Pontifícia Universidade Católica
do Rio de Janeiro

Luis Eduardo Talavera Ríos

An Energy-aware IoT Gateway, with Continuous Processing of Sensor Data

Dissertação de Mestrado

Thesis presented to the Programa de Pós–graduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Markus Endler

Rio de Janeiro
March 2016

Pontifícia Universidade Católica
do Rio de Janeiro

Luis Eduardo Talavera Ríos

An Energy-aware IoT Gateway, with Continuous Processing of Sensor Data

Thesis presented to the Programa de Pós–graduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC–Rio as partial fulfillment of the requirements for the degree of Mestre em Informática.

Prof. Markus Endler
Orientador
Departamento de Informática — PUC–Rio

Profª. Noemi Rodriguez
Departamento de Informática — PUC-Rio

Prof. Jose Viterbo
Departamento de Ciência da Computação e Pós-graduação — UFF

Prof. Márcio da Silveira Carvalho
Coordinator of the Centro Técnico Científico — PUC–Rio

Rio de Janeiro, March 16^{th}, 2016

Luis Eduardo Talavera Ríos

BSc. in Computer Science at San Pablo Catholic University (UCSP), Arequipa - Peru in 2013. Joined the Master in Informatics at Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in 2014.

Ficha Catalográfica

Talavera Ríos, Luis Eduardo

An Energy-aware IoT Gateway, with Continuous Processing of Sensor Data / Luis Eduardo Talavera Ríos; advisor: Markus Endler. — 2016.

73 f. : il. (color); 30 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2016.

Inclui bibliografia.

1. Informática – Teses. 2. Sistemas Distribuídos. 3. Dispositivos móveis. 4. Consumo de energia. 5. Processamento de eventos complexos. 6. Internet das Coisas. 7. SDDL. I. Endler, Markus. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I want to firstly thank my parents Eduardo and Mercedes, and my family in general, for all their unconditional support during these years of hard study away from home. They taught me how important it is to give always the best of myself in order to chase my dreams. Also, I need to thank my sister Natalia, who has been a terrific friend and counselor in my best and worst moments.

I would like to give a special thanks to my advisor Markus Endler, for giving me the incredible opportunity to work under his guidance, and for all the knowledge he shared with me through our several conversations and discussions. Working with him has been the most rewarding experience of my life and without his support this dissertation would not have been possible.

To my friends in Perú, the LAC, the Telemidia and the Informatics Department, for their support, advice and exchange of ideas during the last two years. Specially to José Talavera, Álan Lívio, Marcos Roriz, Bruno Olivieri and André Mac Dowell who have been like brothers to me during this stage of my life.

Finally, I would like to give thank to PUC-Rio and CAPES for bringing me this wonderful opportunity.

Abstract

Talavera Ríos, Luis Eduardo; Endler, Markus (Advisor). **An Energy-aware IoT Gateway, with Continuous Processing of Sensor Data**. Rio de Janeiro, 2016. 73p. MSc. Dissertation — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Few studies have investigated and proposed a middleware solution for the Internet of Mobile Things (IoMT), where the smart things (Smart Objects) can be moved, or else can move autonomously, but remain accessible from any other computer over the Internet. In this context, there is a need for energy-efficient gateways to provide connectivity to a great variety of Smart Objects. Proposed solutions have shown that mobile devices (smartphones and tablets) are a good option to become the universal intermediates by providing a connection point to nearby Smart Objects with short-range communication technologies. However, they only focus on the transmission of raw sensor data (obtained from connected Smart Objects) to the cloud where processing (e.g. aggregation) is performed. Internet Communication is a strong battery-draining activity for mobile devices; moreover, bandwidth may not be sufficient when large amounts of information is being received from the Smart Objects. Hence, we argue that some of the processing should be pushed as close as possible to the sources. In this regard, Complex Event Processing (CEP) is often used for real-time processing of heterogeneous data and could be a key technology to be included in the gateways. It allows a way to describe the processing as expressive queries that can be dynamically deployed or removed on-the-fly. Thus, being suitable for applications that have to deal with dynamic adaptation of local processing. This dissertation describes an extension of a mobile middleware with the inclusion of continuous processing of sensor data, its design and prototype implementation for Android. Experiments have shown that our implementation delivers good reduction in energy and bandwidth consumption.

Keywords

Distributed Systems; Mobile devices; Energy consumption; Complex event processing; Internet of Things; SDDL.

