<td>HTTP</td>
<td>Microphone, GPS, time</td>
<td>Wi-Fi, Cellular</td>
<td>Reduces communications</td>
<td></td>
</tr>
<tr>
<td>EEMSS [8]</td>
<td>N/A</td>
<td>GPS, Wi-Fi, accelerometer, microphone</td>
<td>N/A</td>
<td>Adapting sensing cycles, use of less demanding sensors</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>Interf.</td>
<td>Sensing Module</td>
<td>Comm. Module</td>
<td>Offloading Strategy</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>----------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>effSense [33]</td>
<td>N/A</td>
<td>Generic</td>
<td>Wi-Fi, Cellular, Bluetooth</td>
<td>Offloading techniques, use of less demanding interface</td>
<td></td>
</tr>
<tr>
<td>EMC³ [34]</td>
<td>N/A</td>
<td>Generic</td>
<td>Cellular</td>
<td>Task allocation based on usage and location models</td>
<td></td>
</tr>
<tr>
<td>FindingNemo [35]</td>
<td>Kafka</td>
<td>GPS, Bluetooth</td>
<td>Wi-Fi, Cellular</td>
<td>Choice between GPS or BLE</td>
<td></td>
</tr>
<tr>
<td>McSense [5]</td>
<td>HTTP</td>
<td>Generic</td>
<td>Wi-Fi, Cellular</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>Piggyback Crowdsensing</td>
<td>HTTP</td>
<td>GPS, camera, microphone, accelerometer</td>
<td>Wi-Fi, Cellular</td>
<td>Sampling and uploading based on usage models</td>
<td></td>
</tr>
<tr>
<td>Portolan [37]</td>
<td>HTTP, GCM</td>
<td>GPS, Network properties</td>
<td>Wi-Fi</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>QoI-aware participant selection [38]</td>
<td>N/A</td>
<td>Generic, Battery level</td>
<td>N/A</td>
<td>Adapting sensing frequency</td>
<td></td>
</tr>
<tr>
<td>SmartRoad [13]</td>
<td>HTTP</td>
<td>GPS, battery stats</td>
<td>Wi-Fi, Cellular</td>
<td>Limiting data transmissions, sensing happens when device has enough energy</td>
<td></td>
</tr>
<tr>
<td>TailEnder [39]</td>
<td>HTTP</td>
<td>Generic</td>
<td>Wi-Fi, Cellular</td>
<td>Transmission scheduling, data prefetching</td>
<td></td>
</tr>
</tbody>
</table>
2. Literature Review

2.1.1 Full stack applications

**effSense**
In [33] a crowdsensing framework designed to tackle both energy and bandwidth consumption is presented. This study grouped participants to data-plan and non-data-plan users aiming to alleviate energy and cost demands correspondingly. Based on an algorithm that takes under consideration critical events (such as phone calls, server/user encounters), data cost, energy consumption, and future event probabilities, **effSense** decides whether a device should – or not – offload crowd-sensed data to nearby devices, bluetooth gateways, Wi-Fi access points, or piggyback them during phone calls. This framework proposes two uploading schemes for each category of users: a) the cold-start and b) the prediction-based scheme. The former one does not require any historical data of the user and attempts to offload data the moment a critical event occurs, regardless of future potential encounters. The latter scheme computes and compares the cost of uploading the gathered data during an ongoing critical event with a future-predicted-one and chooses accordingly. The system was evaluated against two real-world datasets reducing energy consumption by 55%-65% and 48%-52%, for data-plan and non-data-plan users respectively, compared to conventional uploading schemes. However, the presented energy consumption measurements did not examine the additional energy burden imposed by the necessary bluetooth scanning, but just the actual data uploading. Furthermore, due to the proposed 24-hour-long uploading cycle, effSense should not be considered appropriate for experiments that require real-time generated data.

**CrowdSense@Place**
Chon, Lane et al. presented a framework that is able to categorise places using opportunistic sensing on mobile devices [30]. **CrowdSense@Place** collects photos and audio clips during predefined events, such as phone calls and web browsing, and uploads them to a cloud-based application server for further analysis. Furthermore, those data are also tagged with location information gathered from Wi-Fi fingerprinting. Then, a total of five classifiers attempt to categorise a place by performing character, object, indoor scene, and speech recognition to the acquired image and audio data. The authors performed a large-scale experiment run across 5 different cities in 2 different continents. Results indicated that despite receiving a lot of unusable data (blurry images, too noisy sound samples etc), their trained models can automatically assort places into 7 different classifications with accuracy of 69%. CrowdSense@Place is intended for non-time-crucial experiments as the upload period of the samples equals to 24 hours. This is because researchers want to be sure that the user has enough time to decide if she wants to share the generated pieces of information or, to delete them. Authors also offer an energy-efficient upload strategy which prioritises offloading via the Wi-Fi interface and while the smartphones are charging.
**Piggyback CrowdSensing**

*Piggyback CrowdSensing (PCS)* is an end-to-end crowdsourcing system developed by Lane *et al.* and presented in [36]. Its primary focus is to minimise the energy consumption during mobile crowdsensing scenarios throughout the sampling, data processing, and uploading phases. PCS’s success relies on the powerful mobility and mobile-usage models build for each participant. By doing so, the PCS crowdsensing mobile application can predict the device’s use and judge whether it is wise to collect measurements or wait for future critical events. Through a series of extensive experiments, the researchers concluded that the energy overheads of power demanding tasks, such as sensor sampling and data uploading, could be significantly lowered when they are happening during critical events such as phone calls, or application fidgeting. This is because the PCS application is taking advantage of the awake state of the CPU that those events enforce. Based on the literature and their finding, an awake CPU needs less power compared to another one which switches between the *idle* and *wake* computational states. Similarly, using a communication interface that was activated by another application can drastically reduce the energy consumption of uploading data. Nevertheless, in order to build the aforementioned mobility models, power consuming operations such as sensor fusion and localisation via positioning sensors are mandatory. The authors stress the need to find the balance between constant location-awareness and energy consumption. By default, all data sampling and uploading are happening while the smartphones are connected to the Internet via Wi-Fi and only if they are charging. However, users can change the power options of the applications and participate to MCS scenarios with the PCS system even when these prerequisites are not met.

**ECM³**

The work presented in [34] is based on assumptions and techniques introduced by the *Piggyback CrowdSensing* [36] system. However, ECM³ has a broader scope of the study as it is attempting to increase user participation during crowdsensing experiments by reducing energy consumption and communication cost, while being privacy-aware. This is achieved by using anonymised call logs together with cell tower IDs to build call and mobility prediction models. The system will assign users to crowdsensing tasks only if it believes that they will be able to carry through with it completely during the task cycle. This study defined *task cycle* as the period of time between the instances when sensor uploads are needed. Furthermore, task assignments and data upload must happen during a 3G call or data connection, such as web browsing, in order to take advantage of the parallel transfer technique. That is because the parallel transfer technique was proven to be up to 68% more energy efficient compared to a simple cellular data-based scheme, when a new connection is established during the task assignment and data uploading.
McSense

McSense [5] is a platform designed to grant intelligence during the selection of users for crowdsourcing experiments in order to reduce the monetary incentives cost, increase tasks completion, and improve gathered data quality. The core endeavour behind its design is the execution of crowdsourcing tasks which achieve the optimal results, while occupying the minimum possible number of users, also known as workers. McSense excludes potential participants and devices through a series of filters including current location, possibility of staying in that location, device characteristics, and battery level. Even though researchers realise that battery life is a major component of any crowdsensing experiment, there is no design of a mechanism that targets that area. Instead they choose to ostracise horizontally devices based on that. On the other hand, location is the key criteria that is shared. More specifically, McSense platform constantly trains a geo-social model with user’s location as the main characteristic deliberated. After a two-month test period, researchers came to the conclusion that leveraging on the history and a future projection of users’ location could lead to improved experiment completion percentage. Additionally, they highlighted the bekown issue of low data quality caused by malicious participants.

AnonySense

Cornelius et al. introduced a pioneer system in the field of privacy of crowdsensing experiments back in 2008 [6]. Even though prior to AnonySense’s release, various researchers had noted the challenges of reliability, energy consumption, and user privacy protection, it was one of the earliest systems providing appropriate techniques targeting them. Authors claimed that cryptography and node anonymisation are not enough, as malicious users could associate data with users based on the time of upload. Hence, they implemented a framework that allows defining and reporting sensing tasks in an anonymous way using a novel, portable, and light language called AnonyTL, over a Mix network which uses proxy servers to hide the source the broadcasted messages. Even though AnonySense does not explicitly provide a power saving mechanism, the very nature of the system is energy aware as it relies on Wi-Fi connection while AnonyTL has a low interpreter footprint. Through a series of tests, the authors claim that the proposed system can perform a sensing cycle – retrieving task, sensing, reporting – consuming the same energy as playing a local music file for 46 seconds. Finally, they point that retrieving a sensing task from the back-end is to the highest degree the most energy demanding operation of the crowdsensing.

Quality-of-Information satisfaction ratio

In [38] Liu, Zhang et al. proposed the concept of Quality-of-Information (QoI) satisfaction ratio in an attempt to assess to what extend crowd-sensed data meet the requirements of a sensing task. Opposing to other systems, this one does not imply that a user’s future trajectory is known, but only his/her location while entering the crowdsensing area. Privacy is ensured by disassociating the
uploaded measurements from the personal information of the users. As with similar studies, authors associate the success of a sensing task with the location of participants and their enthusiasm to provide data until the QoI is satisfied. Furthermore, they note the correlation of this enthusiasm to the energy consumption of their device. Hence, using a kth-order Markov chain they built a model that projects participant’s movements given her historical data. They tested the introduced user-selection system against a dataset containing real movement information. Authors concluded that the mathematical relation between crowdsensing samples and energy levels, together with mobility projection results to a higher level of quality of information satisfaction.

**Context-aware Real-time Open Mobile Miner**

*Context-aware Real-time Open Mobile Miner* (CAROMM) [26] is a framework that attempts to minimise the energy consumption associated with mobile crowdsensing, while maintaining comparable data accuracy with the conventional crowdsensing systems. CAROMM consists of a smartphone application and a cloud-based data processing module. The former module is responsible for collecting and mining data generated from inertial, positioning, and environmental sensors embedded on Android devices. The latter component aggregates data from multiple users with information gathered from social media to construct comprehensive contexts of the users’ environment. CAROMM performs real-time analysis to the sensed data and uploads them to the back-end only when significant changes are detected. This ensures the energy efficiency of the system, as the network transmissions are minimised. However, as noted by the authors, one must find the golden ratio between data accuracy and energy efficiency. After conducting a series of experiments, the researchers concluded that their approach offers reduction in energy consumption and bandwidth usage by 3 and 17 times correspondingly compared to traditional crowdsensing techniques.

**SmartRoad**

Hu *et al.* demonstrated a crowdsourcing system that uses smartphones to determine the location of traffic lights, regulators, and stop signs [13]. *SmartRoad* follows the client-server architecture and consists of an Android application and a back-end server. The former gathers data generated by the embedded sensors of a smartphone, applies classification techniques to them, and finally uploads them to the server. The latter, aggregates and stores the incoming streams of data to extract information concerning the aforementioned elements of traffic. The authors realise that continuous location detection and sharing are power hungry operations that need to be confronted. Hence, they choose not to upload streams of raw GPS data, rather the outcome of local processing that was proven to be more lightweight than data transmissions. Furthermore, a power-aware component verifies that devices have sufficient battery load, or they are charging, before attempting to collect and broadcast any data. Following a real-life experimental period, SmartRoad managed to correctly
identify roads with 90% and 80% accuracy for supervised and unsupervised training information respectively.

2.1.1.2 Middleware applications

CUPUS
As part of the FP7 research project OpenIoT, CUPUS (CloUd-based Publish/Subscribe) middleware was presented, attempting to reduce the overall energy consumption of mobile crowdsensing systems [31]. The proposed solution relies on the well-established publish/subscribe model but, offers a different approach which is more decentralised than the norm. More specifically, the CUPUS Mobile broker is installed and runs on mobile devices and is used as a gateway for embedded and local bluetooth-connected sensors. This broker can filter out irrelevant, or redundant sensor measurements and prevent their transmission to the back-end. The back-end consists of a Cloud Broker architecture which is mainly responsible to handle all publications and subscriptions. The novelty introduced by the authors is the requirement-based exchange of data. A mobile broker collects and publishes generated measurements only when they are valuable and needed from ongoing MCS applications. Cloud and mobile brokers communicate using either persistent transmission control protocol (TCP) connections or, via the Google Cloud Messaging platform. Through a real-life scenario in the city of Zagreb, Antonić et al. demonstrated that the CUPUS ecosystem is scalable and energy-efficient.

2.1.1.3 End nodes applications

Crowd++
In [29], Xu, Li, and Zhang present a MCS application that is able to infer the number of people engaging in conversations without the need for external hardware. Using off-the-shelf smartphones, Crowd++ captures audio segments and runs an algorithm which estimates the active speakers with a very narrow error margin of ± 1.5 speakers [29], based on speech recognition. Even though this system excels in loud environments such as restaurants, or meeting rooms, its approach falls short on crowded places where people do not usually interact such as the subway or, designated quiet areas like movie theatres and libraries. Therefore, even though it achieves promising results, it is proposed to work in parallel with other crowd estimation systems, rather as a stand-alone system [40]. In terms of energy consumption, Crowd++ adapts the sensing window, when no speech is recognised, in an attempt to minimise the effect of the energy-consuming sound recordings which are needed. Even though they are still not implemented, the authors discuss about potential strategies that could alleviate the energy problem, such as using speech recognition hardware dongles or CPU cores, conversation prediction taking advantage of personal usage logs, and nearby
phones detection. It is important to underline the fact that there is no information concerning any data exchange between the smartphones and a back-end server.

**BX Tracker**

*BX Tracker* is a platform proposed to create heatmaps of cell networks characteristics striving for energy efficient crowdsensing [27]. The authors run a survey which indicated that users are more accepting to crowdsensing experiments when their active participation is limited, as well as, when the expected impact on battery life is minimum. However, in order to collect the required data, a perpetual, power hungry, location monitoring is needed. To tackle the battery exhausting nature of location tracking, this study utilised the accelerometer, which is a few orders less energy-demanding than the GPS. Additionally, when GPS fix was needed, the platform obeys a set of rules that minimises the time of usage. More specifically, the GPS would not be active for more than 30s and would gradually stop attempting to get a satellite fix, after a few failed attempts. All the recorded cell network data are compressed and transmitted to a server using an HTTPS connection at a 3-hour interval. The system claims to reduce the battery lifetime by 20%, which is a 5% better energy consumption compared to the Google Tracker. This was achieved by minimizing the GPS active time, by applying human activity detection, and by taking advantage of the devices’ inertial sensors.

**FindingNemo**

Liu and Li attempted to tackle the issue of lost children in open public areas in [35]. They proposed a system that leverages on the embedded bluetooth interfaces of modern mobile phones in order to locate missing children who are equipped with portable Bluetooth low energy (BLE) tags. More specifically, *FindingNemo* is a smartphone application that continuously scans for nearby bluetooth devices at a low-duty frequency even when the device is not used. When a kid goes outside the range of the device, the application warns the user who can choose to trigger a “lost” alert. When doing so, all neighbouring participating devices are alerted and start scanning for bluetooth devices hoping to match the one installed on the child’s clothing. At the same time, the engaged smartphones upload their findings and their location to a cloud aggregation server for further processing. The authors underline that system’s success is correlated with the number of active users, hence they propose an incentive mechanism that will reward participants with credit that can be used in future crowdsourcing experiments. With a clear view of how energy-demanding this system could be, researchers try to minimize the background scanning as much as possible. Additionally, there is an option to use only BLE communication assistance and opt-out from offering GPS localisation support during an emergency scenario.
Portolan
A crowdsourcing system that aims to monitor and measure the performance of the available networks is presented in [37]. Portolan consists of a server that can create and broadcast crowdsensing campaigns and of a smartphone application which runs on Android devices. Portolan application receives available sensing tasks by contacting the server via Hypertext Transfer Protocol (HTTP) connections at regular intervals. Additionally, during “urgent” scenarios the smartphones are notified using the Google Cloud Messaging service. The mobile app can measure bandwidth, route between two endpoints, round trip time and signal strength. The selection of the available participants is mainly based on their location for the time being. Researchers validated the system using real domains in Italy and proved that even a single device can offer impressive insights on the network it is connected to. The authors claim that network-related tasks are energy efficient and a device can execute them continuously for 15 minutes without consuming more than 1% of battery life. Concluding, even though the system offers fine grain localisation based on GPS and network info there is no provision for user’s privacy and security.

Ear-Phone
Ear-Phone is a system presented by Ran et al. in [32] that builds a time-aware noise pollution map in urban areas. Since its introduction, the mobile application scene has changed drastically. Even though modern mobile operating systems provide rich application programming interfaces (APIs), back in 2010 the authors had to build their own signal processing module to translate microphone input to sound level. Sound level measurements were tagged with location data, acquired from GPS measurements and converted to the Military Grid Reference System (MGRS) standard\(^3\). After calibrating the test devices, Ear-Phone achieved sound level precision of ±2.7dB. However, when sound signal processing and localisation services take place simultaneously they took a great toll on device resources, occupying 40% of system RAM and 98% of CPU load. This resource-demanding behaviour of Ear-Phone was rationalised by the researchers stating that the main target of the system was the reconstruction of a noise map and not energy efficiency. However, based on the conducted evaluation Ear-Phone was able to create an accurate noise map with 40% less samples than intended. Hence, reducing sampling on the mobile devices could potentially alleviate the energy issue, while maintaining adequate accuracy.

---

\(^3\) https://www.maptools.com/tutorials/mgrs/quick_guide
TailEnder
A protocol that can reduce energy consumption by up to 52% is introduced by Balasubramanian in [39] called TailEnder. Combining aggressive data prefetching and transmission-scheduling it drastically minimises the power needs of delay-tolerant applications. The former strategy estimates the number of documents that need to be downloaded prior to user’s inquiry based on statistics and logs. The latter attempts to broadcast requests in groups in a predefined timeframe in order to reduce the tail energy associated with the end of a transmission.

Energy Efficient Mobile Sensing System
Wang et al. proposed a framework that promises to increase battery life by 75% during mobile sensing experiments, without compromising data accuracy [8]. The introduced Energy Efficient Mobile Sensing System (EEMSS) extracts user’s state which is defined as a combination of context features such as location, motion and ambient sound level. By combining sensor measurements, EEMSS can infer the current state of a user, and depending on the state it keeps monitoring a subset of these sensors. This system is using mainly the accelerometer and the microphone to extract meaningful data in an energy-aware way, instead of utilising more power-demanding alternatives like GPS and bluetooth scanning. Additionally, EEMSS heuristically adapts sensing cycles to minimise energy consumption, without compromising its accuracy, as it managed to recognise user’s state precisely 92.5% of the times.

CenceMe
Back in 2008, a group of mobile sensing pioneers presented CenceMe [28]. At the time, it was one of the first third-party applications that could be installed to everyday smartphones and offer context-aware functionalities by leveraging on embedded sensors. CenceMe consists of a mobile application and a back-end server that communicate via HTTP web services. Using a combination of inertial and environmental sensors, the mobile application can infer users’ activity (walking, sitting, dancing etc) and context (engaging in conversation, attending a party). A significant number of classifiers were used, including audio/voice, activity, mobility, and social. Some of them are running locally on the phone, while the most energy demanding in the back-end. The authors attempted to minimise the energy consumption during the background sensing in two ways. First, the local analysis on the sensor data decreases the total number of transmissions and the volume of the data exchanged. Secondly, the sensing components are not engaged continuously, but rather follow a power aware duty-cycle that avoids rapid drain of the battery but sacrifices system’s accuracy.
2. Literature Review

2.1.1.4 Findings
This section offers an insight on the current state of MCS and presents a taxonomy of existing approaches grouped into Full stack, Middleware, and End-node applications based on their scope. After examining the introduced solutions, it is evident that most of the researchers have attempted to tackle the challenges that modern MCS systems are facing. Techniques such as group sensing, offloading, data encryption, sensor fusion, and context-aware sampling are a few examples of the proposed solutions targeting in the areas of resource constraint, privacy, and data quality. A number of these techniques are seamless and could be easily accepted by the end users, however, some are the opposite. More specifically, few researches have some prerequisites, such as root access on mobile devices, that may limit their acceptance rate or their practicality in real-world environments. This is because end-users may lack the necessary expertise, do not want to void the warranty of their device, or deteriorate their user experience. A more extensive analysis of the author’s findings is presented in Section 2.4.

2.2 Internet of Things (IoT) protocols
As discussed, Internet of Things relies on the ability of everyday objects to be inter-connected. These devices utilise a transport protocol to send and receive data, regardless of whether it happens over mobile data, Bluetooth, Radio-Frequency Identification (RFID), Wi-Fi, or any other wired, or wireless mean. In order to respond to the ever-increasing demand of exchanged information, researchers and companies introduced, or extended a series of IoT-oriented communication protocols. This section paints the picture of the main IoT protocols used by the crowdsourcing community.

2.2.1 Message Queuing Telemetry Transport
The Message Queuing Telemetry Transport (MQTT) is an open publish-subscribe messaging protocol developed by IBM in 1999 and adapted as a standard in 2014 [41] [31]. It is highly used in the area of IoT thanks to its low-bandwidth nature and tolerance in high-latency environments and runs over the TCP/IP stack [42]. Devices that utilise MQTT are subscribed to topics and receive the messages that are transmitted to them. All the clients of this pub/sub model must be connected to a broker which is responsible for all message handling. To do so, they need to exchange at least one message within a predefined time interval known as Keep Alive Interval.
MQTT offers three different levels of Quality of Service (QoS). In QoS 0, each message will be delivered at most once, without any confirmation, or message storage (also described as “fire and forget”). It is the fastest mode of transfer but comes with a risk of the message to be lost. QoS 1, makes sure that the message will always be delivered at least once, using confirmation messages called Publish Acknowledgements (PUBACK). However, the message might be sent or delivered multiple times if the acknowledge (ACK) takes too long to be received. QoS 2, provides an exactly-once type of message delivery using a handshaking mechanism which is more advanced than the one used by QoS 1. The additional messages sent between sender and receiver prevent the multiple delivery but make this mode of transfer slower. MQTT brokers can retain published messages and push them to future subscribed clients.

2.2.2 Advanced Message Queuing Protocol (AMQP)

The Advanced Message Queuing Protocol (AMQP) is a standardised protocol that was developed by a group of companies targeting the need for interoperable messaging in the financial sector [43]. Key members of the AMQP consortium include Barclays, JP Morgan Chase Bank, Cisco, Microsoft, Red Hat, Goldman Sachs, VMware, and Bank of America. Early versions started to be released in 2006 but ultimately, they had great differences from the 1.0 specification which was introduced in 2011. This version was later approved as a standard in 2012 by OASIS and in 2014 by ISO/IEC [44]. Despite the fact that it was intended for enterprise applications and server-to-server communications, it is highly used in the IoT area due to its features [45].