Resumo

Talavera Ríos, Luis Eduardo; Endler, Markus. **Um Energy-aware IoT Gateway, com Processamento Contínuo de Dados de Sensor**. Rio de Janeiro, 2016. 73p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Poucos estudos têm investigado e propôs uma solução de middleware para a Internet das Coisas Móveis (IoMT), onde as coisas inteligentes (Objetos Inteligente) podem ser movidos, ou podem mover-se de forma autônoma, mas permanecem acessíveis a partir de qualquer outro computador através da Internet. Neste contexto, existe uma necessidade de gateways com eficiência energética para fornecer conectividade para uma grande variedade de objetos inteligentes. As soluções propostas têm mostrado que os dispositivos móveis (smartphones e tablets) são uma boa opção para se tornar os intermediários universais, proporcionando um ponto de conexão para os objetos inteligentes vizinhos com tecnologias de comunicação de curto alcance. No entanto, eles só se preocupam apenas sobre a transmissão de dados de sensores-primas (obtido a partir de objetos inteligentes conectados) para a nuvem onde o processamento (e.g. agregação) é executada. Comunicação via Internet é uma atividade de forte drenagem da bateria em dispositivos móveis; Além disso, a largura de banda pode não ser suficiente quando grandes quantidades de informação estão sendo recebidas dos objetos inteligentes. Por isso, consideramos que uma parte do processamento deve ser empurrada tão perto quanto possível das fontes. A respeito disso, processamento de eventos complexos (CEP) é muitas vezes usado para o processamento em tempo real de dados heterogéneos e pode ser uma tecnologia chave para ser incluído nas Gateways. Ele permite uma maneira de descrever o processamento como consultas expressivas que podem ser implantados ou removidos dinamicamente no vôo. Assim, sendo adequado para aplicações que têm de lidar com adaptação dinâmica de processamento local. Esta dissertação descreve uma extensão de um middleware móvel com a inclusão de processamento contínuo dos dados do sensor, a sua concepção e implementação de um protótipo para Android. Experimentos têm mostrado que a nossa implementação proporciona uma boa redução no consumo de energia e largura de banda.

Palavras–chave

Sistemas Distribuídos; Dispositivos móveis; Consumo de energia; Processamento de eventos complexos; Internet das Coisas; SDDL.

Contents

List of Figures

List of Tables

1
Introduction

Nowadays information about the physical world is unconsciously sensed and gathered through smartphones, smartwatches, sensors and other devices which are quickly becoming a crucial part of our daily lives. However, these immense volumes of data are not easily accessible or remains separated from the Internet since each type of hardware device requires a specific software to communicate, yielding a very restricted view of the environment. Additionally, in order to include a meaning to those continuous streams of data, they need to be processed in real-time in order to detect important situations (e.g. a gas leak, a IT system failure), and allow fast responses.

In this context, Internet of Things (IoT), one of the major paradigms that computing is facing nowadays (Gubbi et al., 2013), appears as a vision to extend the Internet connectivity to a wide variety of small, embedded, low-power wireless devices also known as *Smart Things*. Furthermore, there has been an incredible increase in the demand for IoT applications in many areas which go from sports, health care, public safety to business, commercial services, social networks, etc. But despite the huge number of potential applications and the increasing proliferation of appliances with embedded processing and wireless communication capacity, yet there is no widely accepted approach, established standards and consolidated technologies for IoT. In other words, extraction of meaningful information from tens of billions of sensors and prompt control of actuators in (near) real-time is still a challenge.

IoT is evolving towards a heterogeneous network including a mix of IP-based connectivity and an array of short-range, wireless technologies (e.g. Bluetooth Smart, ZigBee). The latter allows personal mobile devices (smartphones, and tablets) to connect to the available sensors in their vicinity which can be considered as small private sensor networks that provide heterogeneous data about the physical environment. In particular, very few studies have focused on the Internet of Mobile Things (IoMT) where a Mobile Object (M-OBJ) may be any movable thing that carries sensors and/or actuators and has some means of wireless connectivity. In this context of general and unrestricted mobility of smart things, different resources (M-OBJs) can become available and unavailable at any moment without previous warning. Finally, it is worth noting that in IoMT mobility is just an option: M-OBJs and IoT-gateways may also be deployed in a static configuration, and be associated with a specific place, for applications such as Smart Home/Buildings.