AMQP usually runs on top of TCP, but it can be used with any reliable underlying transport-layer protocol such the Stream Control Transmission Protocol (SCTP). Same as MQTT, AMQP also utilises a broker and the pub/sub scheme, but it also supports the request/response architecture.

Similar to MQTT, AMQP offers three Quality of Service levels for message delivery: At-Most-Once is the scenario where the message is broadcasted but the sender is not waiting for a delivery confirmation and no retries are applied. It is also known as the fire-and-forget. At-Least-Once is the level of delivery-tag used to assures the sender that the receiving application will process at minimum one copy of the send message, with a chance of duplicate deliveries. Finally, the exactly-once, as one might guess, guarantees that a message will be certainly delivered just once, without any redundant duplicates [46].

2.2.3 eXtensible Messaging and Presence Protocol (XMPP)

The eXtensible Messaging and Presence Protocol (XMPP) enables the “near-real-time” exchange of XML (Extensible Mark-up Language) messages [47] and was developed in 1999 under the name
“Jabber”. It has been used for instant messaging, voice and video calls, and gaming applications among others. Clients cannot communicate to each other directly, as all messages have to be sent to and handled by an XMPP server. Similar to MQTT and AMQP brokers, the XMPP servers are decentralised, hence can be deployed and run on any supported machine.

Every XMPP client has a unique id which consists of a username and a domain address resembling the structure of an email address. This protocol runs over TCP/IP and supports asynchronous communication only via XML streams and stanzas. These stanzas encapsulate simple message strings, together with presence information of the client. XMPP does not intrinsically support any kind of quality of service, however, its extensible nature allows researchers and developers to build on-top mechanisms that provide this missing functionality. Most QoS solutions include additional information inside the exchanged stanzas.

2.2.4 Representational State Transfer (REST)

The Representational State Transfer (REST) is an architecture style that was defined by Roy Fielding in 2000 in an attempt to set guidelines to the development of the modern – at the time – Web. The big advantage of RESTful web services is that they can provide any information as a resource to the Internet. All the interactions are stateless, which means that a request must contain all the information needed for the server to understand it, regardless of any previous communications and exchanged messages [48]. RESTful architectures can be applied on various Application Layer protocols [46], but it is most frequently used with the Hypertext Transfer Protocol (HTTP).

2.2.4.1 Hypertext Transfer Protocol (HTTP)

The HTTP is an application protocol that was introduced in 1991 and is widely used in the World Wide Web. It runs over TCP and uses the client and server architecture. RESTful applications that run over HTTP can use the GET, POST, PUT, and DELETE methods [49]. Since REST is not a transport protocol it does not provide any kind of Quality of Service but inherits the characteristics of the underlying transport protocol.

2.2.4.2 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) is a web transfer protocol that was designed by the Internet Engineering Task Force (IETF) and is intended for constrained devices and networks [50]. It runs over the User Datagram Protocol (UDP) and follows the request/response message architecture. Moreover, same as HTTP, CoAP is a document transfer protocol and can be RESTful. Its clients can send GET, PUT, POST, and DELETE resource requests [51] [52]. CoAP offers a
basic level of ensuring quality of service. More specifically, all sent messages can be defined as “confirmable” or “non-confirmable”. The former type must be acknowledged by the recipient with an ACK message, while the latter type is similar with MQTT’s level 0 QoS “fire and forget” [53].

Throughout Chapters 3 – 7, when “REST” protocol is mentioned, it is implied that we refer to HTTP client-server communications that follow the RESTful architecture.

2.2.5 Comparison of IoT protocols

This section presents a comparison of the protocols discussed in the previous sections of this chapter. More specifically, Table 2 summarises some of the features that characterise them and are important to experimenters and end-users such as architecture, quality of service, payload size, and level of security.

<table>
<thead>
<tr>
<th>Comparison Criteria</th>
<th>MQTT</th>
<th>XMPP</th>
<th>AMQP</th>
<th>REST/HTTP</th>
<th>CoAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction</td>
<td>Pub/Sub</td>
<td>Peer-to-peer Pub/Sub</td>
<td>Peer-to-peer Pub/Sub</td>
<td>Request/Reply</td>
<td>Request/Reply</td>
</tr>
<tr>
<td>Architecture</td>
<td>Broker</td>
<td>Broker</td>
<td>Broker</td>
<td>Server-client</td>
<td>Server-client</td>
</tr>
<tr>
<td>Mobile devices</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td>&lt; 256MB</td>
<td>Undefined</td>
<td>Undefined</td>
<td>Undefined</td>
<td>&lt; IP datagram</td>
</tr>
<tr>
<td>QoS Levels</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Security</td>
<td>Authentication, SSL</td>
<td>TLS, SASL</td>
<td>TLS, SASL</td>
<td>SSL, TLS</td>
<td>DTLS</td>
</tr>
<tr>
<td>Standard</td>
<td>OASIS proposed</td>
<td>IETF</td>
<td>OASIS</td>
<td>N/A</td>
<td>IETF proposed</td>
</tr>
<tr>
<td>Transport</td>
<td>TCP</td>
<td>TCP</td>
<td>TCP</td>
<td>TCP</td>
<td>UDP</td>
</tr>
<tr>
<td>Licensing Model</td>
<td>Open source</td>
<td>Open source</td>
<td>Open source</td>
<td>Free</td>
<td>Open source</td>
</tr>
</tbody>
</table>
2.1.1.1 Payload size

- **MQTT**: according to OASIS standardisation document, MQTT supports up to 256Mb of payload size. However, depending on the platform used, there might be some limitations like the IBM’s *Watson IoT Platform*[^4] which supports up to 0.13Mb.

- **XMPP**: there is no official definition inside the XMPP Core documentation[^47]. It is up to the XMPP server implementation to force a maximum stanza size.

- **AMQP**: Same as XMPP, AMQP has no set value for maximum payload size[^54]. It is agreed between the server and the client at the start of a connection. In some cases when there are no sufficient resources to allocate, a server can reject a transmission of a very large frame. RabbitMQ, which is one of the most popular AMPQ brokers and servers, defines the maximum payload size to 2Gb[^55] but suggests users to exchange smaller chunks of data.

- **REST**:
  - **HTTP**: In IETF’s standard[^56] there are no explicit limits imposed to HTTP transmissions. Any request though is subject to server and client storage limitations and transmission requirements.
  - **CoAP**: protocol’s specification states that a message should be compact enough to fit within a single Internet Protocol (IP) packet and avoid any fragmentation. For IPv6 that translates to 1024 bytes of payload, while for IPv4 it is a bit more complicated. Due to IPv4’s support on “unusual networks”, IETF recommends datagrams of up to 576, while in some cases it is noted that the maximum transmitted payload could be as low as 40 bytes[^57].

2.1.1.2 Security

- **MQTT**: The first layer of security in MQTT can be provided by using TLS/SSL which is over the transport layer (TCP). By default, data exchanged are not encrypted in MQTT. On the application layer, the MQTT protocol can authenticate clients using username/password credentials. Furthermore, depending on the broker’s implementation there is additional control of the allowed actions of each client/device.

- **XMPP**: XMPP’s specification is packed with two security features. Like MQTT, the first one is the Transport Layer Security (TLS) mechanism which relies on the transport layer and provides secure communications through encryption. The second feature is the use SASL (Simple Authentication and Security Layer) technique which provides authentication support.

[^4]: https://console.bluemix.net/docs/services/IoT/reference/mqtt/index.html#ref-mqtt
• **AMQP**: Same as XMPP, AMQP provides two security layers. TLS sessions are used to ensure data encryptions, while SASL offers authentication capabilities.

• **REST**:  
  o **HTTP**: The most commonly used HTTP security mechanism is the Secure Sockets Layer (SSL). More recently, TLS implementations are applied over RESTful HTTP connections. Both solutions offer symmetric encryption of data transmitted and asymmetric cryptography to authenticate the key exchange [58].  
  o **CoAP**: IETF defines two levels of security for CoAP. The first one is “NoSec” when no security is provided in the protocol-level. Authors encourage the use of lower-layer techniques such as IPsec to be used, where appropriate. The second way of securing CoAP is through the Datagram Transport Layer Security (DTLS) [57], which is an enhanced TLS designed to cope with the unreliable nature of the UDP connections.

This subsection presents and compares the most popular IoT communication protocols in an attempt to draw a map of their strengths, weaknesses and their potential application in crowdsensing environments. With the help of Table 2, the reader can clearly identify their general features and not only spot their similarities, but also understand that their philosophies are quite diverse. It is evident that no protocol can be labelled as the best for crowdsensing scenarios, as their characteristic may satisfy different needs. However, some conclusions can be drawn based on their pros and cons as follows.

If the importance of each message and the repetitiveness of them is crucial to an experiment then it would be wise to choose either the MQTT or the AMQP for a solution out-of-the-box, as they both offer three levels of Quality of Service. Following them should be CoAP and then XMPP and HTTP. Nonetheless, if one is willing to extend these protocols, then the extensible nature of XMPP could render it as a potential choice.

In case that interorganisational collaboration is a key criterion then a broker-based architecture might be the ideal solution like the ones used by MQTT, XMPP, and AMQP. In that way, one could easily subscribe to a 3rd party broker and receive all the transferred data without the need to create additional APIs to access a remote database, as it would be required for HTTP. On the other hand, if a more centralised approach is needed without the disadvantages that a broker introduces (such affected scalability), then HTTP or CoAP should be adopted.

Moreover, all the presented protocols except CoAP are operating over TCP, which was designed many years before the introduction of power constrained IoT devices. The handshaking that is required to set-up a communication channel before the start of message exchange results in increased wake-up times. Thus, it affects the long-term energy demands.
Each of the protocols discussed in this section offers certain advantages but also present some drawbacks. Hence, it is only fair to say that selecting one for a crowdsensing experiments has to be based on system and/or user requirements.

2.3 Definition of key performance indicators

In order to evaluate the performance of the selected IoT protocols during crowdsensing experiments, key performance indicators (KPI) have to be defined. Some of them derive from objective network performance metrics, others from subjective targets set from crowdsensing experiments, while the rest are determined by participants’ preferences. Below are some criteria that can be used as KPIs:

**Energy consumption**: energy consumption is an aspect of crowdsensing that affects all stakeholders. On one hand, if the end nodes are personal portable smart devices, then an experiment should not compromise the users’ experience [7]. If the end nodes are standalone constrained sensors, the power needs of an experiment may seriously influence battery life span. On the other hand, energy consumption is directly linked with the cost of running a crowdsensing platform, which is crucial for the service providers [59].

**Message success**: reliable transmission and storage of the crowd sensed data is, in principal, a concern predominantly of the service providers. The need of acquiring as much data as possible is apparent, as they will be essential to extract the targeted results. However, end users are becoming gradually more concerned about this feature, as incentives and message delivery success are interdependent [60] [16].

**Security**: security has been a major concern in the discipline of computer science. Data privacy is of great importance to users in order to contribute to mobile crowdsourcing experiments [26][15]. Additionally, criminals can exploit data gathered from stationary IoT devices in order to plan malicious acts, such as robberies, or cyber-crimes. Furthermore, service providers must ensure data privacy, as law acts are continuously introduced to many countries’ legislations.

**Data usage**: even though mobile data plans are more affordable than ever, data usage is still a concern to users [7]. Hence, many applications that can perform data exchange in the background ask user’s permission to do so. Furthermore, they enable users to change their settings and provide a personalized set of rules. Additionally, data traffic is a concern to service providers, especially in areas characterized by high population density.
Message length: depending on the nature of a crowdsourcing/crowdsensing experiment message length may be a key factor. In a wireless network of constrained sensors, the exchanged messages shall be as small as possible. On the other hand, in power-supplied devices who have different sensing priorities, message length is not as important.

Ease of setup: stakeholders' proficiency in crowdsourcing systems might affect their choice when selecting a communication protocol. That is due to the fact that some require additional components/middleware to ensure data exchange, while others are less complicated by design.

2.4 Gap analysis

The previous sections of this chapter provide a plethora of crowdsensing systems that were developed and presented by the research community. Except a few, the majority of them acknowledged the challenges of mobile sensing and attempted to address them. The introduced solutions targeted different parts of a MCS system. Figure 2, illustrates the stages of a crowdsensing experiment after it has been initialised. Continuing, a per-step discussion takes place to better comprehend and map the state-of-the-art solutions.

![Figure 2: The steps of a crowdsensing experiment.](image)

2.4.1 Sampling

Sampling is the first step in the crowdsensing cycle and raises a series of obstacles in terms of energy consumption and data quality. Current solutions can be grouped as follows:

- **Energy**: a common practise used to alleviate the energy demands of a mobile sensing system includes limiting the use of power-demanding sensors such as the GPS. Instead, more energy-efficient alternatives are exploited. Another technique relies on group sensing, where participating devices in vicinity collect sample data in turns, hence minimising the energy cost. Moreover, prediction models make sure that devices perform demanding tasks when in-use by the owners, thus reducing their effect on the battery. Finally, machine learning algorithms reduce sampling windows when appropriate and consequently energy demands.
2. Literature Review

- **Data quality**: prediction models prevent the system from sampling irrelevant data. Additionally, machine learning algorithms can infer user’s context and decide whether sampling would provide meaningful quality data or not. Finally, sensor fusion techniques during the sampling stage benefit the overall quality.

2.4.2 Transmission

Transmitting generated data has proven to be an energy demanding aspect of the crowdsensing cycle, sometimes more demanding than the actual sampling. Below are the techniques proposed.

- **Energy**: machine learning algorithms can predict critical events such as using the phone or, making a call and piggyback the crowd-sensed data to the already enabled channel. Additionally, a number of MCS systems choose to transmit their data using less energy demanding interfaces such as the BLE, or Wi-Fi instead of cellular data. Similarly, other researchers suggest not to transmit data after they are captured but store them and wait for an encounter with another participant or a power efficient gateway to offload them. Finally, by using light proprietary programming languages or by compressing the generated data, the energy demands are minimised.

- **Privacy & Security**: to prevent potential malicious users from sniffing any transmitted data, researchers obfuscate and encrypt them. Additionally, the communication channel is encrypted.

2.4.3 Storage

Modern crowdsensing experiments leverage the rapid growth and the accessibility of cloud providers. In that way, energy cost is no longer a factor that affects researchers and experimenters. Additionally, from that stage and onwards, the device of a user is not engaged anymore, thus is not suffering from any energy burden.

- **Privacy & Security**: it is of paramount importance to keep the data acquired from crowdsensing experiments safe from any malicious users. Role-based access to the databases and data encryption are the most popular methods used by the community.

From the literature review conducted on current systems it is evident that there are numerous techniques that strive to make mobile crowdsensing as efficient as possible. It is fair to say that most of them have managed to mitigate the effects of ubiquitous sensing on portable devices. Nonetheless, despite all the work that has been done, some of the solutions are unsuitable for the
everyday users (i.e. need to root the devices\(^5\)) and some of them have some serious sensing constraints. Additionally, by examining Table 1 one can notice that researchers are inclined to use a single communication protocol to exchange data between participating devices and the back-end of a system. Additionally, despite that there are multiple IoT-oriented communication protocols, most implementations are using the HTTP-REST. Even though, protocol selection is frequently overlooked as something of a minor importance, it is an area of interest that could potentially make a great difference, since communication is key factor of battery life and consequently of user experience [26].

2.5 Next steps

The previous section identifies the research gap during the transmission stage of mobile crowdsensing systems and the static approach that system designers prefer. In such a dynamic environment as the one that mobile devices are operating, one has to be flexible to face the everchanging parameters adequately.

The first step to examine the impact of IoT communication protocols on modern crowdsensing scenarios is to make a collection of them, amongst the available ones. MQTT and CoAP are two communication protocols that were specifically designed for constrained environments and should be the ideal candidates for any IoT and, particularly, MCS application. Nevertheless, the literature review concluded that this is not reflected by reality, as the lion’s share of crowdsensing research systems utilise HTTP and, in some cases, XMPP variations. Hence, this study includes the HTTP and XMPP protocols. Furthermore, between the analogous MQTT and CoAP, we are choosing the former, as it presents the biggest penetration to both research studies and commercial systems. Finally, it is documented in the literature that AMQP is a power-demanding protocol that should be avoided in constrained environments, such as the ones met in IoT systems, as it constantly keeps a connection open [61]. However, high level of security, scalability, and reliability are features of AMQP that could potentially tackle the current challenges of mobile crowdsensing. Furthermore, we desire to monitor its energy demands in constrained systems first-hand. Therefore, the last protocol assessed is AMQP.

\(^5\) http://unbrick.itcse.com/rooting-advantages-disadvantages/
2.6 Conclusion

This chapter analyses the current landscape in mobile crowdsensing by scrutinising existing platforms and solutions. It presents the numerous ways researches attempt to alleviate the problems associated with MCS such as energy demands and overall robustness. Additionally, a survey of the most popular IoT communication protocols sheds light on their advantages and drawbacks when used during MCS scenarios against a variety of key performance indicators (KPIs). By examining the past work, the available IoT and the KPIs the author identifies a research gap during the data transmission of a crowdsensing experiment related to the static use of communication protocols. In order to investigate further this finding, a system that can support and monitor crowdsensing experiments is designed and presented in Chapter 3.
Chapter 3

Overall System Definition

As discussed earlier in this study, the area of crowdsourcing still has open efficiency issues in terms of both energy and privacy oriented. This chapter constitutes the link between the literature review and the remaining of this research as follows; In order to examine and identify existing limitations to conventional crowdsourcing systems, a testing system is needed. Hence, the following sections present an abstracted overview of the architecture of the system to be implemented and evaluated. Furthermore, an analysis of core functionalities and roles is showcased. Additionally, the experimental set-up is defined, alongside with detailed presentation of the experiment lifecycle.

3.1 System components

Internet of Things experiments suggest that nodes gather pieces of information in order to either process them locally and share the outcome or upload them to a remote server for storage and future processing. Additionally, IoT by definition is an umbrella term that includes a wide spectrum of connected machines, gadgets, appliances and more, which could potentially participate in crowdsourcing scenarios. Since this study focuses on constrained crowdsourcing environments, only wireless devices are taken under consideration. Devices that are part of a wired network, customarily have access to a power source, hence could not be tagged as constrained in terms of power needs.

End nodes that generate and collect data samples use the Internet as the primary medium to share and upload them to a main framework for further analysis, or simple storage. If one would like to depict these actions in an abstracted and coarse-grained way, Figure 3 would come naturally. Notice that this illustration presents no information concerning the data exchanged between the end nodes, nor the rest of the participating components. The following sub-sections cover these aspects.
3. Overall System Definition

3.1.1 Data aggregation and consumption server

The illustrated *storage/processing server* in Figure 3 could either be installed on a physical computer or a virtual machine (VM). For this study a data aggregation and consumption server (DAC) that runs on a virtual machine located at the University of Surrey was designed and implemented by the author. The purpose of this server is to create crowdsensing experiments and notify the involved devices, by passing the corresponding sensing parameters to them. To do so, our DAC server supports a variety of IoT communications protocols, which also enable the monitoring of ongoing experiments. The DAC server provides an intuitive user interface (UI) that enables users to easily setup, define, and configure on the fly all the experiment-oriented parameters of a crowdsensing scenarios. The said parameters are discussed in depth in Sections 3.2 – 3.4.

Finally, this server is responsible for storing all the uploaded crowdsensing data. To achieve that, a MongoDB database was deployed on the same virtual device, which is accessible via a set of RESTful web services created with the help of DreamFactory’s interface builder. The author decided to use off-the-shelf products whenever possible rather than re-inventing the wheel, to minimise the risk of bottleneck due to human errors.

More details on the storing of data are presented in Section 3.5.

---

6 https://www.dreamfactory.com/
3. Overall System Definition

3.1.2 Connected devices

As noted by Atzori et al. [62], the “Things” in the IoT could be translated into something low-cost and simple, such as RFID tags, all the way to autonomous objects equipped with communication, sensing, and collaboration capabilities. This study is focused closer to the latter end of the above-mentioned spectrum; on devices with embedded sensors, actuators, and wireless communication hardware, which are easily accessible, have already achieved a high user acceptance, and are power constrained. A few examples include:

- Sensor suites/devices
- Smartphones
- Tablet computers
- Fitness trackers
- Wearables

Given that this study aims to compare and evaluate the way IoT communication protocols perform, it shall start the assessment by using generic approaches and gradually move to more fine-grained concepts. However, our measurement techniques must be applicable to all types of systems and complexities in terms of end-nodes participating. Hence, our first step includes abstract versions of clients that utilise a specific communication protocol and then move to more concrete ones.

3.1.3 Components coordination

For our system to be able to conduct crowdsourcing experiments, it is necessary that our DAC server and the sensing devices are bridged. The majority of the IoT communication protocols need an intermediate component that handles and distributes the exchanged messages. The following brokers/servers were used in this work:

- **MQTT**: all end nodes interact via the Mosquito\(^7\) message broker that is a free implementation of the MQTT and is installed on the Virtual Machine server. The version used was v1.4.11 and the configuration file was left to its default state\(^8\).
- **AMQP**: in order to queue and distribute AMQP messages, the RabbitMQ server was adopted, which is an open source implementation of an AMQP broker written in Erlang. RabbitMQ can also be run as a service on Windows environments, such as ours. Again, the default configuration was preferred. Version installed was v3.6.6.

---

\(^7\) [https://mosquitto.org](https://mosquitto.org)

\(^8\) [https://mosquitto.org/man/mosquitto-conf-5.html](https://mosquitto.org/man/mosquitto-conf-5.html)
• **XMPP**: our system’s endpoints exchange XMPP messages using Openfire’s open source instant messaging Java broker. Openfire v.4.2.3 was deployed on the same Virtual Machine server.

Figure 4: Architecture of IoT experiments using AMQP, MQTT, and XMPP

Figure 4 and Figure 5 depict a more detailed view of our system’s architecture presented earlier. IoT devices with the help of brokers can receive messages and upload generated data to the storage and processing server. Then, the said server is responsible to store them in the deployed MongoDB database. Notice that HTTP/REST experiments (Figure 5) do not require a broker in order to receive data from the back-end, as they are accessible using their IP address. Similarly, the IoT devices that implement the HTTP/REST communication protocol can access the database and store the measured data directly using the provided set of APIs.

Figure 5: Architecture of IoT experiments using REST

More details on how the crowdsourcing experiments take place are presented in Section 3.5.
3. Overall System Definition

3.1.4 System resources

All sensors are treated as crowdsourcing resources. The implementation of an index filled with these resources is mandatory, if there is desire to perform any experiments using them. Every sensor is treated as a separate resource which can be reserved from the system, independently of the nature of the host device. For example, the system treats a stand-alone temperature sensor the same way as an embedded one inside a portable smart device.