In the following we present two hypothetical IoMT applications:

- **Application 1**: In a region with high density of smartphone users, such as a metropolitan area, we can imagine the need for collaborative air quality monitoring. Common citizens may obtain tax incentives to install and deploy affordable Wireless Air Monitoring Stations (WAMS) in their yards, along neighborhood driveways, parks or other public spaces. All such WAMS would have CO, NO_2, SO_2, Lead sensors, etc., a short range and low-power wireless interface, be weather-resistant and run on solar energy. In this context, the collaborative monitoring app would be a crowd-sourced one where pedestrians passing close to some WAMS would donate their smartphone's Internet connectivity and energy to upload the current collected sensor data from the nearby WAMS to a city-wide monitoring service in the cloud, where all this information would be presented on a map both on-line and in consolidated statistics, for access by any citizens. Actually, through this IoMT application air quality indeed would not be be monitored uniformly for the entire city region, but only at those parts of the city where it is most relevant, i.e., the places with much intense pedestrian and bike traffic. Moreover, some citizens may even opt for an air monitoring on the go-attitude, carrying a smaller and lighter version of the WAMS on their bike baskets or their knapsacks, so as to measure the air pollution on their ways. And through their smartphone, this data would be uploaded to the central monitoring service.
- **Application 2:** Nowadays, several goods and merchandise require specific minimal transportation and storage conditions on their routes from producer to consumer. For example, meat and some fruits need ambient temperatures of less than 10 degrees Celsius, special flowers and plants should be in environments with air humidity above certain level, livestock requires smooth movement as well as places with sufficient air circulation (i.e., O_2 concentration), etc. Hence, it would be advantageous to be able to monitor the ambient and movement conditions along all the transport path of these and other "sensitive" merchandise. This could be done by placing some M-OBJs close to the goods, and having the sensor values probed in all the stages of transportation and intermediate storage. These data could then be send in real-time to interested users (e.g. the customer, or the transportation company) so as to early detect some non-conformance in the transportation conditions, or else, be the input for transport reports. Such a monitoring could be achieved by having IoT-gateways placed in the array of vehicles (trucks), vessels or

delivery personal involved in the transportation. By such, as soon as the merchandise cargo is loaded, stored or picked by the next transportation means, the IoT-gateways would connect with the M-OBJs, send messages acknowledging the arrival/ handing-over of the goods to the clients, and start receiving, checking the M-OBJs sensed data, and eventually sending alert notifications about inadequate transport conditions.

Furthermore, according to Francis daCosta (daCosta, 2013), IoT communication involves small but frequent messages, where each message individually is unimportant, but the statistical properties of the corresponding data flows carry the relevant pieces of information. In this regard, Complex Event Processing (CEP) is a rather novel software technology for continuous real-time processing of high volumes of data items (low level information) that allows an easy specification of patterns (using SQL like queries) that represent complex situations (e.g. a fraud in a bank transaction) (Mayer, 2013). CEP processing considers heterogeneous data from different sources, and looks for event patterns of interest (causal, logical or temporal) and eventually generates high-level events that represent these detected patterns (e.g. a crash or a fraud). Nevertheless, current CEP solutions are cloud-centralized, meaning that they only distribute their processing among server nodes (Govindarajan et al., 2014). Almost none of them have considered using CEP components that execute on nodes at the edges of the network.

Moreover, recent works (Zachariah et al., 2015) have proposed the use of mobile devices (phones/handhelds) as the propagator nodes in IoT, since they can gather the information of the environment and transmit it to cloud-based servers. However, the huge volume of transmissions among those mobile devices and the cloud would drain the device's battery, overload the network with low-level information (raw pieces of data) and delay the delivery of data if the network quality is not good enough to provide sufficient bandwidth. As the data volume grows, this approach of data processing solely in the cloud becomes unsustainable (Saleh e Sattler, 2013). For the previous reasons, many works advocate that for IoT it is important to have as much in-network processing as possible, by performing filtering, aggregation and pre-processing over the data streams before sending any information to the cloud (Billet e Issarny, 2014). All these in-network pre-processing functions can be well expressed in CEP; which therefore should be provided in the IoT-gateways close to the sources, e.g., sensors. In fact, recent works have shown that current mobile devices have sufficient capacity to execute CEP, allowing them to perform such processing (Stipkovic et al., 2013).