This index is implemented using the MongoDB database. In order to make a new resource record, a RESTful API call has to be made.

Table 3: Registering a new crowdsourcing resource

<table>
<thead>
<tr>
<th>Call</th>
<th>http://&lt;server_id&gt;/insertResource.php?name=magnetometer&amp;node_id=527&amp;function_set=ipso.sen&amp;resource_type=mot&amp;unit=mT&amp;path=&lt;Android_ID&gt;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>JSON Response:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Message:</td>
</tr>
<tr>
<td>{   &quot;Text&quot;: &quot;Fail&quot; }</td>
</tr>
<tr>
<td>Success Message:</td>
</tr>
<tr>
<td>{   &quot;Text&quot;: &quot;Success&quot;,   &quot;ID&quot;: &quot;700&quot; }</td>
</tr>
</tbody>
</table>

The fields below are provided to the back-end of the system, which takes care the new database insert:

- **Name**: Define a simple name for the provider (e.g. University of Surrey).
- **Function set**: Define the resource Function Set as defined by the Internet protocol for the networking of smart objects (IPSO) Application Framework [63]. Examples are “ipso/gpio” and “ipso.sen”.
- **Resource type**: Define the resource type as defined by the IPSO Application Framework. Examples are “ain” and “on”.
- **Unit**: Define the unit of the measurement the resource provides.
- **Path**: Define the path of resource that can be used to access it.
- **Ipso index**: Defined the IPSO index of the resource in case the node requires multiple resources of the same function set and resource type. Default is 0.
Below, Table 4 presents an example of how a disabled light sensor that is not-reserved for experiments can be seen in the database:

Table 4: Representation of a sensor in the resource index

```json
{
    "resource_id":"699",
    "name": "light",
    "urn": "",
    "root path": "/seen",
    "function_set_name":"ipso.sen",
    "resource_type": "mot",
    "resource_type_name": "Motion (sen)",
    "lon": "0",
    "lat": "0",
    "path": "-co6Qw5OvP-jjlbfWGK4U_eI8yp4NK",
    "unit": "lx",
    "last_updated": "2018-07-24 18:03:01",
    "currently_reserved": "0",
    "disabled": "1"
}
```

### 3.2 Experiment set-up

As discussed earlier, there are cases when crowdsensing experiments require data captured by multiple sources from every participating device. This variety of possible sensing loads is reflected in our study by assuming the following three different states of sensing:

- **Light-load sensing**: a device is uploading data captured by 1 source at a time.
- **Medium-load sensing**: a device is uploading data captured by 3 sources at a time.
- **Heavy-load sensing**: a device is uploading data captured by 5 sources at a time.

The reason behind the number of sources mapped to the different types of loading is the following: Modern connected devices can pack from a single sensor, to a handful of them. It is evident that the light-load sensing should be utilising just one source. On the other side, according to our literature review, no evidence was found to support that there are crowdsourcing experiments that need more than five sources, even in the most complex of the cases, such as sensor fusion [64][28][11]. Additionally, the fact that there are smart devices that embed more than five sensors is recognised, however not all of them are put to use simultaneously under crowdsourcing experiments due to the insurmountable energy tax that will come with it [18]. Hence, the heavy-load scenario should consist of five sources been utilised. Finally, during medium-load experiments, the devices upload data captured by three sources, as that was the closest rounded-up number to the half of the maximum number of sources set.
3.3 Transmitted data

In real life scenarios, the messages transmitted by the devices would encapsulate data gathered directly from some hardware, or a user. However, since the actual sampling is out of the scope of this study, a simpler sensing scenario was considered, where all smart devices would always broadcast a simple String “Hello World”, instead of accessing any hardware. This would make our evaluation fairer, as all test configurations transmit the same volume of data. Additionally, the measured energy consumption would not be affected by the different types of hardware sensors that might have been accessed – a sensor manufactured by vendor A embedded in device X might be more energy efficient that the equivalent sensor by vendor B embedded in device Y.

3.4 Experiment duration

Crowdsourcing experiments duration could vary from just a few seconds, to several days [60]. Furthermore, sampling and transmitting frequency follow similar trends; a sensor could provide data in a sporadic manner, while under different configuration a demanding continuous burst of data may have to be uploaded. It is evident that energy consumption is of a greater factor in the latter type of experiments – the long lasting and higher frequency ones. Therefore, our study assesses test configuration three times – in 5, 10, and 20 minute-long runs. Consecutively doubling the runtime allows us to detect any significant anomalies that could be linked to the duration of the experiment (a protocol’s performance might deteriorate, or even plummet after prolonged use, in terms of time). Moreover, data are sent in 5 seconds intervals in an attempt to simulate even the most demanding crowdsourcing scenarios.

3.5 Experiment sequence

This section introduces the steps that take place during our crowdsourcing experiments from the moment of creating a scenario, to the actual data exchange. Initially, the user creates a crowdsensing experiment via the User Interface of the DAC server. Parameters such as sensing load, protocol used, experiment duration, and sampling frequency can be set as shown in Figure 6.

![Figure 6: Setting experiment parameters using DAC server's UI](image-url)
When the user selects the “Create” button, the DAC server will send an appropriate message to all the IoT devices that will participate in the experiment, which contains all the defined parameters. The delivery of this trigger is managed either from the corresponding broker/messaging server, or is sent directly when using REST, as depicted in Figure 7 and Figure 8 respectively.

When a device receives the sensing parameters, it sets a repeating timer that will handle the data sampling and uploading. When the timer expires, the device “wakes up” and collects the data needed. As discussed in Section 3.3, no actual sensors will be accessed, however in real-life scenarios this process will remain untouched, expect the part where the actual sampling will take place. Data are sent to the DAC server using the intermediate broker/server. Finally, the DAC server relays the received measurements to the MongoDB using the corresponding APIs.

The aforementioned sequence gets less complicated during crowdsourcing experiments that use REST. More specifically, since there is no need for an intermediate message handler, the DAC server notifies the participating devices directly. In a similar manner, during the sensing loop, the IoT devices access and store the captured data directly using the exposed RESTful APIs. The rest of the steps remained unchanged though.
3.6 Conclusion

Chapter 3 presents the author’s work on defining, designing, and implementing an end-to-end crowdsensing platform. This includes i) a back-end which is deployed on a virtual-machine that can create tailored scenarios, collect uploaded data, and store them in NoSQL database with the help of RESTful webservices, ii) a middleware that runs instances of MQTT, XMPP, and AMQP brokers, and iii) a way to index and manage all the available crowdsensing resources. Commercial and open-source robust tools like DreamFactory and MongoDB are used, to ensure the reliability of the system, while the middleware is empowered by the OpenFire, Mosquitto, and RabbitMQ message brokers which all are well adopted and tested by the community.

The system architecture and the crowdsensing experimental set-up presented in this chapter constitute the backbone of this study. Chapters 4 and 5 build on top of them as they map all the system definitions and requirements to simulation tools and real-world devices correspondingly to provide all around testing results.

The author was responsible for setting up and deploying all the components of the defined system across different servers that run on virtual machines. Additionally, to ensure the fairness and repeatability of the experiments, all systems were left in their default configurations, while the implemented DAC server is not in a position to interfere with the results.
Chapter 4

Simulation-based Evaluation

In this chapter we evaluate our system and subsequently the selected Internet of Things communication protocols via a simulator, which provides valuable insights of the protocols’ performance, without influencing factors that are introduced when actual devices are engaged. The evaluated system follows the architecture that is defined in Chapter 3. A brief survey of existing solutions allows us to compare and reason our selection, given the criteria set by this study. Continuing, we analyse the changes we had to make to our selected tool, in order to serve its purpose. Finally, we discuss the simulation results starting from 1, all the way to 100 simulated devices.

4.1 Internet of Things (IoT) simulators

Given the penetration of IoT devices in our lives, it comes with no surprise that there is a plethora of available tools which can simulate a whole network of them. Companies such as MathWorks, Tetcos, and Automatski are some of the many that have implemented and released stand-alone, or add-on solutions. Apart from the commercialised ones, many researchers have developed their own tools focusing on aspects that are not covered by the existing ones [65][66][67][68]. It is evident that these solutions help researchers explore the way modern IoT topologies behave, in a convenient, cost-effective, and time-saving manner. The remaining of this section provide us with a brief insight on a number of existing network and IoT simulators.

OMNet++
OMNet++ is a simulation library and framework built by Andras Varga, which runs on top of the Eclipse integrated development environment (IDE). As stated by its creator, it is not a simulation tool per se, but a tool that helps researchers build and run their simulations [69]. A researcher can either upload or create from scratch a Network Description (NED) file. Predefined run time, real-time network status and node log information are some of the functionalities available. OMNet++ customisation capabilities and ease of use make it one of the most popular tools utilised for simulations.
IoTIFY
IoTIFY is a cloud-based IoT simulator that supports the MQTT, HTTP, CoAP, and Lightweight Machine-to-Machine (LWM2M) communication protocols [70]. The testing platform can be accessed via any web browser and an academic trial licence is available. One can choose from the default network templates or create a new one by defining the necessary parameters including connection protocol, connection timeout, and message content. Additionally, IoTIFY virtual devices can be tailored to the researchers’ needs as they can run JavaScript functions. This, together with their bidirectional communication capabilities render the IoTIFY platform a powerful tool that can simulate complex IoT scenarios.

NetSim
NetSim is an event simulator developed by Tetcos which offers customisation throughout the levels defined by the Open Systems Interconnection (OSI) model [71]. It can simulate sensor motes, gateways, switches, access points, and routers. NetSim can deploy up to 100,000 sensors simultaneously (Pro version) running both on IPv4 and IPv6, over a wide spectrum of wireless networks such as cellular, Wi-Fi, Long-Term Evolution (LTE), and even military radio bands. This tool can be installed on any physical or virtual machine, and through its intuitive UI, a user can drag-and-drop any of the aforementioned devices to form a simulation environment. Similar with the IoTIFY, the metric results are exported to log files and graphs. NetSim enables fine grained customization and through the “emulator” add-on, real hardware running can be engaged and tested with real traffic running through the simulator.

SimpleIoTSimulator
SimpleIoTSimulator is a licenced product offered by SimpleSoft; a company with a wide spectrum of Network tools [72]. It can be downloaded and installed on any 64-bit machine that runs RedHat Enterprise Linux9, however an online evaluation platform, with limited capabilities, is also available. The test environments created by SimpleIoTSimulator can operate under numerous communication protocols such as MQTT, CoAP, HTTP, and Modbus over IPv4 and IPv6. It supports up to 10,000 simulated devices that can be configured to publish messages at specific time intervals. An interesting feature offered by SimpleIoTSimulator is the “ability to learn” from real devices. This can be accomplished either by uploading a log file, or by observing in real-time the devices of interest. Thus, the simulated network is as realistic as possible.

**Node-RED**
Node-RED is a JavaScript cross-platform development tool created by IBM designed to connect hardware devices and online services. It can be installed locally or, deployed on the cloud and offers an intuitive user interface, which allows users to create a flow of functions in a matter of minutes. One can create as many MQTT, TCP, HTTP, WebSocket and UDP devices as desired simply by drag-n-dropping them in the “Flow” area. As usual, device parameters are fully customisable and given the provided range of capabilities such as functions, templates, and triggers, complicated experiments and scenarios are possible.

**IoTSimulator**
Developed by Bevywise, the IoTSimulator can be easily installed on any server environment and simulate thousands of sensors and devices. Through its web-based user interface, a user can create new networks, devices (bulk spawning available), and time-based events. Configuration options include device uptime, message frequency, and quality of service (QoS) [73]. IoTSimulator, comes with a plethora of predefined sensors and network templates. Similar to the majority of simulators, IoTSimulator is a licenced product that offers a free evaluation version that supports up to 100 devices.

**CupCarbon**
CupCarbon is a tool created by the research project “Persepteur10” and it is a wireless sensor network (WSN) simulator. It enables the user to define a network of sensors, mobiles (such as vehicles, unmanned aerial vehicles, and events [74]. Furthermore, via the OpenStreetMap framework integration, the WSN design and deployment is straightforward, making this Java-based simulator ideal for educational purposes as well [75]. The simulated sensor nodes support ZigBee, Long Range (LoRa), and Wi-Fi interfaces but with there is no configuration available over the communication protocol to be used. More specifically, the data captured and uploaded are exchanged based on multi-hop routing until they reach the “Base Station” [76].

**Cooja**
Cooja is the network simulator provided by Contiki, the open source operating system for the IoT. Cooja is designed to simulate Contiki motes, however we can define our own motes through the user interface. These simulated devices can be positioned in space either randomly, or in a user-specified way. Depending on the application run, they can send, receive, or broadcast messages in user-defined intervals. Cooja is protocol agnostic but some implementations include MQTT [77], CoAP [78], XMPP [79], and HTTP [77]. Given that Contiki operating system (OS) is open source

---

10 http://www.agence-nationale-recherche.fr/Project-ANR-14-CE24-0017
and provides full IP networking, one could port more communication protocols to Cooja. Finally, this simulator offers extensive network traffic reporting, mote energy consumption, and logging in real-time.

**AutoSim**

AutoSim\textsuperscript{11}, from Automatski, is a Java-based solution that enables the instantiation of multiple virtual nodes\textsuperscript{80}. Those nodes can communicate using a variety of protocols including AMQP, MQTT, XMPP, CoAP, REST, WebSockets, and LWM2M. The open source (for non-commercial projects) nature of AutoSim enables the users to modify the existing node templates, as well as, to introduce novel communication protocols.

At this stage of our study, we focus on the way the IoT communication protocols respond during crowdsourcing experiments and whether the experimental setup defined in Chapter 3 affects their performance in a “sanitised” environment. Furthermore, we are not testing the actual network, therefore we are not interested to examine scenarios like node-mobility, hand-offs, or congestions. Hence, we chose the *AutoSim* simulator as our preliminary evaluation tool, as it supports all the protocols we are interested in, is easily deployable and scalable, and allows extensive modification.

We installed AutoSim on our Virtual Machine server and by using client URL (cURL), we were able to post the necessary JavaScript Object Notation (JSON) messages which configured our experiment environment. In a similar way, we started and stopped the data exchange between the instantiated nodes and the server itself. AutoSim deployment is presented in the following section.

### 4.2 Simulator adjustments

As noted in Section 4.1, AutoSim simulator is open source and can be downloaded, installed, and run on both local and remote machines. An instance of this tool was deployed on a new virtual machine, different from the one hosting our DAC server. Before experiments could be run using AutoSim, a couple of changes needed to be made in order to be compliant with the system architecture defined in Chapter 3. More specifically, this simulator does not inherently support two-way communication between the IoT nodes and the server. When the nodes are spawned, the sensing parameters are already hardcoded and there is no way for the server to transmit messages to them, resulting in a very limited room for real-time adjustments. However, this limitation can be overcome by modifying the project classes that define the functionality of the devices together with the corresponding classes for all the examined protocols. Sections of code that provide extensive logging and communication capabilities to the nodes were added by the author. Now, each

---

\textsuperscript{11} [http://automatski.com/autosim.html](http://automatski.com/autosim.html)
individual node can be accessed either from the AutoSim, or the DAC server and monitor the exchanged messages. Furthermore, by sending a message that includes the necessary information the sensing parameters can be changed, even after all the nodes are spawned and ready to participate in simulations.

4.3 Sensing parameters

The tests were run using the configuration defined in Chapter 3 starting with a single node per protocol and then scaling it all the way to 100 devices. During our testing it was observed that the more the number of devices increased, the more issues were experienced by MongoDB, as too many connections were made simultaneously. In real life scenarios, when one is conducting large crowdsensing experiments, multiple servers or even cloud infrastructures are engaged to meet the needs imposed by the devices. Table 5 showcases the three different parameters that affect the simulation runs.

Table 5: Simulation-based evaluation parameters

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Parameter range</th>
</tr>
</thead>
<tbody>
<tr>
<td>IoT protocol</td>
<td>{XMPP, MQTT, REST, AMQP}</td>
</tr>
<tr>
<td>Sensing load (l)</td>
<td>{Light (n=1), Medium (n=3), Heavy (n=5)}</td>
</tr>
<tr>
<td>Devices (d)</td>
<td>{1, 3, 5}</td>
</tr>
</tbody>
</table>

4.4 Simulator results

At this point, focus is on the message delivery performance of the protocols under evaluation, as this is the main metric that the simulator offers. It is clear that energy consumption and message success often are linked together. This is due to the fact that failure of delivery will often lead to message retransmissions, depending on the quality of the service defined by the system or the user. According to the literature, the majority of crowdsourcing/sensing systems use mechanisms that maximise the delivery of the sampled data. Furthermore, it is accepted by the community that the more messages are broadcasted, the more energy will be needed by the end points and the system collectively [81].

Table 6 offers a summarised view of how the IoT protocols performed in terms of message delivery when the number of devices (d) is d=1. It comes with no surprise that all the IoT oriented protocols performed perfectly under the various load types with just one device in use.
Table 6: Message delivery percentages for devices d=1

<table>
<thead>
<tr>
<th></th>
<th>XMPP</th>
<th>MQTT</th>
<th>REST</th>
<th>AMQP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Light load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(n=1)</em></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Medium load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(n=3)</em></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Heavy load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(n=5)</em></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

*Devices in use d=1*

According to Table 7, as we scale our model to 50 devices, a swift of momentum for one of the protocols is noticed. During the light-load sensing, once again, the IoT protocols managed to transmit successfully all the measurements from all the devices. However, starting from the medium sensing load scenario, REST started to drop some messages resulting in a 98.46% delivery percentage.

Table 7: Message delivery percentages for devices d=50

<table>
<thead>
<tr>
<th></th>
<th>XMPP</th>
<th>MQTT</th>
<th>REST</th>
<th>AMQP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Light load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(n=1)</em></td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Medium load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(n=3)</em></td>
<td>100%</td>
<td>100%</td>
<td>98.46%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Heavy load</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>(n=5)</em></td>
<td>100%</td>
<td>100%</td>
<td>92.13%</td>
<td>100%</td>
</tr>
</tbody>
</table>

*Devices in use d=50*

The lack of REST’s support for queuing and QoS protocol becomes more noticeable throughout the last scenario. The message delivery recorded was even lower during the heavy load sensing, as just the 92.13% of the measurements was stored successfully in the MongoDB, as opposed to the “competition” who achieved 100% message delivery.

Table 8 summarises the results achieved by the four protocols when handling 100 devices simultaneously. All implementations except REST, were able to handle exceptionally the increased traffic throughout the three load scenarios, as they managed to broadcast all measurements without dropping not even one. Following the path that was paved when devices in use d=50, REST was subject to unsuccessful transmissions. More specifically, during the light load scenario, 99.32% of the messages were passed on and stored, while that number dropped to 94.57% for the medium load...
scenario. Finally, during the heavy load testing, REST transmitted successfully just 89.8% of the messages.

Table 8: Message delivery percentages for devices d=100

<table>
<thead>
<tr>
<th></th>
<th>XMPP</th>
<th>MQTT</th>
<th>REST</th>
<th>AMQP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light load</td>
<td>100%</td>
<td>100%</td>
<td>99.32%</td>
<td>100%</td>
</tr>
<tr>
<td>n=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium load</td>
<td>100%</td>
<td>100%</td>
<td>94.58%</td>
<td>100%</td>
</tr>
<tr>
<td>n=3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heavy load</td>
<td>100%</td>
<td>100%</td>
<td>89.8%</td>
<td>100%</td>
</tr>
<tr>
<td>n=5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Devices in use d=100

4.5 Discussion

The goal of this chapter is to investigate the performance of the selected IoT protocols in a series of different sensing loads, under a set of parameters. That would allow us to identify potential drawbacks that could affect crowdsensing experiments during more realistic scenarios or, when using physical devices.

From our findings presented in Tables Table 6, Table 7, and Table 8, it is apparent that XMPP, MQTT, and AMQP are suitable for small-scale, at least, crowdsourcing experiments. They faced no issue in delivering all the messages in the sanitised environment created by our simulation tool. On the other hand, REST’s performance was gradually deteriorating as the sensing load became more intense. Figure 9 depicts the decline of its efficiency for two different settings with number of virtual devices chosen as d=50 and d=100.

The almost-linear plummeting trend noticed is an indicator that using the REST protocol should be limited in systems that include retransmission techniques in order to counterbalance its error-prone nature. On the other hand, if the KPIs set by a system allow it, the 90% message delivery is nothing to be frowned upon.
4.6 Conclusion

Concluding this section, it is important to note that the behaviours observed throughout this chapter are unlikely to be reproduced with either physical devices or with any tool that provides mobility simulations. Nonetheless, the primary target of this chapter was not to evaluate how the assessed protocols handle mobility and handovers, but to measure their performance and ensure that they can handle the crowdsensing experiment scenarios defined in Chapter 3 of this study. To that extend, the simulation-based results demonstrate that the IoT communication protocols can support crowdsensing experiments with the exception of REST protocol, which seems to struggle as the parameters become more demanding.

Finally, the author needs to stress the fact that 100 devices might not be representative of a typical large-scale crowdsensing application. However, this maximum number of simulated devices was defined both by licensing factors that did not permit him to use more nodes and by the fact that the crowdsensing backend was hosted in a small VM slice. In real life scenarios, an experimenter would build additional queuing mechanisms and the backend would be hosted in a cloud infrastructure with multiple servers. In any case, as indicated, the target of assessing the IoT protocols in a sanitised environment is achieved even with smaller-scale simulation runs.
Chapter 5

Implementation-based Evaluation

This chapter extends the work presented in the previous chapter by introducing real devices during the crowdsensing experiments. The chapter starts with a brief discussion on both the software and the hardware of the selected devices. Later on, we introduce a powerful crowdsourcing tool that was designed and implemented by the author. It was installed in the selected devices and was responsible for generating and broadcasting data. We discuss about its communicational capabilities, privacy-preserving techniques, and crowdsensing functionalities in-depth. Section 5.1.2 reviews the smartphone-oriented assumptions made and reasons them. In Section 5.2, we analyse the methods available to measure the performance of the IoT communication protocols during crowdsensing experiments, covering both hardware and software approaches.