To illustrate the need for in-network processing, consider a medical mobile application that should detect changes in the patient's health condition, so that corresponding health professionals can be notified and respond immediately with some action. In this context, heart rate, temperature, and movement sensors could provide information about the patients, and the mobile applications could pre-process the sensor data to only send relevant events (e.g. high temperature and/or low heartbeats) to the cloud. In another scenario, imagine the actions that you will need to be automatically executed in your home and work place. Houses are very likely to have temperature, light, and motion sensors in order to start some actions like set the temperature level to keep a fresh environment, or turn on the lights at night. However, it will a completely different case in the work place, where we could use information about available parking places, scheduled meetings, and rain forecasts to alert us about different opportunities at work. Hence, our research aims to find out the feasibility of using CEP at mobile devices in order to allow in-network processing of sensor data (local pre-processing). Discover if it is suitable to have a dynamic deploy and exchange of CEP queries in mobile devices. And, expose the main trade-off between local pre-processing versus send all the sensor data to servers in the cloud for processing.

1.1 Problem Statement

Being able to detect relevant situations (e.g. change in the stock market or an accident) over continuous data flows in (near) real-time has many applications in different areas like industry and health care (Schilling et al., 2010; Stipkovic et al., 2013). In this context, due to the need for immediate responses, solutions like Hadoop or database management systems (DBMS) cannot be applied for their high latencies for large volumes of data (Chen et al., 2014). In such cases the use of CEP has been widely used thanks to its characteristics that facilitate the specification of event patterns, and its fast detection on high traffic event streams that can be considered (almost) real-time (Saleh e Sattler, 2013; Schilling et al., 2010; Govindarajan et al., 2014; Stojanovic et al., 2014; Jaein et al., 2012; Eggum, 2014; Schmidhäuser, 2014; Stipkovic et al., 2013; Dunkel et al., 2013).

In the new generation of IoT, the edge nodes are usually mobile devices (e.g. smartphones and tablets) which are becoming more popular and widely spread (Schmidhäuser, 2014). These devices are used as propagator nodes for simpler smart objects in order to enrich and route the sensed information to the cloud. Moreover, recently Cisco has proposed a new computing paradigm

called Fog computing (Bonomi et al., 2012), allowing for low-latency processing on resource-constrained devices near the edge like mobile devices or routers, while computational-intensive processing is performed in the cloud. In fact, there are already some previous studies where Android smartphones work with Esper (an open source CEP engine). Results in (Jaein et al., 2012) show low levels of throughput and imply that it has a substantial overhead. However, a more recent study (Eggum, 2014) mentions that the introduction of ART[1] (the new Android runtime) resolves many performance related issues that were previously seen with Dalvik (Jaein et al., 2012), and proves that future revisions of Android could be suitable for this environment.

However, most of the current CEP solutions for IoT are cloud-centralized (only distribute their processing among server nodes), and mobile devices (IoT-gateways) act only as data sources to the cloud. These approaches will be unfeasible, since in the IoT domain it is expected that 50 billion or more devices will be interconnected through the Internet, generating an enormous amount of information to be continuously sent to the cloud (Govindarajan et al., 2014). Network quality in mobile environments could not be always good enough to provide sufficient bandwidth. Moreover, forwarding all the information to the cloud is unfeasible for mobile devices since it will consume their energy, and also "introduces a too high energy cost for the envisioned IoT scale, considering the energy cost of communication over computation. Given the growing requirement of a distributed solution for data stream management in IoT" (Billet e Issarny, 2014), the increasing popularity of mobile devices functioning as edge nodes (Schmidhäuser, 2014) and the fact that CEP is a promising technology for stream processing in IoT (Chen et al., 2014; Govindarajan et al., 2014), this dissertation aims at answering following research questions:

a) Is it feasible to deploy CEP queries at mobile devices to do in-network processing of sensor data from the mobile device and from nearby Smart things (a.k.a. M-OBJs)?

b) How is the energy consumption of the mobile device affected by the choice between local pre-processing of data versus send all the sensor data to servers in the cloud for processing?

c) Which is the main trade-off between local pre-processing in the mobile device versus processing in the cloud?

[1]https://source.android.com/devices/tech/dalvik/

1.2 Objective and Contribution

The main objective of this work is to respond to the statements of section 1.1, for that we designed and implemented a dynamic complex event processing service that allows in-network processing of sensor data streams presented by smart objects. Thus, we propose the concept of *Mobile Event Processing Agent (MEPA)* as a new service for IoT gateways, that allows an easy specification of situations (e.g. a crash, a fraud) with the use of CEP. It can offload part of the cloud processing to reduce the amount of transferred data. Moreover, depending on the available resources, it is easy to change the running CEP queries since Esper (the CEP technology used in this dissertation) allows an on-the-fly reconfiguration avoiding the service downtime. The previous points lead us to research the impact of local processing on mobile device's energy and bandwidth consumption compared to sending the raw sensor data to a service in the cloud. A benchmark showing the performance and the overhead of the presented solution could act as a guideline for who wishes to use CEP solutions in this domain.