5.1 Crowdsourcing devices in your pocket

As discussed thoroughly in Chapter 2, commercial off-the-shelf smart devices nowadays are perfectly capable of participating even in the most complex opportunistic and participatory crowdsourcing experiments. Hence, at this point we also need to move the spotlight of this study to mobile crowdsourcing (MCS). In order to conduct our tests in the most realistic and fair way possible, we selected an assortment of devices running various versions of the Android OS, with different computational capabilities. As presented in Table 9, our diverse selection further supports the integrity and abstraction of the results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Brand</th>
<th>Android OS</th>
<th>CPU</th>
<th>RAM</th>
<th>More</th>
</tr>
</thead>
<tbody>
<tr>
<td>One S</td>
<td>HTC</td>
<td>4.1.1</td>
<td>Dual-core 1.5 GHz Qualcomm MSM8290</td>
<td>1 GB</td>
<td>Stock</td>
</tr>
<tr>
<td>Xperia S</td>
<td>Sony</td>
<td>4.4.2</td>
<td>Dual-core 1.5 GHz Qualcomm MSM8260</td>
<td>1 GB</td>
<td>Custom ROM</td>
</tr>
<tr>
<td>Nexus 5</td>
<td>LG</td>
<td>5.1.1</td>
<td>Quad-core 2.3 GHz Qualcomm MSM8974</td>
<td>2 GB</td>
<td>Stock ROM, Rooted</td>
</tr>
</tbody>
</table>

Table 9: Smart mobile devices used.
5. Implementation-based Evaluation

5.1.1 Crowdsourcing software

We present a powerful, yet energy aware tool, designed and implemented to extract socio-economic profiles and generated data from users and sensors. Using backwards compatibility libraries, it runs on Android OS v4.1 and higher (Code name: Jelly Bean), supporting 99.7% of active devices [82]. Furthermore, it offers advanced developer tools that provide insights on the application’s usage during crowdsourcing experiments. This tool was designed and developed by the author as part of the European research project called IoTLab\(^\text{12}\). However, for the purposes of this study, vital parts of the mobile application’s back-end had to be extended and re-configured. The following sections present in-detail the IoTLab application.

5.1.1.1 User profile

Many crowdsourcing studies such as [38], [83], [84] rely primarily on location when selecting users for crowdsourcing experiments. However, surveys like [85]–[87] demonstrate the importance of having a base knowledge of the social aspect of participants. Understanding peoples’ backgrounds and needs results to higher participation rates, increased data quality, and in some cases, lower monetary cost. Furthermore, the more details available to the experimenter about the user pool, the more advanced filtering could be performed to address explicit requirements set. Our tool completes socio-economic profiles by encouraging users to provide the details below:

- **Username**: Specifies the username of the new user.
- **Gender**: Specifies the gender of the user. (Male/Female/Other)
- **Age**: Specifies the age of the user.
- **Hometown**: Specifies the hometown of the user.
- **Country**: Specifies the country of the user.
- **Employment status**: Specifies the employment status of the user.
- **Employment sector**: Specifies the employment sector of the user.
- **Education level**: Specifies the education level of the user.

\(^\text{12}\) https://www.iotlab.eu/
Since the success of crowdsourcing experiments is knit together with participants’ satisfaction, none of these categories’ input is mandatory. To gain crowd’s trust, we allow users to choose the level of shared information. Moreover, as stated by Gustarini et al. in [15], crowdsensing users are reluctant to share their data even when they are promised anonymized location. Hence, we believe that by having additional filters available in the form of a socio-economic profile, a researcher can run sufficiently crowdsensing tasks without depending on the location of the user.

Socio-economic profiles are stored at a remote MySQL database, using RESTful web services. Table 10 presents the call made to the back end to register a new user. Notice that not all fields are provided.

Table 10: Registration of user's socio-economic profile

<table>
<thead>
<tr>
<th>Call</th>
<th>http://&lt;serverid&gt;/insertUser.php?username=fousekis&amp;email=<a href="mailto:nikos.koukos@surrey.ac.uk">nikos.koukos@surrey.ac.uk</a>&amp;roles_id=1&amp;providers_id=1&amp;age=30&amp;hometown=Thiva</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON Response:</td>
<td></td>
</tr>
<tr>
<td>Error Message:</td>
<td>{ &quot;Text&quot;: &quot;Fail&quot;</td>
</tr>
<tr>
<td>Success Message:</td>
<td>{ &quot;Text&quot;: &quot;Success&quot;, &quot;ID&quot;: &quot;21&quot;</td>
</tr>
</tbody>
</table>

The proposed system provides a way to update a user’s profile when desired, since most of the information is dynamic and can change over time. When a user adjusts something inside his profile,
a background synchronisation takes place which informs the MySQL database. Similar to the “insert user”, the “update user” functionality is done via RESTful web services as presented in Table 11.

**Table 11: Updating of user's socio-economic profile**

<table>
<thead>
<tr>
<th>Update Purpose:</th>
<th>Update the email and age of the user with id=3.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON Response</td>
<td><strong>Success</strong> Message: { &quot;Text&quot;: &quot;Success&quot; }</td>
</tr>
<tr>
<td></td>
<td><strong>Error</strong> Message: { &quot;Text&quot;: &quot;Fail&quot; }</td>
</tr>
</tbody>
</table>

According to Section 3.1.4, every device is represented in our resources directory in an abstract way. Each of its embedded sensors (accelerometer, magnetometer, thermometer etc) is registered using a background asynchronous task.13

In order to conduct our evaluation, we equipped our crowdsensing tool with the following communication protocol implementations:

- **MQTT**: messages were exchanged using the Paho project, which is an open-source client implementation of MQTT. Additionally, we extended the also-open-source MqttService for Android14, provided by Dirk Moors. When activated, the MQTT service runs on a background service, on a separate process.
- **AMQP**: our AMQP client was based on RabbitMQ’s Java client library15. Again, when connected, RabbitMQ client runs on a background service on a dedicated process.
- **HTTP**: the Apache HTTP library is included in the Android software development kit (SDK) by default. That enabled us to construct the appropriate messages to execute HTTP requests targeting our REST APIs.

---

14 [https://github.com/dirkmoors/MqttService](https://github.com/dirkmoors/MqttService)
- **XMPP**: we used Google Cloud Messaging as an XMPP client, which is inherently supported by Android OS devices that have Google services installed. More details are presented in Section 5.1.1.2.

### 5.1.1.2 Google Cloud Messaging

Google Cloud Messaging (GCM) is a service developed and provided by Google that allows notifications and messages to be exchanged between mobile smart devices and application servers [88]. This service offers downstream-only, or downstream and upstream communication depending on the server implementation; HTPP and XMPP-based respectively. The great advantage of this service is that it is inherently supported and run in the background by all Android OS devices that have Google Play Services installed. More specifically, an application can receive messages even when it is not running, as the system itself is the one responsible for notifying the targeted application so that it can handle the aforementioned messages. This reduces the computational needs of the developed software [89] since synchronisation services that run in the background, or keep alive messages, are redundant since they are handled by the OS.

The system that implements GCM has to register itself to the GCM server in order to receive its unique registration ID, same as every XMPP server. This ID serves as an identifier when we want to deliver a message to a specific device, or in the case of sending an ACK message. Even though XMPP does not offer QoS by default, GCM introduced a mechanism that notifies the device in case a message faced delivery problems, as well as a mechanism that automatically handles the sending retries when the network is offline. This was accomplished by extending the protocol with additional elements in the default XML stanza.

When the application is installed and launched, in order to preserve energy and to avoid interference during our experiments’ monitoring, both the MQTT and AMQP clients are inactive by default. However, they can be instantiated when the crowdsourcing tool receives the corresponding trigger prior to an experiment. As for HTPP, thanks to REST’s nature, there is no need to keep a connection alive in order to transmit any messages. Finally, Google Cloud Messaging (GCM) connections are handled automatically by the system. More specifically, each device has a unique ID, which is used to target messages, or notifications, to particular users. When a device receives a message via the GCM, the system “wakes-up” the application in-demand and delivers it.

All protocols run in separate background services, which are killed at the end of each experiment cycle. All suggested guidelines and principles set by Google and the respective libraries’ developers were considered to ensure the maximum energy efficiency in our application.
5.1.1.3 A hybrid protocol

AMQP’s energy demanding nature has been thoroughly documented in the literature. On the other hand, Google Cloud Messaging is in theory very energy efficient in idle mode, as it does not require any active connection. We are proposing a variation of our crowdsourcing tool that utilises both GCM and AMQP. The former protocol will be used when the DAC server wants to communicate with a device, while the latter when the device has to upload data. This approach will eliminate the need to keep an AMQP connection alive throughout the test runs, as the tool will establish and close a new one when necessary. The author believes that the inherent support of GCM on Android devices will counterbalance AMQP’s energy demanding nature that roots from the need to keep communication channels open. On the other hand, AMQP security features and message reliability might be important to specific crowdsensing scenarios, hence it was selected as the upload communication protocol.

5.1.1.4 Privacy by design

User privacy has always been an important aspect of crowdsensing systems whose level of adoption was usually under researchers’ discretion. However, since the end of May 2018, data protection and privacy for all individuals in the European Union is now regulated and imposed by law, according to the General Data Protection Regulation (GDPR) [90]. IoTLab was designed to comply with the GDPR in the following ways:

Right to be forgotten: IoTLab application offers the right to be forgotten inside the settings menu. This option will trigger the mechanism which deletes the anonymised socio-economic profile of the user from the storage server and the device itself, as displayed in Figure 11. A notifying message is then displayed which confirms the success of user’s request. Following that, the application returns to its default state, as illustrated in Figure 12.
Location data: as described in the literature, location is one of the most sensitive data that a participant can share during crowdsensing experiments. Malicious users can exploit them and extract knowledge concerning a user’s routine and habits. In our case, even though crowd-sensed data are not related to the participants, one could still find patterns of an anonymous individual by combining input from multiple sources and train an appropriate model. This risk could be mitigated by decreasing the granularity of the generated raw data. To do so, IoTLab application never shares a device’s coordinates as sampled by the GPS sensor but through an energy-efficient mechanism the data uploaded represent a neighbourhood-wide region with a radius of 500m. The granularity is easily adjusted and in case a user desires additional protection, his/her data can be mapped to a bigger radius.

User consent: Our crowdsourcing tool offers a multi-layer user consent mechanism that ensures that participants are always aware of their engagement in crowdsourcing tasks. The first layer is inherently provided by Android OS and asks for user’s permission on accessing their location. However, this is a one-off question whose answer might not reflect user’s point-of-view in the future. Moreover, the operating system does not alerting users about the exploitation of embedded
Implementation-based Evaluation

sensors such as accelerometer and light sensor. During our literature review, we encountered numerous studies that can infer user’s location and context with impressive accuracy just by leveraging on ambient and inertial sensors. Hence, we believe that additional consent layers are necessary. To this extent, even after getting a preliminary approval (via Android OS) from the user, all accessible-by-the-device sensors are opted-out from future experiments. A user should manually enable the embedded inertial, ambient, and positioning sensors from the corresponding section of the application’s menu. The third and fourth layers of consent consist of user notification mechanisms prior to and during opportunistic experiments. More specifically, the former includes the broadcast of a notification to all the users that were selected to participate to an experiment, 5 minutes before the start of it. That offers adequate time to a user to exclude themselves in case they want to avoid cellular data charges, minimise energy consumption, or simply because they do not desire to participate. The latter mechanism incorporates an ongoing (non-dismissible) message that notifies the user about any ongoing experiments and offers the option to withdraw, as illustrated in Figure 14 and Figure 15. Additionally, when the application is accessing device’s location through the GPS an additional notification is displayed, as illustrated in Figure 16.
5. Implementation-based Evaluation

5.1.1.5 Additional Features

The IoTLab application apart from extracting the socio-economic profile of a user and collect sensor data is equipped with a plethora of additional features. More specifically, a user participant can fetch all the available crowdsensing experiments and view them. On top of that, he/she can engage with the rest of the community by proposing new crowdsourcing ideas and by ranking the ones proposed by others. All the aforementioned are possible through an innovative user-interface that was continuously improved and polished thanks to following design guidelines and to feedback acquired by focus groups. The remaining of this section provides additional details to the additional features of the IoTLab smartphone application.

Participant profile, incentive and earnings

As discussed in Chapter 2, incentives are a great way to motivate users to engage in crowdsensing experiments. The introduced tool offers a dashboard which enables the user to see information concerning his/her participation. From the main screen of the application, using the side-menu one can navigate to this dashboard. Information such as username and join date are provided, together with a summary of the money earned via participation, as depicted in Figure 17. Moreover, the “My profile” screen is logging user activity in terms of personal earnings, donations, and the status of a proposed research idea.

![Figure 17: User profile.](image-url)
Research proposal and ranking
Another functionality offered by the introduced tool, is the research proposal and ranking. This allows users to express their satisfaction but also to share their ideas. Literature suggests that by engaging with the experiment platform a user feels empowered and part of it leading to higher levels of confidence, trust and participation.

Using the side navigation menu, a user can select the “Propose Idea” option (see Figure 18) which introduces him/her to a page where he/she can fill the necessary text fields and proceed to the next steps.

In order to submit research ideas, the following information is needed:

- **Title**: Specifies the title of the proposed research.
- **Description**: Specifies the details of the proposed research. Since it is a “long text” text filed, the user can elaborate on the suggested idea for as long as it is considered necessary.
- **Location**: Specifies the targeted location of the research.
- **Category**: Specifies the category of the idea through a pre-defined list.

Figure 18: Research Idea Proposal

In the same manner, the “Idea ranking” functionality is accessible through the main menu of the application. When a user selects the “Ideas” or the “Rank” section (see Figure 19), the application fetches all the ideas that have been proposed by other users of the platform using consuming a REST web service. If an idea is selected, the user is transferred to a new corresponding page.
This page contains all the details of the idea, such as description, location, and date. Additionally, the user rating is displayed, which is based on the votes that this idea has received from the rest of the crowd. This provides an estimation of how popular an idea might be.

A user can rank the showcased idea by selecting the star shaped button that floats on top of the ideas details as seen in Figure 20.
Research participation
As discussed in Section 5.1.1.1, the introduced crowdsensing tool can extract the socio-economic profile of a user in order to run targeted crowdsensing experiments. The system is able to invite users to participate to research by filtering their information and their preferences. When the system selects a user to participate to an experiment, it sends a confirmation request even if he/she has agreed to participate in all experiments from his/her settings menu. If the user accepts, then the device will start to send data in the background. This is to ensure that a user will always be aware and notified prior to the beginning of an experiment. As described in Section 0, during the course of a crowdsensing experiment, participating users are continuously informed of their engagement by the system, as depicted in Figure 21.

![Figure 21: System notifications during experiments.](image)
Map tools
Finally, the IoT Lab crowdsensing tool can display sensing resources on a map. More specifically, by accessing a REST API, the mobile application fetches all the resources in the form of a JSON feed and then loads them on a map, as illustrated in Figure 22.

5.1.1.6 Dissemination of the crowdsourcing tool in the world
Our crowdsourcing tool was uploaded to the Google Play store and it was available to download for free. In less than a year, it managed to attract more than 250 unique users and score more than 1,000 downloads in total. Furthermore, its outreach was remarkable, as people from more than 40 countries were running this tool. Figure 23 depicts in blue the countries of residence of IoTLab’s users.

Figure 22: Resources projected on a Map

Figure 23: Global outreach of our crowdsourcing tool.
Part of the work detailed in Sections 5.1.1.1 – 5.1.1.5 has also been presented by the author in the deliverable “D2.2 – Crowdsourcing Tools and Social Integration Report\(^{16}\)” composed in the context of the EU funded project IoT Lab.

5.1.2 Assumptions

To provide a fair comparison between the smartphones and the protocols, during the testing all background activities and tasks were limited. That was achieved by either removing all the unnecessary applications (bloatware), or by force-stopping the remaining non-vital ones. Furthermore, all devices were restored to factory state, with no additional applications installed. The crowdsourcing application was deployed, started, but was not running in the foreground. Its service\(^{17}\) was still alive, but the application itself was cleared by the recent task list\(^{18}\), where all the recent active applications are stored/displayed. Additionally, location services and the Bluetooth interface were deactivated. During our experiments the screen was off, and the reason was twofold:

1. Even though Android OS defines screen brightness using steps, or percentage, the actual screen performance differentiates between models and manufacturers, as each panel has different achievable maximum brightness. In plain words, the X% screen brightness of device Y, might be dimmer than the equivalent of device Z, or vice versa. That could potentially lead to inaccurate comparison in the energy efficiency aspect of the experiments.

2. As noted in [91] and [92], the display of a smartphone is by far its most energy demanding component. Despite contemporary screen technologies have introduced energy-efficient panels [93], the burden they impose to the battery is still much greater that the rest of the embedded hardware. As a result, during a crowdsensing experiment any additional energy consumption caused either by software, or hardware may be insignificant to the total energy demands of the device and consequently, can be overshadowed. Therefore, big chipset manufacturers such as Intel [94] and Qualcomm [92], even software providers like Android [95], advise to keep the screen deactivated when applying energy consumption monitoring techniques.

\(^{16}\) https://www.iotlab.eu/IOTLabProject/OpenDeliverables?name=D2.2
\(^{17}\) https://developer.android.com/guide/components/services
\(^{18}\) https://developer.android.com/guide/components/activities/recents
Our devices were connected to the Internet through the Wi-Fi interface. The usage of cellular data was avoided as network coverage fluctuates and consequently the energy consumption of the connected devices. Network’s latency was monitored to ensure fair comparison of the protocols.

5.2 Monitoring energy consumption

One can find numerous power measuring and profiling tools in the literature. Some of them provide great accuracy, others offer large component-level support, and some have prerequisites in order to run. They can be classified into two main groups: hardware and software-based solutions. The former solutions are robust and provide the most reliable results, while the latter category typically offers easier deployment.

5.2.1 Hardware solutions

Using hardware monitoring systems to monitor and log energy consumption on smart devices is consistently accurate, nonetheless it introduces certain limitations. First and foremost, in most cases a physical modification to the device is needed. Researchers would need to remove the battery and directly connect cables with power supply [96][97][98], which may render the test devices unusable, as modern smartphones do not equip a removable battery. Furthermore, external components such as clamps meters [83], sense resistors on the power supply rails [78], and resistances between the power supply and the device (in the form of a multi-meter) [6] are bulky approaches which cannot be replicated in real-life testing scenarios. Finally, hardware solutions may experience issues when monitoring and profiling the energy consumption of a single component/application on a device. This is due to the fact that hardware equipment obtain voltage and current readings to estimate the overall energy consumption of a device. In order to identify the consumption of a specific component or application, the base energy consumption of an idle device is established and then subtracted from the readings sampled during experimental runs. Even though it is adequately accurate, the base energy consumption of complex devices like modern smartphones may fluctuate as they embed dozens of micro-electro-mechanical systems which behave differently depending on external parameters such as signal strength, temperature etc.

5.2.2 Software solutions

Using Software monitoring solutions is an active research topic [96], [99]–[103] since the majority of hardware solutions are difficult to be applied in real-life scenarios. Numerous software tools are used to model and predict the energy consumption of a device based on a series of parameters. However, at this stage, modelling and predicting the energy requirements of the proposed crowdsensing tools are out of scope. The required tool must provide robust real-time data hence,
the solutions proposed by the research community are excluded. A list of applications and methods to monitor the performance of a mobile device that could potentially be used is as follows:

Android OS can provide battery stats to an application in real-time by accessing the `BatteryManager` API. Information concerning battery health, voltage, capacity, current, and charging status are available among others. However, some of those metrics require root privileges to be accessed [99] and additionally, this might not be the most suitable approach for real-time monitoring as battery stats are updated randomly. On the other hand, it is an excellent choice for developers/researchers that look for a simplistic and quick way to profile energy consumption.

**Snapdragon profiler** is a free desktop tool created by Qualcomm that enables developers and researchers to identify bottlenecks and optimise Android applications [104]. It is designed for devices that are equipped with Snapdragon processors, run at least Android 5.0, and have an active Android Debug Bridge (ADB) connection (via USB or, Wi-Fi). One can monitor in real-time or store for later analysis metrics concerning CPU, GPU, memory, network, thermal, and power performance. Research community rely on Snapdragon profiler thanks to the advanced level of details it provides [105], [106]. Finally, even though Snapdragon profiler does not officially support non-Snapdragon devices, some basic functionality may exist on them.

**Android profiler** is a performance tool integrated to Android Studio, the official IDE of Android OS and runs on Unix-based, Mac, and Windows computers. It profiles and displays information in real-time concerning an application’s behaviour in aspects such as CPU, network, RAM, and energy consumption. The amount of details offered is one of the best available. However, some of these data are not available to all the devices. According to Google, to enable the “advanced profiling” a physical or, an emulated device should run Android 8.0 (API 26) or higher, which as of the end September 2018 represents only the 19.2% of active devices [82]. Furthermore, a device has to be connected through ADB to provide access to the Android profiler.

**Nokia-Energy Profiler** (NEP) was a powerful stand-alone profiling tool developed by Nokia and supported Symbian S60 devices. It could be run in the background of the device and collect a wide spectrum of data including RAM, CPU load, network speed, signal levels, energy consumption, and battery voltage. It was able to capture measurements with a 4Hz frequency and export them locally for further analysis. However, as stated by Ahmad et al. in [102], NEP adds a significant energy overhead and it should be taken under consideration when used, as it may misrepresent the

---

energy consumption of the device. During the past decade it was widely used by the research community [28][32][39][8][97], [107]–[109] however, it came to its end-of-life together with the discontinuation of Symbian OS [110] by Microsoft.

**Trepn Profiler** [111] is an Android application, developed by Qualcomm that is free to download and use. It offers both device-wide and application-targeted profiling and it can be deployed to most Android smart phones and tablets. When used on devices that are equipped with a Qualcomm Snapdragon processor, additional monitoring parameters are accessible. It is capable of profiling CPU (load and frequency per core, or total), GPU (load and frequency), network interfaces (mobile data state, Wi-Fi received signal strength indication (RSSI) and state, Bluetooth state), RAM, temperature, GPS, and screen (state and brightness) [112]. Additionally, it captures the current battery power and level. These data points can be sampled on profiling intervals as short as 100 milliseconds and exported on a local database or a .csv document for later analysis. Despite that Trepn can offer visual representation (graphs, charts, and readings) of performance at runtime, this option was not utilized according to Section 5.1.2.

**GSam Battery Monitor** [113] is an all-around energy-oriented Android application. It is aimed at both enthusiast and non-enthusiast users as it offers a wide spectrum of functionalities. Its main purpose is to monitor the system-wide energy consumption and notify the user if an application is draining the battery. This is achieved by detecting the *wakelocks* to20 together with battery stats that come through Android API, as presented earlier in this sub-section. GSam can monitor CPU usage, network data consumption, sensor engagement time, phone signal, battery temperature, and percentage of power used per application. To gain access to all these statistics root permissions are required. Even though it is a commercial app, it is also used by the research community in studies such as [114]–[116].

At this point it is important to underline that the tools that access application power stats via an active ADB connection are not suitable for our study. That is because an active ADB connection requires a smart device to be connected to a computer either via USB cable or, over the Wi-Fi. Both scenarios are interfering with our experimental scenarios in the following ways:

- When a USB cable is attached to Android devices, they are automatically charged. Even though the conventional USB 2.0 port is providing only 500mA, this current is still enough to alter the device’s behaviour. Workarounds suggest to manually modify system files to

---

20 [https://developer.android.com/training/scheduling/wakelock](https://developer.android.com/training/scheduling/wakelock)
“trick” the device into believing it is not charging. Nonetheless, the battery will continue to receive charge.

- Recent Android devices offer the option to establish an ADB connection over Wi-Fi. Despite the fact that this is not interfering with the battery status, it keeps the Wi-Fi interface not only enabled, but also in transfer state. As discussed in another section of this study, different IoT communication protocols have different connection preferences. By keeping the Wi-Fi interface active we risk ignoring the energy consumption spikes associated with how a protocol handles data transfers.

5.3 Conclusion

This chapter presented IoTLab, a powerful, privacy-aware crowdsensing tool. Thanks to its features, it can support crowdsensing tasks in a secure and transparent way. IoTLab smartphone application was designed and implemented by the author and offers a wide range of functionalities that are tied with crowdsourcing and crowdsensing platforms such as push notifications, ranking system, and data collection using REST, GCM, and AMQP. For the purposes of this study, the communication protocols were extended to include MQTT as well. IoTLab was the official mobile application delivered by the University of Surrey for the EU research project IoTLab21.

Additionally, this chapter includes a study on monitoring the performance of mobile devices was. Based on the literature, it is evident that we need an in-device profiler that provides great level of detail and per-application analysis. To the best of our knowledge, the most appropriate tool is the Trepn Profiler [107] which does not require an ADB connection and can target a specific application that runs on a mobile device, without being affected by the rest of the background services, or processes [113].

21 https://www.iotlab.eu/
Chapter 6

Results and Discussion

This chapter includes a detailed analysis of the acquired measurements during the crowdsensing experiments conducted on mobile smartphones. The chapter initially presents summarised views of the results recorded during the different experimental set-ups in accordance with section 3.2 of this study. We analyse the energy and message delivery results per protocol and reason them. Later, we introduce a novel crowdsensing approach that incorporates sensing triggers in order to achieve increased message delivery success. Following the introduction, a proof-of-concept assessment was performed, where smartphones were set-up and run under the same set of parameters. Finally, there is a discussion comparing two types of sensing, energy consumption, and message delivery.

6.1 Summarised evaluation review

Chapter 5 introduced the means and devices used for evaluating Internet of Things protocols in Android phones/tablets. From our literature review, we established that the most suitable software profiling tool is the Trepn Profiler. Trepn profiler can run efficiently on all three testing devices by default as they are powered by a Snapdragon processor. However, the HTC and Sony smartphones provide power metrics inferred by energy consumption models, which introduce a margin of error. On the other hand, the LG device enables direct power readings, thanks to the Android version it runs and to the chipset it embeds. Hence, this chapter mainly presents results based on experimental runs conducted on LG Nexus 5 smartphones. To eliminate device-specific errors, we used a total of six Nexus devices in the course of this study.

Despite the fact that we predominantly focus on the LG & Trepn configuration, we also carried a series of experiments that prove the generality of our scenarios and measurements. More specifically, during the test runs on all three device models, we measured the energy consumption by accessing the battery stats API provided by Android OS which were presented in Section 5.2.2. Even though this approach is not the most robust one available today, it is sufficient to detect patterns among the devices and the protocols’ performance. The results of these runs are depicted in Table 12.
Table 12: Median Energy consumption during preliminary and generalisation runs.

<table>
<thead>
<tr>
<th>Sensor Type</th>
<th>REST</th>
<th>MQTT</th>
<th>GCM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Light</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Sony Xperia</td>
<td>0.27592</td>
<td>0.32496</td>
<td>0.36607</td>
</tr>
<tr>
<td>HTC One S</td>
<td>0.2692</td>
<td>0.2925</td>
<td>0.3371</td>
</tr>
<tr>
<td>LG Nexus 5</td>
<td>0.21093</td>
<td>0.30527</td>
<td>0.42076</td>
</tr>
</tbody>
</table>

Energy consumption in W

Green indicates the protocol that had the least impact on the energy consumption of our devices for a given load setting. For example, during the light-load sensing experiment using the Sony Xperia device, REST was the least energy demanding. It is evident that all three devices present similar trends throughout the crowdsensing scenarios, regardless of the accuracy of the obtained power measurements. This allows us to believe that the power data gathered from one device model will reflect the behaviour of the rest of the device models.

The results presented in this chapter are based on 239 trials run using the crowdsensing system the author built for the purposes of this study. This number includes official, controlled, and extended runs that were used to validate the overall robustness and fairness of the system.

### 6.1.1 Idle state

Crowdsensing experiments take most of their toll on the battery of constrained devices during the sampling and broadcasting phase. Nonetheless, one must not overlook the power needs of a crowdsensing system while in idle state. As discussed earlier in this study, certain protocols require frequent exchange of messages to keep a communication channel active. Keep-alive messages can add considerable burden over time since they actively engage the CPU and the network interfaces. This section presents the energy consumption associated with the IoT protocols under examination. As depicted by Figure 24, Google Cloud Messaging is the most energy efficient one during idle mode, followed by MQTT with 0.17W and 0.19W average energy consumption. The most demanding was RabbitMQ that consumed 0.2W in average. The RabbitMQ + GCM hybrid and the
REST protocols were skipped by this chart, as they consume the same amount of energy with GCM in idle state.

![Mean Energy Consumption per Protocol](image1)

Figure 24: Energy consumption per protocol during idle state

Figure 25 illustrates the mean energy consumption of each protocol during the light-load sensing scenario. The most energy efficient was MQTT, followed by the RabbitMQ + GCM hybrid with mean energy consumption of 0.42W and 0.43W correspondingly. On the other hand, the most energy demanding protocol was the AMQP (RabbitMQ client) which needed 0.54W of power.

![Mean Energy Consumption per Protocol](image2)

Figure 25: Energy consumption per protocol during light-load sensing, sensors used: n=1
As discussed earlier in this study, power is just one aspect of the crowdsensing that interests the researchers. Message delivery is another one and it is a key factor for some experiments, depending on their nature and their goals. As seen in Figure 26, among the evaluated protocols, MQTT was the most robust offering 97.8% message delivery, followed by GCM with 97.5%. The third best was RabbitMQ scoring 95.16%, while REST and the RabbitMQ + GCM hybrid achieved 94.4% and 93.9% message exchange respectively.

The medium-load sensing, as expected, marked a notable increase in the average energy consumption of the protocols, except for REST and the RabbitMQ + GCM hybrid. More specifically, GCM was the most demanding (0.65W), while REST was the most efficient (0.45W).
Despite consuming the most energy, GCM had the smallest delivery percentage (54.6%) as illustrated by Figure 28. In contrast, the RabbitMQ + GCM broadcasted successfully 94.6% of the sensor measurements back to the DAC server. Second best was REST with 94.49% delivery success, followed by Rabbit and MQTT with 93.06% and 88.7% respectively.

![Message Delivery Success](image)

Figure 28: Message delivery per protocol during medium-load sensing, sensors used: n=3

During the high-load sensing scenario, REST was once again the most energy efficient protocol consuming 16.7% less power from RabbitMQ, which was the second best (0.45W and 0.54W correspondingly). Same with the last discussed scenario, GCM was the most demanding with average consumption 0.67W, 14.39% and 14.5% higher than MQTT and the Rabbit + GCM hybrid.

![Mean Energy Consumption per Protocol](image)

Figure 29: Energy consumption per protocol during heavy-load sensing, sensors used: n=5
As for the message delivery, three out of five of the evaluated protocols achieved higher scores compared with the medium and light load scenarios, according to Figure 30. More specifically, RabbitMQ and the RabbitMQ + GCM hybrid were almost perfect with 99.5% success, followed by REST with 97.4%.

GCM was the least reliable achieving to deliver just the 42.5% of the measurements, while MQTT delivered the 67.2% of them, making them 57.2% and 32.4% less effective than RabbitMQ respectively.

The following sections provide a detailed analysis of each of the protocols individually.

### 6.1.2 GCM

GCM proved to be a reliable choice for sensors n=1, as it achieved the second-best message delivery percentage (97.5%), just 0.5% behind MQTT. However, when the parameters were more demanding it failed to deliver a big portion of the intended measurements. More specifically, as illustrated clearly in Figure 31, the delivery percentage plummeted to 54.6% and to 42.5% for sensors n=3 and n=5 respectively.
Google Cloud Messaging did not excel in energy consumption either. Compared with the most efficient in each case, GCM was more demanding by 20.2% for sensors $n=1$, 45.0% for $n=3$, and 47.8% for $n=5$, as depicted in Figure 32.

To summarise, GCM was the either most energy demanding protocol, or the second-most one, while it failed to deliver a great portion of the measurements. The only exception was during the light-load scenario, when it proved to be quite robust.
6.1.3 MQTT

MQTT was the most reliable communication protocol during the light-load sensing experiment, as it delivered 97.8% of the messages. At the same time, it managed to keep the power demands lower than the competition, rendering it the best candidate for light-load scenarios.

![Figure 33: MQTT message delivery trend](image1)

However, for sensors used \( n=3 \), MQTT was not as reliable by delivering 88.7% of the messages. It was second from the end, 6.23% worse than the best of the category. Meanwhile, it was more demanding compared to REST by 13.3% with mean energy consumption equal to 0.51W.

![Figure 34: MQTT energy consumption trend](image2)
During the heavy-load sensing scenario, MQTT kept delivering less messages (67.2%) while consuming even more power (0.57W) as illustrated in Figure 33 and Figure 34. These results make MQTT the second worst in both categories.

Concluding, similar with GCM, MQTT seems like an excellent choice for light-load sensing scenarios as it was both the most reliable and the most energy efficient. However, it is not well-suited for medium and heavy load sensing.

6.1.4 REST

Despite that REST’s 94.4% message delivery success was the second-worst during the light-load sensing scenario just ahead of the RabbitMQ + GCM hybrid. However, as shown in Figure 35, the more demanding the sensing became, the more reliable it turned to be achieving 94.49% and 97.4% successful broadcasts for the medium and heavy-load sensing. During the former scenario, it was just 0.1% worse from the first one, and 2.11% during the latter.

In terms of power, REST managed to keep the consumption rather stable as the relative increase from sensors used $n=1$ to $n=5$ was just 3.18%. At the same time, compared with the rest of the protocols, it was the second best behind MQTT during the light-load sensing, and the most efficient one during the medium and heavy-load sensing with 0.44W, 0.45W, and 0.454W corresponding average energy consumptions.
To sum up, REST was able to handle quite remarkably the progressive overload imposed to it, while keeping the power demands low, making it suitable for all types of crowdsensing scenarios.

### 6.1.5 RabbitMQ

As depicted in Figure 37, RabbitMQ dropped 4.8% of the measurements during the light-load sensing scenario and consequently was ranked third amongst the other solutions. Moreover, its message delivery performance dropped to 93.06% for sensors used $n=3$ and once again was the third best choice. However, there was a swift on the delivery percentage trend during the heavy-load sensing, as RabbitMQ managed to be almost perfect by successfully transporting 99.5% of the messages making it the best choice, together with the RabbitMQ + GCM hybrid, as illustrated in Figure 37.
RabbitMQ demonstrated an interesting behaviour in the power aspect of the evaluation experiments. More specifically, as depicted in Figure 38 the energy consumption trend was flat, with almost identical average energy demands for the light and heavy-load sensing (0.547W and 0.546W respectively). During the medium load sensing, a relative decrease by 3.22% was marked. Compared with the rest of the protocols, for n=1, RabbitMQ was the least favourable choice, for n=3 was the second worst one, while for n=5 it was the second best.

Overall, RabbitMQ achieved a high delivery rate (93%-99.5%) while consuming almost the same amount of power throughout the different sensing scenarios. It is noticeable that despite being more demanding than the competition for light-load experiments, it is the best choice for heavy-load ones.

6.1.6 RabbitMQ + GCM

The RabbitMQ + GCM hybrid is the last examined solution. According to Figure 39, hybrid’s performance improved gradually as the parameters became more demanding, achieving 99.5% during the high-load sensing scenario. For sensors in use n=1, it was the least reliable solution compared to the rest protocols with just 93.9% delivery success. However, it was the most stable of all for medium and heavy-load sensing by achieving 94.6% message delivery during the former scenario and 99.5% during the latter one.
On the power side, the RabbitMQ + GCM hybrid was the second most efficient during the light and medium-load scenarios with 0.43W and 0.49W average energy consumptions respectively. On the high-load scenario, it was placed third with mean energy consumption equal to 0.57W.

The RabbitMQ + GCM hybrid seems like an appropriate choice for medium and high-load crowdsensing scenarios, as it manages to deliver the most messages in comparison to the other protocols, while being the third most energy efficient approach.
6.2 A new sensing approach

The previous sections of this study presented in-depth the burst-type of collecting data from mobile smart devices. All five solutions performed inside a wide spectrum both in terms of message reliability and energy consumption. During the light and the medium-load sensing scenarios, the message delivery percentage was far inferior than the one we achieved during our simulation tests in Chapter 4. Thus, we believe that this aspect of the mobile crowdsensing aspect should be investigated further.

We propose another approach to mobile crowdsensing that aims on alleviating the performance issues concerning message delivery. During the burst-type of sensing, the mechanism that is responsible to wake the device, sample and upload generated data is embedded to the smartphone application and is based on timers. The main concept is modifying the sampling technique by removing the interval timers from inside the smartphone application, as they seem to suffer from the Android OS power optimization. More specifically, in recent versions of the Android OS, Google has been continuously becoming more “aggressive” against all background tasks in an attempt to extend the battery life of the devices. This frequently leads to killed processes such as the one responsible for the opportunistic sensing presented in this study. Hence, we propose the usage of a triggered-based type of sensing instead of the conventional burst one. The aforesaid timers will be set on the DAC server and when appropriate, a corresponding message will notify the mobile devices. The updated message flows are demonstrated in Figure 41 and Figure 42.

![Diagram](image-url)

Figure 41: Message sequence during triggered XMPP, AMQP, and MQTT experiments
We believe that removing the timers from inside the mobile crowdsensing application will result in improved message delivery to the DAC server and the MongoDB database respectively. Conversely, we predict an augmented energy consumption compared to the burst type of experiment, as during this constant two-way communication, the IoT devices will consume double the incoming messages.

The first stage of the experiment remains the same, when the DAC server notifies the smart devices about the upcoming crowdsensing experiments. This allows the mobile application to set-up the necessary communication protocol client. After that, the sensing loop begins, where the DAC server sends, at specific intervals, sensing triggers to the message brokers. When the sensing triggers are finally delivered to the IoT devices, the system wakes up in order to consume the incoming message. At this point, the crowdsensing application generates the response and pushes it forward to the broker. When the DAC server receives the message, it then stores the crowdsensing data to the MongoDB – in a similar way with what happens with the burst type of experiments.

Figure 42 illustrates the message sequence of the proposed approach during crowdsensing experiments that use RESTful web services to store the generated data. The devices examined, due to their mobile nature do not have a static IP address. Hence, we cannot access them directly using a REST API in order to send the sensing triggers. To overcome this obstacle, we may use one of the rest of the evaluated protocols as the mean of passing the trigger. We selected Google Cloud Messaging, as it is the only one that is inherently supported in all devices officially supplied.
globally. Additionally, as mentioned in an earlier section of this study, it is the only communication protocol that is not imposing an additional energy burden to the device while being idle, as the applications using it do not have to keep any connection alive.

Hence, during the sensing loop, all the triggers are sent to the IoT device using GCM, while the device is storing the crowdsensing data directly to our database using the corresponding RESTful web services.

### 6.2.1 Evaluation of the new approach

In this section, we evaluate the proposed crowdsensing approach against the same experiment parameters defined in Section 3.2. Additionally, the parameters set are identical to the ones imposed in Section 5.1.2.

As illustrated in Figure 43, MQTT and REST were the two protocols who achieved the lowest energy consumption (0.46W) during the light-load sensing scenario, followed by RabbitMQ (0.52W), the RabbitMQ + GCM (0.55W) hybrid, and finally GCM (0.57W). As predicted, there is a slight increase in the energy demands of this sensing approach compared with the burst type. More specifically, energy consumption grew by 9.7% for MQTT, 6.4% for REST, 25.6% for the RabbitMQ + GCM hybrid, and 13.5% for GCM. On the contrary, RabbitMQ needed 4.6% less energy than the burst type of experiment.

![Mean Energy Consumption per Protocol](image-url)

Figure 43: Energy consumption per protocol during light-load triggered sensing, sensors used: $n=1$
Figure 44 presents some interesting results in the area of message delivery success, that may counterbalance the newly imposed energy demands of the triggered type of sensing. All five of our protocol variations managed to successfully transmit the crowdsensing measurements to the full extend achieving 100% message delivery, which is 2.5%, 2.2%, 5.6%, 4.84%, 6.1% better for the GCM, MQTT, REST, RabbitMQ, and the RabbitMQ + GCM hybrid respectively, compared to the burst approach.

The mean energy consumption per protocol during the medium-load sensing scenario is depicted in Figure 45, when RabbitMQ was the least energy demanding one (0.54W). The second best was once again REST (0.58W), while this time MQTT was the third one (0.61W).
Similar to the light-load sensing, the RabbitMQ + GCM hybrid and GCM were the two worst with 0.62W and 0.67W mean consumption. In other words, GCM, MQTT, REST, and the RabbitMQ + GCM hybrid had increased energy demands equal to 3.83%, 20.4%, 29.87%, and 25.38% correspondingly compared to the burst-sensing approach. RabbitMQ was the only protocol which marked a decreased consumption by -2.21%.

When it comes to the message delivery success, the medium-load scenario results follow the path set by the light-load one. More specifically, as seen in Figure 46, all protocols, except GCM, did not lose any crowdsensing measurements achieving 99.6%-100% message delivery. Furthermore, despite the fact that GCM managed to only broadcast 85.15% of the messages – the worst compared to the rest protocols, it is still a great improvement compared to the 54.6% of the burst-type sensing. The remaining protocols improved their delivery percentage as follows: MQTT by 10.9%, REST by 5.41%, RabbitMQ by 6.94%, and the RabbitMQ + GCM hybrid by 5.4%.

![Message Delivery Success](image)

Figure 46: Message delivery per protocol during medium-load triggered sensing, sensors used: n=3

Figure 47 and Figure 48 illustrate the protocols’ behaviours during the heavy-load triggered sensing scenario. It is evident that GCM was the most energy demanding as it’s mean consumption skyrocketed to 0.72W, while the corresponding delivery percentage was 81.7%. Despite that GCM reported a 10.3% increase in energy consumption it showcased it was better at transmitting the crowdsensing data by 49.63%.
The remaining four protocols however reported a notable decrease in terms of energy consumption, while maintaining the message delivery percentage to perfect – or almost perfect – levels compared to the medium-load sensing. More specifically, energy demands reduced by 19.37% for the MQTT, by 11.08% for the RabbitMQ, and by 4.57% for the RabbitMQ + GCM variation.

The remaining of this section presents an in-depth review of each protocol’s behaviour under our experimental scenarios.
6. Results and Discussion

GCM
During the new proposed sensing approach, Google cloud messaging performed in a similar way as in the original burst-type one. More specifically, as seen in Figure 49, it performed adequately in light-load sensing scenarios, but gradually the message delivery percentage dropped from the initial 100% to 85.15% and finally to 81.7% for the medium and heavy load sensing correspondingly.

![Message Delivery Trend](image)

Figure 49: GCM message delivery trend (triggered sensing)

GCM was the most energy demanding protocol amongst all throughout the different scenarios. Figure 50 illustrates clearly the energy consumption trend from light to heavy-load sensing, which is linear and reached the 0.72W for sensors n=5.

![Energy Consumption Trend](image)

Figure 50: GCM energy consumption trend (triggered sensing)
Summarising, this protocol consumed the most power throughout our scenarios and was the least successful at delivering the sensor measurements.

**MQTT**

As presented in Figure 51 MQTT achieved excellent results during our triggered sensing scenario independent of the sensing load. More specifically, it was perfect during the light-load sensing (100%), while for the medium-load one it managed to deliver 99.6% of the messages, and 99.9% during the heavy-load scenario. Compared to the burst type, the triggered type of sensing approved MQTT’s message delivery efficiency by 2.24%, 12.28%, 44.6% for sensors $n=1$, $n=3$, and $n=5$ correspondingly.

![Message Delivery Trend](image1)

*Figure 51: MQTT message delivery trend (triggered sensing)*

Energy-wise, MQTT was the least demanding amongst the competition during the light and heavy load sensing with 0.46W and 0.49W mean energy consumption. However, throughout the medium-load scenario, the average power demands climbed to 0.61W, making MQTT the third worst option.

![Energy Consumption Trend](image2)

*Figure 52: MQTT energy consumption trend (triggered sensing)*
For sensors \( n=1 \) and \( n=3 \), the triggered-sensing MQTT was more energy demanding compared to the burst-sensing MQTT by 9.6% and 20.5%, but at the same time it ensured a substantially superior message delivery. On the other hand, for sensors \( n=5 \), our new approach was more energy-efficient by 13.5%.

Overall, MQTT’s message delivery was spotless throughout our evaluation experiments, while it managed to be least energy demanding when sensors used \( n=1 \) and \( n=5 \).

**REST**

REST’s message delivery trend is quite similar to the MQTT’s, as it was nearly perfect during our different type of sensing loads. More specifically, for light and heavy sensing, it managed to deliver 100% of the messages intended, while during the medium-load scenario it dropped just a limited number of them, achieving 99.9% message delivery, as depicted in Figure 53.

As discussed in the previous section, REST was together with MQTT the most energy efficient protocols during the light-load sensing scenario with mean energy consumption 0.46W, just 0.8% increase from MQTT. Additionally, for sensors used \( n=3 \), it was the second best overall (0.58W) and the third when \( n=5 \) (0.51W). Compared with the original type of sensing, for the light, medium, and heavy load sensing we observed an increase in the energy consumption by 6.19%, 29.87%, and 14.33% respectively.
Again, the counterbalance of the extended energy burden is the improved message delivery. Compared with the burst scenario, REST’s robustness increased by 5.9%, 5.8%, and 2.66% during the different loads respectively.

Summarising REST’s performance, one can clearly identify that it is a low energy consuming protocol which offers a reliable message transportation, even under high stress scenarios.

**RabbitMQ**
RabbitMQ was the only protocol when used that ensured a completely perfect message delivery throughout our different sensing loads, as illustrated in Figure 55. Compared with the burst-type scenario, these results consist of an increase by 5.9% for the light-load, 5.8% for the medium-load, and 2.67% for the heavy-load sensing.

![Energy Consumption Trend](image1)

**Figure 54:** REST energy consumption trend (triggered sensing)

![Message Delivery Trend](image2)

**Figure 55:** RabbitMQ message delivery trend (triggered sensing)
For sensors used \( n=1 \), RabbitMQ was the third best option with average consumption of 0.52W, 11.5% more demanding than MQTT, but 4.74% more efficient than its burst-type counterpart. For \( n=3 \), RabbitMQ was the optimum protocol by being 7.8% more power efficient than the second best (REST), while for \( n=5 \) it was the second behind MQTT with just 3.1% greater energy consumption (0.51W).

As seen in figure RabbitMQ’s energy consumption trend did not fluctuate a lot throughout our various scenarios, a behaviour that was expected as it has already been observed during the burst-type runs.

![Energy Consumption Trend](image)

**Figure 56: RabbitMQ energy consumption trend (triggered sensing)**

Concluding, RabbitMQ demonstrated again that message load does not affect energy consumption drastically as the energy consumption trend remained almost flat. Additionally, it is important to stress the fact that during the light and the heavy load sensing RabbitMQ required less energy from its burst-type equivalent, while handling double the messages.

**RabbitMQ + GCM**

It comes as no surprise that the RabbitMQ + GCM hybrid managed to achieve a perfect message delivery during our triggered sensing approach. During the last section, RabbitMQ prove to be robust even for sensors used \( n=5 \), while GCM had no issues with light-load scenarios. Hence, when we use GCM for the triggering and RabbitMQ for the actual sensor measurements it is expected that the hybrid would perform well.
Respectively, the hybrid will follow the power trends set by the parts it consists of. Hence, the hybrid’s power trend is not flat, as depicted in Figure 58. The Rabbit + GCM hybrid was consistently the most-demanding protocol second to GCM, with average energy consumption equal with 0.55W, 0.62W, and 0.59W for n=1, n=3, and n=5 respectively.

Summarizing, compared to the burst-type approach, message delivery during the three sensing loads was increased by 6.49%, 5.71%, and 0.5%. Likewise, the energy needed raised by 25.6%, 25.37%, and 2.64%.
Overall, despite the excellent message delivery that the RabbitMQ + GCM hybrid offers, the increased energy consumption observed throughout the sensing loads renders it an inappropriate choice for the proposed sensing approach.

### 6.3 Discussion

The previous two sections of this chapter presented a detailed analysis of the outcomes of the conducted crowdsensing experiments. All results were demonstrated as they were, with no in-depth interpretation of the system’s behaviour. The remainder of this chapter will consist of an attempt to clarify the behaviour of the protocols and the impact of the results in real-life terminology.

#### 6.3.1 Power trends justification

During the protocol evaluation experiments, an interesting behaviour was observed in the aspect of energy consumption as the sensing load increased. More specifically, as depicted in Figure 59, the energy consumption of all protocols except RabbitMQ demonstrated an upward trend. This increase ranged from 3.1% (REST) all the way up to 35% (GCM).

![Power Trend per Protocol](image)

Same with the burst-one, the triggered sensing approach presented a similar change in the power demands of each protocol from sensors used n=1 to n=3. As presented in Figure 60, all protocols are more resource-demanding during the medium-load sensing. However, during the heavy-load run, all protocols except GCM were less power consuming.
6. Results and Discussion

This swift of momentum can be explained by the experiment parameters together with the way network interface of the device works. In particular, when it comes to three and five sensors runs, the sensing triggers are not sent at the exact same moment to the devices, but with a time difference of 100-200 milliseconds between them. This is due to the fact that devices may perceive the multiple received triggers as one, when arrived simultaneously. So, during the medium-load scenario, there is more idle time between two consecutive sensing intervals compared to the heavy-load one, as shown in Figure 61.
Throughout this longer inactive time, the power-saving mode (PSM) – which is by default enabled on Android devices – is more likely to set the Wi-Fi interface’s state from ‘transfer’ to ‘idle’. When so, additional energy is required to go revert Wi-Fi interface’s state back to transfer mode again. Moreover, the Wi-Fi power needs rise as the time between transfers increases [39]. To summarise the above, the interface’s status transfers together with the longer idle windows result in the higher energy consumption detected during the medium-load sensing scenarios.

The only notable exception to the discussed power trend is RabbitMQ. As seen in Figure 59, the application’s energy consumption remained virtually the same across the three sensing loads. The reason behind this is that RabbitMQ keeps the connection with the broker always on, even when no data are transmitted, as opposed to MQTT and GCM. The impact of this behaviour is twofold: on one hand, during idle and light load scenarios the energy consumption is higher compared to the competition, but on the other hand, during more demanding situations, the consumption linked to the changes of the interface’s state and reconnection overheads is limited. Based on our results, it is evident that the data transmission overhead during medium and heavy load sensing is negligible compared to the energy consumed to keep the communication channel open. At the same time, RabbitMQ’s nature to keep the connection continuously alive proved to be more efficient than the constant state switching of the network’s interface between ‘transfer’ and ‘idle’ mode. Thus, RabbitMQ might be the most demanding choice for n=1 during burst-scenarios, but its connection properties make it a very efficient protocol for n=3 and n=5.

### 6.3.2 Battery life

Throughout this chapter, energy consumption had been presented from the raw-value point-of-view. However, we can interpret the experiment measurements from the battery life angle, as by using it, it is easier to perceive the gravity of the results. Equation 1, results the battery life given battery capacity and load current.

\[
\text{Battery Life} = \frac{\text{Battery Capacity}}{\text{Current}}
\]

Equation 1

Using Equation 2 (Watt’s law), in conjunction with Equation 1 we are able to calculate the maximum run time of a device performing our crowdsensing tasks.

\[
P = V \times I
\]

Equation 2

At this point we need to stress that the voltage dynamically changes as the battery discharges. However, thanks to the fact that the devices used for our experiments were brand new, the voltage
fluctuation was negligible. Hence, we assumed that voltage was equal with \( V = 4.227 \) volts, as indicated by the battery’s manufacturer. Additionally, it is common practise to introduce an additional parameter in Equation 1 which considers external factor that can affect battery life. This section excluded this parameter, as its scope does not include precise modelling battery discharges but, provide rough estimations.

Figure 62 illustrates a summarized overview of the discharge times for the burst-type of sensing. It is evident that by using the appropriate protocol on each occasion, a device can participate in crowdsensing experiments up to 4.5 hours more when sensors used \( n=1 \), 6.12 hours when \( n=3 \), and finally 6.32 additional hours when \( n=5 \).

Similarly, as presented in Figure 63, during the triggered type of sensing the benefits of choosing the right protocol are great. More specifically, a device can last up to 3.77, 3.26, and 5.5 more hours, during the light, medium, and heavy-load scenarios correspondingly.
Table 13 and Table 14 present the maximum continuous sensing time per sensing-load and protocol for the burst and the triggered type of sensing respectively. Green colour indicates the best result for each category, while on the other hand, red expresses the most power demanding protocol. It is interesting to note that in some cases, triggered sensing was more power efficient than its burst-sensing correspondent, such as during light and heavy-load scenarios for RabbitMQ and during heavy-load for MQTT. Additionally, under certain circumstances, energy consumption is equivalent but the difference between message delivery is vast, like the case of medium and heavy-load sensing using GCM.

### 6.4 Impact and conclusion

From the results acquired and analysed in this chapter, it is clear that the performance of each IoT communication protocol fluctuates depending on the parameters of the crowdsensing experiment. Google Cloud Messaging is the best choice for any inactive crowdsensing tool as the energy overhead is significantly smaller than the other protocols. However, for the rest of the examined scenarios no IoT protocol was observed to be standing out from the competition. As described in earlier in this study, apart from the objective factors that characterise a system as efficient, there is an array of KPIs defined by the stakeholders that could affect the selection of a communication protocol.
The work from this chapter has affected directly the WP2 of the FP7 project \textit{IoTLab}. More specifically, the initial architecture of the system defined that a burst type of sensing should happen on the mobile devices. However, after the first-year review of the project, the triggered type of sensing was proposed and adopted by the platform. This choice not only helped to minimise the energy consumption of the developed system – as it limited the alarms set on the devices, but also rendered the mobile crowdsensing more robust overall.
### Results and Discussion

Table 13: Maximum continuous sensing per sensing-load and protocol (burst type)

<table>
<thead>
<tr>
<th>Sensors Used:</th>
<th>GCM</th>
<th>MQTT</th>
<th>REST</th>
<th>RabbitMQ</th>
<th>RabbitMQ + GCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>17.43</td>
<td>20.97</td>
<td>20.13</td>
<td>16.2</td>
<td>20.27</td>
</tr>
<tr>
<td>Three</td>
<td>13.6</td>
<td>17.34</td>
<td>19.72</td>
<td>16.74</td>
<td>17.92</td>
</tr>
<tr>
<td>Five</td>
<td>13.2</td>
<td>15.42</td>
<td>19.52</td>
<td>16.24</td>
<td>15.44</td>
</tr>
</tbody>
</table>

**Burst sensing**

Table 14: Maximum continuous sensing per sensing-load and protocol (triggered type)

<table>
<thead>
<tr>
<th>Sensors Used:</th>
<th>GCM</th>
<th>MQTT</th>
<th>REST</th>
<th>RabbitMQ</th>
<th>RabbitMQ + GCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>One</td>
<td>15.34</td>
<td>19.11</td>
<td>18.95</td>
<td>17.0</td>
<td>16.13</td>
</tr>
<tr>
<td>Three</td>
<td>13.1</td>
<td>14.4</td>
<td>15.18</td>
<td>16.36</td>
<td>14.29</td>
</tr>
<tr>
<td>Five</td>
<td>12.33</td>
<td>17.83</td>
<td>17.07</td>
<td>17.29</td>
<td>15.0</td>
</tr>
</tbody>
</table>

**Triggered sensing**
Chapter 7

Optimal Protocol Selection on MCS Systems

This chapter focuses on the fact that modern mobile crowdsensing systems are not static during their lifetime. Their dynamic nature makes communication protocol selection a challenging task, as there is no panacea. Based on that realisation, the issue of multi-objective optimisation emerges in relation to the everchanging parameters of MCS. This chapter introduces a mechanism that generates weights in two ways based on user preferences. Additionally, the user-preference efficient crowdsensing algorithm is proposed which determines the optimal protocol given a series of parameters. Then, this algorithm is evaluated against a variety of crowdsensing scenarios created by an artificial problems engine. Finally, the findings are analysed and discussed.

7.1 Dynamic crowdsensing experiments

The notion of using sensors for specialised services is not new and goes back to the days of the Cold War. Wireless sensor networks have come a long way, since the United States navy deployed the Sound Surveillance System (SOSUS) in the 1950s to detect enemy submarines. As opposed to the nature of the early stages of sensing experiments, modern mobile crowdsensing scenarios are not static. On the contrary, they are constantly evolving and adapting on-the-go.

The requirements of one experiment might dictate the need of a specific set of data at its start, gathered from the corresponding sources/sensors. Nonetheless, as the experiment progresses, the types of data needed may change and consequently affect the number of sources engaged. For instance, IoT devices that are participating in positioning scenarios might combine data from a GPS sensor, a Wi-Fi interface, a Bluetooth interface, and cellular information in order to infer their location. However, during the course of a scenario, some of these sources of information might cease to exist due to limited signal reception, or unavailability of a service. For that reason, the participating device will not need to transmit the reciprocal data.
As examined in Chapter 2 of this document, experimenters are another reason that modern crowdsensing experiments change the number of shared data. Aiming to minimise energy consumption, they design systems in a way that reduces the total of engaged sensors. Using the previous example, this technique would be applied as follows: once a device infers its location it will automatically start using the embedded gyro and accelerometer instead of all the available positioning sensors. In that way, not only more energy efficient sensors are used but also, the broadcasted data are significantly less.

Since the parameters of a crowdsensing experiment are not set in stone, there is a need to apply the findings of the previous chapters in realistic scenarios. Some of these parameters are set by the experimenters (like base security of the system), some by the system itself (number of sensors engaged), while others by the participants (energy preferences).

### 7.2 Multi-objective optimisation

Making modern mobile crowdsensing experiments efficient is a study topic in operational research, and more specifically of multiple-criteria decision-making (MCDM). Similar with all the nontrivial multi-objective optimisation problems, due to the conflicting factors such as energy consumption, robustness, and security, there is no single solution to optimise all the required objectives. This section will attempt to formulate the selection of crowdsensing communication protocols based on the aforementioned factors. To achieve that, the generated data have to be normalised to reflect the appropriate magnitude when used in inference. The min-max scaling method was selected to map all values in [0,1]. The min-max formula is given as:

\[
x' = \frac{x - \min(x)}{\max(x) - \min(x)}
\]

\[\text{Equation 3}\]

Since all the variables are now normalised between [0,1] the function of protocol selection, as a result of objectives and parameters can be defined below as:

\[
f(p, d, s) = (w_1 \cdot |p| + w_2 \cdot |d| + w_3 \cdot |s|) \cdot l
\]

\[\text{Equation 4}\]

where \(p\) stands for the energy consumption of the system, \(d\) for the message delivery robustness, \(s\) for the security level, and \(w = \{w_1 + w_2 + w_3\}\) is the set of weights assigned to each objective. Furthermore, \(l\) represents the sensing load of the system and based on its value, the outcome of the
function is affected. Sensing load has four different states, as defined in Section 3.2, hence it can be described as \( l = \{\text{idle, light, medium, heavy}\} \).

\[
\sum_{n=1}^{3} w_i = 1
\]

*Equation 5*

As expected, all weights need to satisfy the constraint presented in Equation 5. Additionally, based on the results acquired in Sections 6.1 and 6.2.1, the ranges of parameters \( p \) and \( d \) would be set as follows:

\[
p \in [0.175, 0.72] \quad \text{and} \quad d \in [42.5, 100]
\]

Nonetheless, as described earlier in this section, the values of both the parameters \( p \) and \( d \) are rescaled between [0, 1] using Equation 5. In a similar manner, the security of the protocols has to be quantified and normalised before it is used as a parameter. To achieve this, each protocol will be given a score \( s \in [0, 1] \) based on its objective security features extracted in Chapter 3, such as data encryption, access control, and authentication.

### 7.2.1 Weight assignment

As described in Equation 4, parameter weights influence essentially the choice of the most appropriate communication protocol during a crowdsensing scenario. There are various techniques used to assign values to weights and some of them are:

- **Literature-based inference**: User recruitment and engagement in mobile crowdsensing experiments are aspects that have been investigated extensively by the research community [15], [24], [62], [117], [118]. These, combined with the analysis that has been performed on understanding the factors that influence the attitude of users towards mobile application [7], [119] are of great importance. Apart from deducing the importance of each factor compared to others, it is also possible to map user preferences to weight values. For instance, if 80% of users claim that security is crucial, while only 25% declare that energy consumption is, it is easy to infer that \( w_{\text{security}} > w_{\text{energy}} \).

- **Sorting algorithms**: as mentioned, efficiency in a crowdsensing experiment is a MCDM problem that can have multiple optimal solutions, called Pareto solutions. All the optimal
solutions form the Pareto frontier, which can be computed via existing multi-objective search algorithms [120]–[122].

- **User input:** even though there is a lot of information available in the literature that can assist on assigning weights, one must not forget that users’ desires concerning the usage of their mobile devices fluctuate constantly. For instance, participant Bob usually is keen on participating in crowdsensing experiments but on a particular day he may want to extend the battery life of his device as he intends to travel. Likewise, user Alice might face financial difficulties and she would be willing to lower her usually high level of security settings in exchange for higher monetary incentives. It is evident that users can provide their ever-changing preferences directly to the crowdsensing system and consequently adjust the weighting. User input is a method that has been adopted widely in modern mobile applications to keep user acceptance rates high.

In the context of this study, weight assignment will be based on user input and more specifically we introduce the following two ways to achieve it:

1) **Random assignment strategy (RAS):** This includes the random generation of values for all the weights in the range $w_x \in [0,1]$. Then, based on constraints $C$ that determine the relation between the weights (for instance: $w_x > w_y$), the UPECA ensures that the assigned values satisfy both user requirements and Equation 5. If not, it starts again.

2) **Direct assignment strategy (DAS):** Based on a more detailed user input, preferences are mapped to values that satisfy both user requirements (constraints $C$) and Equation 5. This method requires additional information from the users and its main advantage is that it ensures that preferences are precisely reflected during a crowdsensing experiment.

Both techniques assume the existence of a corresponding UI embedded in the crowdsensing tool which is responsible to extract user requirements/constraints. On the former, user selects which are the most important criteria based on simple questions (i.e. “Want to trade battery lifetime over message delivery/security?”), while on the latter seekbars$^{22}$ are used (i.e. “From 1-10 how important is battery life to you?”)

---

$^{22}$ https://developer.android.com/reference/android/widget/SeekBar
7.2.2 User-preference efficient crowdsensing algorithm

This study presents the user-preference efficient crowdsensing algorithm (UPECA) which calculates the optimal/best communication protocol based on i) parameters defined by users, ii) ground truth, and iii) system specifications. The first step of this algorithm is mapping user preferences (constraints C) into weights using the techniques described in Section 7.2.1. As detailed in the said section, when RAS is selected, the system generates random values to be used as weights that satisfy both the user-defined constrains C but also their summary equals to 1. If DAS is used, then user constraints are directly mapped to parameter weights.

For the second step, based on sensing load l, which is defined by the experimenter, the corresponding energy consumption, delivery percentage, and security level data are gathered. Following that and before used as an input, each value is multiplied with their respective weight as follows:

\[
p' = w_1 \times p \\
d' = w_2 \times d \\
s' = w_3 \times s
\]

Then, UPECA sorts the communication protocols P by the value that is the most important to the user, in other words, the value with the greatest weight. The sorted results will then be subject to further sorting based on the second greatest weight. Finally, a last sorting will happen in relation to the third value.

At this point, the proposed algorithm has calculated the best protocol \( P_{best} \) that will be selected for use. However, in the extreme case that after the aforementioned sorting UPECA returns more than one best protocol (\( P_{best} > 1 \)), then a random protocol P is selected where

\[
P \in [P_{best1}, P_{best2}, ..., P_{bestn}].
\]

Figure 64 provides pseudocode of the steps followed by the UPECA before the start of a crowdsensing experiment in order to calculate the best protocol.
UPECA: User-Preference Efficient Crowdsensing Algorithm

**Input:** \( W = \{ w_1, w_2, w_3 \}, \ C = \{ c_1, c_2, ..., c_n \} \), sensing load \( l \) (Integer), energy consumption \( p \), message delivery \( d \), and level of security \( s \)

**Output:** Protocol \( P_{\text{best}} = \{ \text{GCM}, \text{MQTT}, \text{REST}, \text{RabbitMQ}, \text{RabbitMQ + GCM} \} \)

if RAS then
\[
\begin{align*}
    w_1 &:= 0 \\
    w_2 &:= 0 \\
    w_3 &:= 0 \\
\end{align*}
\]
while \( C \) not satisfied and \( (w_1 + w_2 + w_3 > 1) \)
forall \( w_1, w_2, w_3 \) do
\[
\begin{align*}
    w_1 &:= (0, 1) \\
    w_2 &:= (0, 1) \\
    w_3 &:= (0, 1) \\
\end{align*}
\]
else
\[
W := (0, 1) \text{ that satisfies } C
\]

**Step 1:** Get values \( p, d, \) and \( s \) for sensing load \( l \)

**Step 2:** Apply assigned weights \( w_1, w_2, w_3 \) to corresponding values \( p, d, \) and \( s \)

**Step 3:** Sort \( P \) by the value that corresponds to the greatest weight
Sort the result by the value that corresponds to the middle weight.
Sort the result by the value that corresponds to the lowest weight

if \( P_{\text{optimal}} > 1 \) then
assign \( P \) randomly, where \( P \in [P_{\text{optimal1}}, P_{\text{optimal2}}, ..., P_{\text{optimaln}}] \)

---

Figure 64: Pseudocode for UPECA

7.2.3 Adapting user-preference efficient crowdsensing algorithm

As discussed earlier in this chapter, since the parameters of modern crowdsensing experiments are dynamic so the performance of the protocols-in-use is. Previous sections of this study have proven that IoT communication protocols could be more efficient that others depending on the nature of the experiment. Hence, we believe that the user-preference efficient crowdsensing algorithm should be extended when used in dynamic environments. Figure 65 illustrates the pseudocode for the extended version of the UPECA - the Adapting UPECA. The difference between the two versions include the way this algorithm should be used and an additional process during Step 3.

The adapting user-preference efficient crowdsensing algorithm should be executed not only before the start of a dynamic crowdsensing experiment, but also every time parameters change. This will ensure that user preferences are continuously followed throughout an experiment’s lifecycle. Additionally, when Step 3 calculates the best communication protocol and \( P > 1 \), then the selection is done as follows: in case this is the start of the crowdsensing experiment, then the algorithm selects
randomly among the available $P_{best}$s options. On the other hand, if the experiment is ongoing, then UPECA checks if the protocol that is already in-use is part of the $P_{best}$s solutions. In case it is ($P_{in\_use} \in P_{best}$s), then the system will continue using it. If not, the new protocol will be the result of random selection.

**Adapting UPECA: User-Preference Efficient Crowdsensing Algorithm**

**Input:** $W = \{w_1, w_2, w_3\}$, $C = \{c_1, c_2, ..., c_n\}$, sensing load $l$ (Integer), energy consumption $p$, message delivery $d$, and level of security $s$

**Output:** Protocol $P_{best} = \{GCM, MQTT, REST, RabbitMQ, RabbitMQ + GCM\}$

if RAS then

$w_1 := 0$

$w_2 := 0$

$w_3 := 0$

while $C$ not satisfied and $(w_1 + w_2 + w_3 > l)$

forall $w_1$, $w_2$, $w_3$ do

$w_1 := (0,1)$

$w_2 := (0,1)$

$w_3 := (0,1)$

else

$W := (0,1)$ that satisfies $C$

**Step 1:** Get values $p$, $d$, and $s$ for sensing load $l$

**Step 2:** Apply assigned weights $w_1$, $w_2$, $w_3$ to corresponding values $p$, $d$, and $s$

**Step 3:** Sort $P$ by the value that corresponds to the greatest weight

Sort the result by the value that corresponds to the middle weight.

Sort the result by the value that corresponds to the lowest weight

if $P_{best} > 1$ then

if experiment is not ongoing then

assign $P$ randomly

else if experiment is ongoing then

if protocol already used $\in P_{best}$s then do not assign new $P$

else assign new $P$ randomly

Figure 65: Pseudocode for the adapting UPECA
7.3 Artificial problems

In order to evaluate the impact that UPECA has on crowdsensing experiments, a set of artificial problems need to be generated. To provide fair comparison, the artificial problems share the same characteristics with the scenarios examined in Chapter 3 but with a significant difference: sensing load is not static throughout their run. The rest of this sub-section discusses in-detail the generated artificial problems.

7.3.1 Artificial problems composition

As described at the beginning of this chapter, modern crowdsensing scenarios are dynamic and change their sensing requirements throughout their lifecycle. This section presents an engine which is developed in Java and generates details about artificial problems given the length of a crowdsensing experiment. As illustrated in Figure 66, the artificial problems engine (APE) receives as input the duration $d$ of a crowdsensing experiment and the constraint $c$, which essentially defines the minimum timeframe during which the system parameters remain unchanged. We assumed that a crowdsensing system will not change its parameters for at least $c = 30$ seconds. The output of the engine is a list of sensing loads $L$ that correspond to a list of time intervals $I$. In other words, the APE is breaking down the total duration $d$ of an experiment, in smaller intervals $i$, while assigning sensing load $l_i$ to them.

**APE: Artificial problems engine**

**Input:** Experiment duration $d \in \{5, 10, 20\}$, interval constraint $c = 30$ sec

**Output:** Intervals $I = \{i_1, i_2, \ldots, i_n\}$, sensing loads $L = \{l_1, l, \ldots, l_n\}$

while $d > 0$ do

**Step 1:** generate $t_i \in [c, d]$

**Step 2:** generate $l_i \in \{0, 1, 3, 5\}$

**Step 3:** $d := d - t_i$

if $d < c$ then

generate $t_i \in [0, d]$

generate $l_i \in \{0, 1, 3, 5\}$

break

Figure 66: Pseudocode for the Artificial problems engine

The artificial problems engine was used to generate the parameters of fifteen crowdsensing scenarios, five for each of the possible experiment durations (5, 10, and 20 minutes) presented in
Chapter 3. The results are illustrated in Table 15. Note that for idle, light, medium, and heavy sensing the numbers 0, 1, 3, and 5 are used correspondingly.

Table 15: Output of the Artificial Problems Engine for d= \{5, 10, 20\}

<table>
<thead>
<tr>
<th>No</th>
<th>Details of generated artificial problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Experiment duration: 5 minutes.</td>
</tr>
<tr>
<td></td>
<td>Experiment seconds remaining: 300</td>
</tr>
<tr>
<td></td>
<td>Interval 1 uses 1 sensors for 149 seconds.</td>
</tr>
<tr>
<td></td>
<td>Experiment seconds remaining: 151</td>
</tr>
<tr>
<td></td>
<td>Interval 2 uses 0 sensors for 100 seconds.</td>
</tr>
<tr>
<td></td>
<td>Experiment seconds remaining: 51</td>
</tr>
<tr>
<td></td>
<td>Interval 3 uses 5 sensors for 48 seconds.</td>
</tr>
<tr>
<td></td>
<td>Experiment seconds remaining: 3</td>
</tr>
<tr>
<td></td>
<td>Interval 4 uses 3 sensors for 3 seconds.</td>
</tr>
</tbody>
</table>

| 2  | Experiment duration: 5 minutes.          |
|    | Experiment seconds remaining: 300        |
|    | Interval 1 uses 0 sensors for 263 seconds.|
|    | Experiment seconds remaining: 37         |
|    | Interval 2 uses 1 sensors for 35 seconds. |
|    | Experiment seconds remaining: 2          |
|    | Interval 3 uses 0 sensors for 2 seconds.  |

| 3  | Experiment duration: 5 minutes.          |
|    | Experiment seconds remaining: 300        |
|    | Interval 1 uses 5 sensors for 253 seconds.|
|    | Experiment seconds remaining: 47         |
|    | Interval 2 uses 1 sensors for 34 seconds. |
|    | Experiment seconds remaining: 13         |
|    | Interval 3 uses 1 sensors for 13 seconds. |

| 4  | Experiment duration: 5 minutes.          |
|    | Experiment seconds remaining: 300        |
|    | Interval 1 uses 5 sensors for 85 seconds. |
|    | Experiment seconds remaining: 215        |
|    | Interval 2 uses 1 sensors for 175 seconds.|
|    | Experiment seconds remaining: 40         |
|    | Interval 3 uses 0 sensors for 36 seconds. |
|    | Experiment seconds remaining: 4           |
|    | Interval 4 uses 3 sensors for 4 seconds.  |

| 5  | Experiment duration: 5 minutes.          |
|    | Experiment seconds remaining: 300        |
|    | Interval 1 uses 1 sensors for 272 seconds.|
|    | Experiment seconds remaining: 28         |
|    | Interval 2 uses 5 sensors for 28 seconds. |

| 6  | Experiment duration: 10 minutes.         |
|    | Experiment seconds remaining: 600        |
|    | Interval 1 uses 1 sensors for 130 seconds.|
|    | Experiment seconds remaining: 470        |
|    | Interval 2 uses 5 sensors for 279 seconds.|
|    | Experiment seconds remaining: 191        |
|    | Interval 3 uses 3 sensors for 95 seconds. |
|    | Experiment seconds remaining: 96         |
|    | Interval 4 uses 5 sensors for 78 seconds. |
|    | Experiment seconds remaining: 18         |
|    | Interval 5 uses 0 sensors for 18 seconds. |

| 7  | Experiment duration: 10 minutes.         |
|    | Experiment seconds remaining: 600        |
|    | Interval 1 uses 1 sensors for 436 seconds.|
|    | Experiment seconds remaining: 436 seconds.|

<table>
<thead>
<tr>
<th>Experiment duration: 10 minutes.</th>
<th>Experiment duration: 10 minutes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment seconds remaining: 600</td>
<td>Experiment seconds remaining: 600</td>
</tr>
<tr>
<td>Interval 1 uses 1 sensors for 436 seconds.</td>
<td>Interval 1 uses 1 sensors for 54 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 164</td>
<td>Experiment seconds remaining: 546</td>
</tr>
<tr>
<td>Interval 2 uses 0 sensors for 82 seconds.</td>
<td>Interval 2 uses 3 sensors for 246 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 82</td>
<td>Experiment seconds remaining: 300</td>
</tr>
<tr>
<td>Interval 3 uses 3 sensors for 63 seconds.</td>
<td>Interval 3 uses 3 sensors for 37 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 19</td>
<td>Experiment seconds remaining: 263</td>
</tr>
<tr>
<td>Interval 4 uses 3 sensors for 19 seconds.</td>
<td>Interval 4 uses 0 sensors for 102 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 15</td>
<td>Experiment seconds remaining: 161</td>
</tr>
<tr>
<td>Interval 5 uses 5 sensors for 95 seconds.</td>
<td>Interval 5 uses 5 sensors for 95 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 66</td>
<td>Experiment seconds remaining: 66</td>
</tr>
<tr>
<td>Interval 6 uses 0 sensors for 51 seconds.</td>
<td>Interval 6 uses 0 sensors for 51 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 15</td>
<td>Experiment seconds remaining: 15</td>
</tr>
<tr>
<td>Interval 7 uses 3 sensors for 15 seconds.</td>
<td>Interval 7 uses 3 sensors for 15 seconds.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment duration: 10 minutes.</th>
<th>Experiment duration: 10 minutes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment seconds remaining: 600</td>
<td>Experiment seconds remaining: 600</td>
</tr>
<tr>
<td>Interval 1 uses 1 sensors for 73 seconds.</td>
<td>Interval 1 uses 1 sensors for 73 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 527</td>
<td>Experiment seconds remaining: 527</td>
</tr>
<tr>
<td>Interval 2 uses 5 sensors for 145 seconds.</td>
<td>Interval 2 uses 5 sensors for 145 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 382</td>
<td>Experiment seconds remaining: 382</td>
</tr>
<tr>
<td>Interval 3 uses 1 sensors for 226 seconds.</td>
<td>Interval 3 uses 1 sensors for 226 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 156</td>
<td>Experiment seconds remaining: 156</td>
</tr>
<tr>
<td>Interval 4 uses 5 sensors for 123 seconds.</td>
<td>Interval 4 uses 5 sensors for 123 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 33</td>
<td>Experiment seconds remaining: 33</td>
</tr>
<tr>
<td>Interval 5 uses 3 sensors for 30 seconds.</td>
<td>Interval 5 uses 3 sensors for 30 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 3</td>
<td>Experiment seconds remaining: 3</td>
</tr>
<tr>
<td>Interval 6 uses 0 sensors for 3 seconds.</td>
<td>Interval 6 uses 0 sensors for 3 seconds.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment duration: 20 minutes.</th>
<th>Experiment duration: 20 minutes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment seconds remaining: 1200</td>
<td>Experiment seconds remaining: 1200</td>
</tr>
<tr>
<td>Interval 1 uses 1 sensors for 802 seconds.</td>
<td>Interval 1 uses 1 sensors for 802 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 398</td>
<td>Experiment seconds remaining: 398</td>
</tr>
<tr>
<td>Interval 2 uses 0 sensors for 239 seconds.</td>
<td>Interval 2 uses 0 sensors for 239 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 159</td>
<td>Experiment seconds remaining: 159</td>
</tr>
<tr>
<td>Interval 3 uses 1 sensors for 113 seconds.</td>
<td>Interval 3 uses 1 sensors for 113 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 46</td>
<td>Experiment seconds remaining: 46</td>
</tr>
<tr>
<td>Interval 4 uses 5 sensors for 45 seconds.</td>
<td>Interval 4 uses 5 sensors for 45 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 1</td>
<td>Experiment seconds remaining: 1</td>
</tr>
<tr>
<td>Interval 5 uses 1 sensors for 1 seconds.</td>
<td>Interval 5 uses 1 sensors for 1 seconds.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment duration: 20 minutes.</th>
<th>Experiment duration: 20 minutes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment seconds remaining: 1200</td>
<td>Experiment seconds remaining: 1200</td>
</tr>
<tr>
<td>Interval 1 uses 1 sensors for 570 seconds.</td>
<td>Interval 1 uses 1 sensors for 570 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 630</td>
<td>Experiment seconds remaining: 630</td>
</tr>
<tr>
<td>Interval 2 uses 1 sensors for 150 seconds.</td>
<td>Interval 2 uses 1 sensors for 150 seconds.</td>
</tr>
<tr>
<td>Experiment seconds remaining: 480</td>
<td>Experiment seconds remaining: 480</td>
</tr>
<tr>
<td>Interval 3 uses 0 sensors for 234 seconds.</td>
<td>Interval 3 uses 0 sensors for 234 seconds.</td>
</tr>
</tbody>
</table>
Experiment seconds remaining: 246
Interval 4 uses 1 sensors for 171 seconds.
Experiment seconds remaining: 75
Interval 5 uses 5 sensors for 42 seconds.
Experiment seconds remaining: 33
Interval 6 uses 1 sensors for 31 seconds.
Experiment seconds remaining: 2
Interval 7 uses 0 sensors for 2 seconds.

Experiment duration: 20 minutes.
Experiment seconds remaining: 1200
Interval 1 uses 3 sensors for 423 seconds.
Experiment seconds remaining: 777
Interval 2 uses 0 sensors for 50 seconds.
Experiment seconds remaining: 727
Interval 3 uses 0 sensors for 268 seconds.
Experiment seconds remaining: 459
Interval 4 uses 0 sensors for 179 seconds.
Interval 5 uses 1 sensors for 48 seconds.
Interval 6 uses 3 sensors for 232 seconds.
Interval 7 uses 0 sensors for 67 seconds.
Interval 8 uses 5 sensors for 70 seconds.
Interval 9 uses 5 sensors for 52 seconds.

Experiment duration: 20 minutes.
Experiment seconds remaining: 1200
Interval 1 uses 0 sensors for 688 seconds.
Experiment seconds remaining: 512
Interval 2 uses 5 sensors for 179 seconds.
Interval 3 uses 3 sensors for 321 seconds.
Interval 4 uses 0 sensors for 12 seconds.

Experiment duration: 20 minutes.
Experiment seconds remaining: 1200
Interval 1 uses 0 sensors for 394 seconds.
Experiment seconds remaining: 806
Interval 2 uses 0 sensors for 61 seconds.
Interval 3 uses 3 sensors for 234 seconds.
Interval 4 uses 5 sensors for 49 seconds.
Interval 5 uses 5 sensors for 89 seconds.
Interval 6 uses 1 sensors for 69 seconds.
Interval 7 uses 5 sensors for 75 seconds.
Interval 8 uses 3 sensors for 31 seconds.
Interval 9 uses 5 sensors for 34 seconds.
Interval 10 uses 5 sensors for 35 seconds.
Interval 11 uses 1 sensors for 40 seconds.
Interval 12 uses 3 sensors for 89 seconds.
The above table contains an assortment of fifteen artificial crowdsensing scenarios that are used to examine the benefits of UPECA in Section 7.3.2. The large number of scenarios together with the wide spectrum of intervals and sensing loads which were randomly generated from the APE ensures the fair assessment and the validity of the results. More specifically, the APE provided us information on experiments with ranging intervals that use all kinds of sensing loads. In that manner, we can assess UPECA against life-like crowdsensing scenarios. Additionally, this diversity of parameters will highlight the potential advantage of the adapting UPECA in long and complex scenarios.

Random weight assignment
As Section 7.2.2 defines, amongst the required inputs of the UPECA are user preferences concerning crowdsensing experiments. The remaining of this subsection will generate arrays of preferences that will be exploited in order to extract and assign weights. Both the random and the direct assignment strategy will be used. Later on, these weights will be mapped to the generated artificial problems presented earlier.

Table 16: Generated weights mapped to the artificial problems

<table>
<thead>
<tr>
<th>No</th>
<th>Weights break down</th>
<th>No</th>
<th>Weights break down</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>w₁=0.42, w₂=0.10, w₃=0.48</td>
<td>9</td>
<td>w₁=0.4, w₂=0.3, w₃=0.3</td>
</tr>
<tr>
<td>2</td>
<td>w₁=0.48, w₂=0.20, w₃=0.32</td>
<td>10</td>
<td>w₁=0.3, w₂=0.6, w₃=0.1</td>
</tr>
<tr>
<td>3</td>
<td>w₁=0.22, w₂=0.17, w₃=0.61</td>
<td>11</td>
<td>w₁=0.8, w₂=0.2, w₃=0.0</td>
</tr>
<tr>
<td>4</td>
<td>w₁=0.46, w₂=0.49, w₃=0.05</td>
<td>12</td>
<td>w₁=0.3, w₂=0.5, w₃=0.2</td>
</tr>
<tr>
<td>5</td>
<td>w₁=0.41, w₂=0.10, w₃=0.49</td>
<td>13</td>
<td>w₁=0.3, w₂=0.0, w₃=0.7</td>
</tr>
<tr>
<td>6</td>
<td>w₁=0.45, w₂=0.24, w₃=0.31</td>
<td>14</td>
<td>w₁=0.1, w₂=0.9, w₃=0.0</td>
</tr>
<tr>
<td>7</td>
<td>w₁=0.16, w₂=0.70, w₃=0.14</td>
<td>15</td>
<td>w₁=0.7, w₂=0.1, w₃=0.2</td>
</tr>
<tr>
<td>8</td>
<td>w₁=0.47, w₂=0.47, w₃=0.06</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 16 presents the fifteen arrays of weights that were assigned to the corresponding artificial problems. The first eight arrays were generated using the RAS, while the final seven using the DAS. Together with the details summarised in Table 15, they will constitute the input of the proposed user-preference efficient crowdsensing algorithm which will infer the most suitable communication protocol per scenario.

7.3.2 Evaluation of UPECA

This section will go through the fifteen crowdsensing scenarios and apply the UPECA in two different ways, the simple and the adaptive way. The former way will assess the parameters only at the beginning of the crowdsensing experiment and propose the most suitable protocol that will be used throughout the scenario. The latter, will evaluate the updated parameters at intervals I, and
adapt the communication protocol correspondingly. The results will be compared based on energy consumption and message delivery percentage. The aim of this comparison is to not only prove the efficiency of the UPECA but also, the benefits of adapting the communication protocols during a dynamic crowdsensing experiment. Moreover, to illustrate the potential impact that user preferences have on an experiment, UPECA will also run using a different set of constraints C. For ease of reading, only six scenarios will be presented and analysed, two for each experiment duration. The results of the remaining nine, can be found in the Appendix.

7.3.2.1 Duration d = 5 minutes

**Scenario 1:** The first scenario defined security as the most important aspect ($w_3 = 0.48$), followed by energy consumption ($w_2 = 0.42$) and finally, by message delivery percentage ($w_2 = 0.10$). UPECA selected the burst RabbitMQ approach to handle the start of the transmissions as it offers high level of security and relatively low power demands. As the load parameter changed, so did the recommended protocol by the proposed algorithm. This resulted in a marginally better performance in terms of message delivery but to a significant superiority in energy efficiency.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td>light</td>
<td>Burst RabbitMQ</td>
<td>0.548</td>
<td>27</td>
<td>Burst RabbitMQ</td>
<td>0.548</td>
<td>27</td>
</tr>
<tr>
<td>100</td>
<td>idle</td>
<td></td>
<td>0.212</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.546</td>
<td>44</td>
<td>Triggered RabbitMQ</td>
<td>0.513</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>medium</td>
<td>Burst RabbitMQ</td>
<td>0.53</td>
<td>2</td>
<td>Burst RabbitMQ</td>
<td>0.53</td>
<td>2</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td><strong>0.43</strong></td>
<td>73</td>
<td>Total:</td>
<td><strong>0.35</strong></td>
<td>74</td>
</tr>
</tbody>
</table>

To further investigate the impact of user preferences on the UPECA, the experiment was run using the weights generated for scenario 4, where $W = (w_1 = 0.46, w_2 = 0.49, w_3 = 0.05)$. As presented in the table below, in both cases (both simple and adapting versions), the algorithm managed to deliver more messages, while having lower energy demands.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>149</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.53</td>
<td>28</td>
<td>Triggered MQTT</td>
<td>0.53</td>
<td>28</td>
</tr>
<tr>
<td>100</td>
<td>idle</td>
<td></td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.59</td>
<td>44</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>medium</td>
<td></td>
<td>0.8</td>
<td>3</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>3</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td><strong>0.42</strong></td>
<td>75</td>
<td>Total:</td>
<td><strong>0.34</strong></td>
<td>76</td>
</tr>
</tbody>
</table>
Scenario 4: The second scenario inspected set the message delivery as top priority \((w_2 = 0.49)\) and then energy consumption \((w_1 = 0.46)\). The weight assigned to system security was so small \((w_3 = 0.05)\) that the effect it had on UPECA was rather insignificant. RabbitMQ was once again used extensively in both runs, however, the adapting version incorporated also the MQTT and the GCM to alleviate power demands, while keeping robustness at high levels. As indicated by the results, even though between the two approaches no significant difference was noted in the message delivery part, the adapting approach managed to lower energy consumption by 12.5%.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Protocol</td>
<td>Mean Energy</td>
<td>Messages Delivered</td>
</tr>
<tr>
<td>85</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
</tr>
<tr>
<td>175</td>
<td>light</td>
<td>Triggered RabbitMQ</td>
<td>0.52</td>
</tr>
<tr>
<td>36</td>
<td>idle</td>
<td>Triggered RabbitMQ</td>
<td>0.21</td>
</tr>
<tr>
<td>4</td>
<td>medium</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>0.48</td>
<td>123</td>
</tr>
</tbody>
</table>

Again, the algorithm was assessed against another set of weights, this time using scenario 1’s. The following Table summarises the outcome of UPECA when \(W = (w_1 = 0.42, w_2 = 0.10, w_3 = 0.48)\). One can notice that during the adapting use of the algorithm the energy demands were slightly lower compared to the default constraints. On the other hand, it delivered 2.4% less messages, as message reliability was not a high priority. The impact of user preferences is visible, but not clear during short crowdsensing experiments as the two presented in this section. It’s in the author’s beliefs that the difference will become more evident as the scenarios become more complex.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Protocol</td>
<td>Mean Energy</td>
<td>Messages Delivered</td>
</tr>
<tr>
<td>85</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
</tr>
<tr>
<td>175</td>
<td>light</td>
<td>Triggered RabbitMQ+GCM</td>
<td>0.43</td>
</tr>
<tr>
<td>36</td>
<td>idle</td>
<td>Triggered RabbitMQ+GCM</td>
<td>0.2</td>
</tr>
<tr>
<td>4</td>
<td>medium</td>
<td>Triggered RabbitMQ+GCM</td>
<td>0.54</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>0.48</td>
<td>123</td>
</tr>
</tbody>
</table>
7.3.2.2 Duration $d = 10$ minutes

Scenario 6: This scenario illustrates vividly the importance of adapting communication protocols during crowdsensing experiments. User preferences were defined as: $(w_1 = 0.45, w_2 = 0.24, w_3 = 0.31)$ As the sensing load fluctuated from light all the way to heavy and back to idle, the burst MQTT approach performed inadequately. On the other hand, by switching between MQTT, REST, and GCM the system managed to transmit successfully 35% more data, while consuming 13% less energy.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>130</td>
<td>light</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>25</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>25</td>
</tr>
<tr>
<td>279</td>
<td>heavy</td>
<td>Burst REST</td>
<td>0.57</td>
<td>184</td>
<td>Burst REST</td>
<td>0.45</td>
<td>267</td>
</tr>
<tr>
<td>95</td>
<td>medium</td>
<td>Triggered MQTT</td>
<td>0.51</td>
<td>50</td>
<td>Triggered MQTT</td>
<td>0.45</td>
<td>53</td>
</tr>
<tr>
<td>78</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.57</td>
<td>50</td>
<td>Triggered RabbitMQ</td>
<td>0.45</td>
<td>73</td>
</tr>
<tr>
<td>18</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td>0.51</td>
<td>309</td>
<td>Total:</td>
<td>0.44</td>
<td>418</td>
</tr>
</tbody>
</table>

This time, in order to cross examine UPECA the user preferences of scenario 7 were used, where $W = (w_1 = 0.16, w_2 = 0.7, w_3 = 0.14)$. As opposed to the initial values, message delivery is the highest priority of the experiment, followed by energy consumption and security. As illustrated by the Table below, during the simple run of the algorithm, the difference is hefty. The system delivered 41.4% more messages, while consuming 3.9% less energy. In the same manner, when the adapting version was deployed an increase on message delivery was noticed equal with 5.26%.
**Scenario 9:** This scenario was characterised by a high number of load adjustments. The primary target was keeping energy demands low ($w_1 = 0.4$), while keeping a balance between security ($w_2 = 0.3$) and robustness ($w_3 = 0.3$). Hence, the protocols were used in burst mode, sacrificing the delivery of a number of sensing data. The outcome once more was in favour of the adapting UPECA, as it was 23% less power consuming and 17% more reliable.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>light</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>9</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>9</td>
</tr>
<tr>
<td>246</td>
<td>medium</td>
<td>Burst MQTT</td>
<td>0.51</td>
<td>130</td>
<td>Burst REST</td>
<td>0.45</td>
<td>138</td>
</tr>
<tr>
<td>37</td>
<td>medium</td>
<td>Burst MQTT</td>
<td>0.51</td>
<td>18</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>102</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>95</td>
<td>heavy</td>
<td>Burst REST</td>
<td>0.57</td>
<td>63</td>
<td>Burst REST</td>
<td>0.45</td>
<td>92</td>
</tr>
<tr>
<td>51</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>medium</td>
<td>Burst REST</td>
<td>0.51</td>
<td>7</td>
<td>Burst REST</td>
<td>0.45</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>0.43</strong></td>
<td><strong>227</strong></td>
<td><strong>Total:</strong></td>
<td><strong>0.33</strong></td>
<td><strong>266</strong></td>
</tr>
</tbody>
</table>

The following table is presenting the outcome of UPECA when user preferences of scenario 7 where used ($w_1 = 0.16$, $w_2 = 0.7$, $w_3 = 0.14$). As expected, the heavy weight assigned to message delivery is reflected to the results. The system transmitted successfully 23.3% and 6.4% more messages during the simple and the adapting version of the algorithm correspondingly. Nonetheless, the energy consumption was higher but it was the least important to the end user.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>10</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>10</td>
</tr>
<tr>
<td>246</td>
<td>medium</td>
<td>Triggered MQTT</td>
<td>0.61</td>
<td>146</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>147</td>
</tr>
<tr>
<td>37</td>
<td>medium</td>
<td>Triggered MQTT</td>
<td>0.61</td>
<td>22</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>22</td>
</tr>
<tr>
<td>102</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>95</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.49</td>
<td>94</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>95</td>
</tr>
<tr>
<td>51</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>medium</td>
<td>Triggered RabbitMQ</td>
<td>0.61</td>
<td>8</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>0.47</strong></td>
<td><strong>280</strong></td>
<td><strong>Total:</strong></td>
<td><strong>0.39</strong></td>
<td><strong>283</strong></td>
</tr>
</tbody>
</table>
7.3.2.3 Duration $d = 20$ minutes

Scenario 11: The first scenario examined for $d = 20$, was concerned predominantly about the energy consumption ($w_1 = 0.8$) and less about message reliability ($w_2 = 0.2$). The majority of the problem included light sensing, hence the most suitable communication protocol was the MQTT used in burst mode. However, a short portion involved heavy sensing. Adapting to the later-introduced load status, the second run was 13% more power efficient and transmitted successfully 6.2% more sensor measurements.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean Energy</td>
<td>Mean Energy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Messages Delivered</td>
<td>Messages Delivered</td>
</tr>
<tr>
<td>802</td>
<td>light</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>156</td>
<td>156</td>
</tr>
<tr>
<td>239</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>113</td>
<td>light</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>45</td>
<td>heavy</td>
<td>Burst REST</td>
<td>0.57</td>
<td>0.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>43</td>
</tr>
<tr>
<td>1</td>
<td>light</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>0.38</strong></td>
<td><strong>0.33</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>207</strong></td>
<td><strong>220</strong></td>
</tr>
</tbody>
</table>

This time the weights assigned for scenario 8 ($w_1 = 0.47$, $w_2 = 0.47$, $w_3 = 0.06$) were used to re-assess the UPECA. We wanted to examine the impact that a less focused weight set would have to the outcome. During the simple scenario, an increase of 5.2% of energy consumption resulted to 9.2% more messages delivered. However, during the adapting run, a 12% increase in energy demands led to 3.2% more successful transmissions.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean Energy</td>
<td>Mean Energy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Messages Delivered</td>
<td>Messages Delivered</td>
</tr>
<tr>
<td>802</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>160</td>
<td>160</td>
</tr>
<tr>
<td>239</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>113</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>45</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.49</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>44</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>0.40</strong></td>
<td><strong>0.37</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>226</strong></td>
<td><strong>227</strong></td>
</tr>
</tbody>
</table>

Scenario 15: The last examined scenario ($w_1 = 0.7$, $w_2 = 0.1$, $w_3 = 0.2$) presents a variety in sensing loads and put into test the UPECA. Since its first part kept sensing transmission to idle, the most appropriate choice was GCM. However, as indicated in Chapter 6, the use of GCM should be
avoided during demanding runs. Its inability to handle the imposed burden was reflected by the results, as it delivered only 414 measurements, as opposed to the adapting approach which was 15\% more efficient. Nonetheless, the most notable difference was concerning the energy demands, as the adapting-protocol approach required 61\% less energy than the simple one.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>394</td>
<td>idle</td>
<td>0</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>idle</td>
<td>0</td>
<td>-</td>
<td>Burst REST</td>
<td>0.45</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>medium</td>
<td>0.67</td>
<td>117</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>heavy</td>
<td>0.72</td>
<td>36</td>
<td>Burst REST</td>
<td>0.45</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>heavy</td>
<td>0.72</td>
<td>69</td>
<td>Burst MQTT</td>
<td>0.45</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>light</td>
<td>0.57</td>
<td>13</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>heavy</td>
<td>0.72</td>
<td>61</td>
<td>Burst REST</td>
<td>0.45</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>medium</td>
<td>0.67</td>
<td>15</td>
<td>Burst MQTT</td>
<td>0.45</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>heavy</td>
<td>0.72</td>
<td>24</td>
<td>Burst MQTT</td>
<td>0.45</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>heavy</td>
<td>0.72</td>
<td>28</td>
<td>Burst MQTT</td>
<td>0.45</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>light</td>
<td>0.57</td>
<td>8</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>medium</td>
<td>0.67</td>
<td>43</td>
<td>Burst MQTT</td>
<td>0.45</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>0.44</td>
<td>414</td>
<td>Total: 0.276</td>
<td>477</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the last scenario, the impact of UPECA was re-assessed using the weights assigned for scenario 8 \((w_1 = 0.47, w_2 = 0.47, w_3 = 0.06)\). On the first part of the re-run, the results were identical with the initial test, as the protocol selected by the algorithm was the same (Triggered GCM). On the second part however, a complete different set of protocols was selected throughout the testing. This resulted to 4.8\% more messages delivered, but also to 46\% increase of energy consumption.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Energy</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>394</td>
<td>idle</td>
<td>0</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>idle</td>
<td>0</td>
<td>-</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>234</td>
<td>medium</td>
<td>0.67</td>
<td>117</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>heavy</td>
<td>0.72</td>
<td>36</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>heavy</td>
<td>0.72</td>
<td>69</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>light</td>
<td>0.57</td>
<td>13</td>
<td>Triggered RabbitMQ</td>
<td>0.46</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>heavy</td>
<td>0.72</td>
<td>61</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>medium</td>
<td>0.67</td>
<td>15</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>heavy</td>
<td>0.72</td>
<td>24</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>heavy</td>
<td>0.72</td>
<td>28</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>light</td>
<td>0.57</td>
<td>8</td>
<td>Triggered RabbitMQ</td>
<td>0.46</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>medium</td>
<td>0.67</td>
<td>43</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td>0.44</td>
<td>414</td>
<td>Total: 0.41</td>
<td>500</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. Optimal Protocol Selection on MCS Systems

7.3.3 Results and Discussion

The previous section evaluated the proposed user-preference efficient crowdsensing algorithm against six different crowdsensing scenarios. For each scenario, the algorithm was run either in a simple (static) or an adapting (dynamic) mode. On top of that, for every scenario we used two different sets of weights. That was done in order to demonstrate the impact of user-preferences on the outcome of an experiment in terms of average energy consumption and number of delivered messages. During all twelve cases, the adapting version of the algorithm was superior to the simple one and was reflecting even further user preferences. In some cases, the difference between them was not that considerable such as in Scenario 1 presented in Section 7.3.2.1, while in others, like Scenario 15 of Section 7.3.2.3, the contrast of efficiency is clear.

Based on our findings, the outcome of the UPECA is significantly affected when a parameter is heavily prioritised against the others (e.g. \( w_x = 0.8, w_y = 0.1, w_z = 0.1 \)). However, when the user preference weights are more equally distributed across the parameters (e.g. \( w_x = 0.4, w_y = 0.3, w_z = 0.3 \)), the differences are harder to notice. Additionally, time is also an important factor on how the system is behaving overall. More specifically, the asymmetry of message delivery and energy consumption is more likely to be evident when the system switches between protocols more times, which is usually the case during longer scenarios. The author believes that overall, heavily prioritised weights, which are more linked with direct assignment from end-users, should be avoided, as the equally distributed ones are resulting in more well-balanced behaviours. Moreover, no drawbacks were noticed during the IoT protocol switch during the course of the experiment. However, the need to implement numerous protocol clients inside the mobile application installer results to bigger space requirements on the smart devices.

7.4 Conclusion

The previous sections examined the effect that the proposed user-preference efficient crowdsensing algorithm has on dynamic mobile crowdsensing experiments. UPECA was used to find the optimal/best IoT communication protocol in different sensing scenarios. Based on the input received, it was able to calculate the Pareto frontier in an automatic and robust way.

Additionally, this chapter addressed the potential benefits offered by adapting to the dynamic nature of modern MCS experiments. This was achieved by comparing the conventional way that MCS experiments are conducted to a proposed one. More specifically, the former exploits a single IoT communication protocol throughout a scenario. The latter suggests that any time a parameter of the
system is modified, the UPECA has to infer a new Pareto optimal/best protocol given the updated facts of the experiment.

The \textit{adapting user-preference efficient crowdsensing algorithm} was applied to fifteen generated sensing scenarios. Adapting IoT protocols was consistently the better approach without a doubt. The more changes were noted during a run, the more advantage the adapting UPECA had. Thus, the effects are further noticeable as the duration $d$ of an experiment grows. On the other hand, when no numerous fluctuations take place, the performance of both approaches tend to converge.

Concluding, the proposed algorithm should always be used during crowdsensing scenarios, as its potential impact on the efficiency of a system is indisputable. Additionally, thanks to its design it is easy to be extended with more than three parameters and used in a broad range of scenarios.
Chapter 8

Conclusion

8.1 Closing remarks

This study investigates the notion of mobile crowdsensing which has been gaining a lot of focus in the past decade thanks to the raising percentage of people using smart devices on a daily basis and the penetration of broadband Internet. It presents novel ways of interaction between the crowd and the plethora of available devices and sensors, proposed by the research community. By exploiting the power of the crowd, daily routine tasks have been optimised, making everyone’s lives easier and better. However, like any other discipline, mobile crowdsensing is facing a number of challenges, both technical and practical.

This research clarifies the reasons behind the energy, data quality, and privacy issues of contemporary systems, as well as, a series of attempts and solutions introduced to tackle them. By doing so, a research gap in the transmission aspect of MCS was identified and more specifically, in the exploitation of IoT communication protocols. Based on the conducted literature review, despite the plethora of available IoT communication protocols, only very few of them are actually used. In the opinion of the author, adopting the appropriate protocol based on the parameters of a crowdsensing experiment can be remarkably beneficial and renders the whole system more efficient. However, efficiency is an elastic term that could have a different meaning depending on the viewpoint of the evaluator. Hence, a series of MCS-oriented key performance indicators are defined and analysed.

This research provides the blueprints of an end-to-end system that supports crowdsensing scenarios and evaluates the performance of Internet of Things communication protocols, based on a series of experimental set-ups. This system initially tests the efficiency of IoT protocols in terms of message delivery reliability. By utilising an IoT simulator it was apparent that modern IoT protocols are adequate enough to perform the defined tasks.
However, even though simulations provide a great insight of how a system performs, they are not enough. Hence, this study presents the *IoTLab* smartphone application that runs on off-the-shelf devices. IoTLab enables researchers to conduct and monitor crowdsensing experiments in an energy and privacy-aware way. This tool is characterised by its transparency concerning the types of data shared and user’s consent. Furthermore, all captured and shared data are anonymised and completely disassociated from the owner of the smart device. This boosts user engagement during crowdsensing scenarios. Apart from its privacy-oriented features, one shall not forget that IoTLab is a powerful tool that supports a wide range of IoT communication protocols and has access to all the embedded sensors of a device. The author used IoTLab in an assortment of devices in order to evaluate the performance of the examined protocols in real-life scenarios. In doing so, a hybrid protocol was proposed that utilised both RabbitMQ and GCM.

The results highlight that the efficiency of each protocol fluctuates depending on the parameters of the experiment scenario. One can be power efficient in idle state or during light-load sensing, but may have unacceptably high energy consumption as the sensing becomes more frequent and demanding. On the other hand, a protocol can be imposing a big burden in terms of energy consumption but be able to transmit messages adequately independently of the sensing load. However, it was clear that the more demanding a crowdsensing scenario became, the less messages were successfully transmitted by the protocols. This led the author to propose a novel approach of conducting opportunistic sensing that utilises triggers. The results indicated that *triggered-sensing* is more robust than the conventional one.

Finally, this study points out that since all protocols have their advantages and drawbacks, selecting the “most appropriate protocol for a crowdsensing scenario” is predominantly subjective and depends on user preferences. It proposes an algorithm called *UPECA* to solve the multi-objective problem of identifying the best IoT communication protocol based on user requirements and experiment parameters. However, not all systems provide means to extract detailed user preferences. Hence, a mechanism that assigns weights based on simple relations is introduced. UPECA is evaluated against a series of crowdsensing experiments and manages to detect the best protocol per scenario. Following that, the author suggests that sensing tools also need to self-adjust during the course of a crowdsensing experiment in order to maintain high level of efficiency. As a result, the last part of this research examines the performance of the *adapting UPECA* and compares it to the “static” version. It is evident that adjusting the communication protocol, as a crowdsensing experiment evolves, is certainly beneficial for the overall performance of the system.
8. Conclusion

8.2 Future work

This research strives to investigate the impact of IoT communication protocols in the efficiency of mobile crowdsensing. Having identified the challenges and gaps of the literature, it presents solutions that would mitigate the problematic aspects of mobile crowdsensing. Nonetheless, as Section 2.4 presents, data transmission is only one of the steps during the lifecycle of an experiment. It would be interesting to explore the overall benefits of using the adapting UPECA together with techniques proposed by the research community.

Moreover, this work could be extended by evaluating the effect of using LTE, 5G networks, or even BLE instead of Wi-Fi, as there is proof that the latter might not be the most suitable approach in small data transfers [81].

Like smartphones, wearable devices are also resource-limited in terms of energy and computational power. Continuous sensor monitoring and sampling, as well as data manipulation, must rely on heavyweight processes that inevitably intensify energy consumption, resulting in reduced user acceptability. A potential solution to alleviate these limitations is mobile offloading. Nowadays, mobile offloading can be achieved by using a series of different approaches that have been designed and implemented in order to exploit the resources available in the cloud [123]. It is the author’s opinion that using “local” offloading approaches, in which a wearable device shares the computational burden by uploading data streams to a smartphone that belongs to the same micro-system could be advantageous. UPECA could have a great impact on the data-exchange process.
References


9. References


9. References


[36] N. D. Lane et al., “Piggyback CrowdSensing (PCS): Energy Efficient Crowdsourcing of Mobile Sensor Data by Exploiting Smartphone App
9. References


[41] MQTT.org, “MQTT Standardization discussion.”.


9. References


9. References


9. References


9. References


9. References


9. References


Appendix

This section includes the remaining artificial problems handled by the user-preference efficient crowdsensing algorithm. Since six of the original fifteen scenarios are already presented in section 7.3.2, only nine problems are detailed here, three for every experiment duration. Note, that for every artificial problem and for every set of weights generated, running UPECA results in significant lower energy consumption and higher message delivery. At worst, adapting communication protocols is as efficient as using one protocol throughout a sensing scenario.

A.1 Duration \(d = 5\) minutes

Scenario 2

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Simple UPECA</th>
<th></th>
<th>Adapting UPECA</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>263</td>
<td>idle</td>
<td>Burst GCM</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>35</td>
<td>light</td>
<td>0.50</td>
<td>6</td>
<td>Burst MQTT</td>
<td>0.42</td>
</tr>
<tr>
<td>2</td>
<td>idle</td>
<td>0</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
</tr>
<tr>
<td>Total:</td>
<td>0.058</td>
<td>6</td>
<td></td>
<td>Total:</td>
<td>0.049</td>
</tr>
</tbody>
</table>
### Scenario 3

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>253</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>250</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>250</td>
</tr>
<tr>
<td>34</td>
<td>light</td>
<td>Burst RabbitMQ</td>
<td>0.52</td>
<td>6</td>
<td>Burst RabbitMQ</td>
<td>0.43</td>
<td>6</td>
</tr>
<tr>
<td>13</td>
<td>light</td>
<td>Burst RabbitMQ</td>
<td>0.52</td>
<td>2</td>
<td>Burst RabbitMQ</td>
<td>0.43</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Total:</strong></td>
<td><strong>0.511</strong></td>
<td><strong>258</strong></td>
<td><strong>Total:</strong></td>
<td><strong>0.497</strong></td>
<td><strong>257</strong></td>
</tr>
</tbody>
</table>

### Scenario 5

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>272</td>
<td>light</td>
<td>Burst RabbitMQ</td>
<td>0.54</td>
<td>51</td>
<td>Burst RabbitMQ</td>
<td>0.54</td>
<td>51</td>
</tr>
<tr>
<td>28</td>
<td>heavy</td>
<td>Burst RabbitMQ</td>
<td>0.54</td>
<td>53</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Total:</strong></td>
<td><strong>0.547</strong></td>
<td><strong>104</strong></td>
<td><strong>Total:</strong></td>
<td><strong>0.544</strong></td>
<td><strong>105</strong></td>
</tr>
</tbody>
</table>
### Appendix

#### A.2 Duration d = 10 minutes

**Scenario 7**

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean Power</td>
<td>Mean Power</td>
</tr>
<tr>
<td>Time (s)</td>
<td>Load</td>
<td>Protocol</td>
<td>Messages Delivered</td>
<td>Protocol</td>
</tr>
<tr>
<td>436</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>87</td>
</tr>
<tr>
<td>67</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>13</td>
</tr>
<tr>
<td>52</td>
<td>medium</td>
<td>Triggered MQTT</td>
<td>0.61</td>
<td>29</td>
</tr>
<tr>
<td>39</td>
<td>heavy</td>
<td>Triggered MQTT</td>
<td>0.49</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>medium</td>
<td>Triggered MQTT</td>
<td>0.61</td>
<td>3</td>
</tr>
</tbody>
</table>

**Total:** 0.476 \( \times \) 166 \( \times \) 0.47 \( \times \) 168

**Scenario 8**

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mean Power</td>
<td>Mean Power</td>
</tr>
<tr>
<td>Time (s)</td>
<td>Load</td>
<td>Protocol</td>
<td>Messages Delivered</td>
<td>Protocol</td>
</tr>
<tr>
<td>436</td>
<td>light</td>
<td>Burst MQTT</td>
<td>0.42</td>
<td>85</td>
</tr>
<tr>
<td>82</td>
<td>idle</td>
<td>Burst MQTT</td>
<td>0.19</td>
<td>-</td>
</tr>
<tr>
<td>63</td>
<td>medium</td>
<td>Burst MQTT</td>
<td>0.51</td>
<td>31</td>
</tr>
<tr>
<td>19</td>
<td>medium</td>
<td>Burst MQTT</td>
<td>0.51</td>
<td>7</td>
</tr>
</tbody>
</table>

**Total:** 0.40 \( \times \) 123 \( \times \) 0.36 \( \times \) 129
### Scenario 10

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Protocol</td>
<td>Mean Power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Triggered RabbitMQ</td>
<td>0.52</td>
</tr>
<tr>
<td>73</td>
<td>light</td>
<td>0.51</td>
<td>145</td>
</tr>
<tr>
<td>145</td>
<td>heavy</td>
<td>0.52</td>
<td>45</td>
</tr>
<tr>
<td>226</td>
<td>light</td>
<td>0.51</td>
<td>120</td>
</tr>
<tr>
<td>123</td>
<td>heavy</td>
<td>0.54</td>
<td>18</td>
</tr>
<tr>
<td>30</td>
<td>medium</td>
<td>0.21</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>idle</td>
<td>0.54</td>
<td>19</td>
</tr>
</tbody>
</table>

**Total:** 0.249 | 342 | Total: 0.248 | 343
## Appendix

### A.3 Duration $d = 20$ minutes

**Scenario 12**

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>570</td>
<td>light</td>
<td>Simple UPECA</td>
<td>0.46</td>
<td>114</td>
<td>Adapting UPECA</td>
<td>0.46</td>
<td>114</td>
</tr>
<tr>
<td>150</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>30</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>30</td>
</tr>
<tr>
<td>234</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>171</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>34</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>34</td>
</tr>
<tr>
<td>42</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.49</td>
<td>39</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>40</td>
</tr>
<tr>
<td>31</td>
<td>light</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>6</td>
<td>Triggered MQTT</td>
<td>0.46</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>idle</td>
<td>GCM</td>
<td>0.19</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

**Total:**

<table>
<thead>
<tr>
<th>Simple UPECA</th>
<th>Adapting UPECA</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.40</td>
<td>0.37</td>
</tr>
<tr>
<td>223</td>
<td>224</td>
</tr>
</tbody>
</table>
### Scenario 13

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
<th>Protocol</th>
<th>Mean Power</th>
<th>Messages Delivered</th>
</tr>
</thead>
<tbody>
<tr>
<td>423</td>
<td>medium</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>252</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>252</td>
</tr>
<tr>
<td>50</td>
<td>Idle</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>268</td>
<td>Idle</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>179</td>
<td>Idle</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td>light</td>
<td>Triggered RabbitMQ</td>
<td>0.52</td>
<td>9</td>
<td>Triggered RabbitMQ</td>
<td>0.52</td>
<td>9</td>
</tr>
<tr>
<td>67</td>
<td>medium</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>69</td>
<td>Triggered RabbitMQ</td>
<td>0.54</td>
<td>69</td>
</tr>
<tr>
<td>70</td>
<td>Idle</td>
<td>-</td>
<td>0.21</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>43</td>
<td>light</td>
<td>Triggered RabbitMQ</td>
<td>0.52</td>
<td>8</td>
<td>Triggered RabbitMQ</td>
<td>0.52</td>
<td>8</td>
</tr>
<tr>
<td>52</td>
<td>heavy</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>50</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>50</td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td><strong>0.38</strong></td>
<td><strong>388</strong></td>
<td><strong>Total:</strong></td>
<td><strong>0.28</strong></td>
<td><strong>388</strong></td>
</tr>
</tbody>
</table>
## Scenario 14

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Load</th>
<th>Simple UPECA</th>
<th></th>
<th></th>
<th>Adapting UPECA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Protocol</td>
<td>Mean Power</td>
<td>Messages Delivered</td>
<td>Protocol</td>
<td>Mean Power</td>
<td>Messages Delivered</td>
</tr>
<tr>
<td>688</td>
<td>idle</td>
<td>Burst GCM</td>
<td>0</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>179</td>
<td>heavy</td>
<td></td>
<td>0.67</td>
<td>74</td>
<td>Triggered RabbitMQ</td>
<td>0.51</td>
<td>175</td>
</tr>
<tr>
<td>321</td>
<td>medium</td>
<td></td>
<td>0.65</td>
<td>104</td>
<td>RabbitMQ</td>
<td>0.54</td>
<td>192</td>
</tr>
<tr>
<td>12</td>
<td>idle</td>
<td></td>
<td>0</td>
<td>-</td>
<td>GCM</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td>0.27</td>
<td>178</td>
<td>Total:</td>
<td>0.22</td>
<td>367</td>
</tr>
</tbody>
</table>