1.3 Methodology

An initial study about energy consumption in mobile devices and recent works in sensor-gateways, in-network processing and CEP solutions in IoT was first executed in order to identify the State of the Art. It also helped us to understand how CEP should be evaluated in a mobile environment, where energy overhead and bandwidth usage are considered the most important requirements.

Given the nature of this dissertation, it required an empirical evaluation where we conducted a series of quantitative experiments using several M-OBJs (SensorTags [2] and different CEP queries with different complexities. We measured the **energy overhead** and **bandwidth usage** generated from the communication, and local processing. Relating the outcome between the observed values helped us to draw strong conclusions.

1.4 Outline

This dissertation is structured in the following way:

- **Chapter 2 - Background:** This chapter provides an overview of the enabling technologies and main concepts used in the project.

[2] Texas Instruments CC2541 Sensor Tag - http://www.ti.com/lit/ml/swru324b/swru324b.pdf

- **Chapter 3 - Energy Consumption in Mobile Devices:** Here we introduce the basic knowledge about energy consumption in mobile devices.
- **Chapter 4.2 - Mobile Processing and Dynamic Deployment:** We present our approach to handle in-network processing and its general architecture.
- **Chapter 5 - Evaluation:** This chapter presents and discusses the evaluation results obtained from the tests over the local processing service with real sensors.
- **Chapter 6 - Related Work:** In this chapter we introduce similar works related to distributed CEP and sensor data processing.
- **Chapter 7 - Conclusion:** Finally, this chapter summarizes our work and propose the main lines for the future work.

2
Background

This chapter introduces the main concepts and basic technologies used in this dissertation. Section 2.1 describes Complex Event Processing (CEP), the technology used for in-network sensor data processing. Afterwards Section 2.2 gives an insight about the middleware technology used for cloud-based services.

2.1
Complex Event Processing

Complex Event Processing (CEP) is a software technology for the dynamic analysis of massive data- or event-streams in (near) real-time. It was initially proposed by David Luckham in the mid 1990s. Using CEP it is possible to describe causal, temporal, logic and contextual relations among the events, and check the event stream for their occurrence. An event is any activity, procedure or decision that happens in the real world. In general, an event describes the change of a state or a property of a real or virtual object. Examples range from technical events such as the change of the measured temperature probed from of a sensor in a production plant to business events, such as changes in a company's stock value.

CEP provides capabilities of filtering, aggregation, correlation and analysis over continuous streams of data. Moreover, the search for event patterns considers the event stream over a period of time (in a time our event window). Depending on the expressiveness of the used operators, different kinds of patterns may be detected:

- Detection of simple patterns using boolean operators. For example, the simple pattern (A and B and not(C)) describes the occurrence of events A and B but the absence of event of type C.
- Detection of complex patterns using the followed-by ($\rightarrow$) and the Every operators. With these, it is possible to check the causality and the repetition of simpler events and patterns. For example ((A or B)$\rightarrow$C) checks if there is the occurrence of events A or B that are followed by any event of type C. And the pattern every (A$\rightarrow$B) picks all the pairs of instances of event types A and B where B follows A.
- Abstraction of Event patterns: whenever a pattern is detected from simple events, a new event of a higher level of abstraction can be generated, that represents the detected pattern. These more complex

events yield high-level views of such patterns. Typical examples are the sum or the average of simpler events. For example, an average of the stock price over the last x days can be consolidated in a x-day-avg event, and this event can in turn be subject to further pattern analyses.

In CEP the logic of the event processing is explicitly and declaratively expressed as queries. An event processing query describes a specific (re)action that should take place whenever the corresponding pattern is detected. Each query consists of a condition and an action part. The condition part consists of one or more connected patterns that are checked against the event stream. If the condition is satisfied (the analysed events in the stream match the pattern) we say that the query matched, and then the action part of the rule is fired. The action part typically describes the activation of a callback procedure of a listener, which in turn may activate a application-specific service, or may generate a complex event to be re-inserted into the event stream.

The entire event processing logic of an application is structured in CEP as a workflow among connected Event Processing Agents (EPA). EPAs are software modules that receive an input event stream and execute one or more event processing queries for pattern matching and eventually reacting upon them by outputting other events. The complete set of interconnected EPAs is called an Event Processing Network (EPN), an EPN can be distributed among multiple physical networks and computers. The top-level architecture of an EPN consists of three layers: event producers, event consumers and event processing logic (see Figure 2.1), plus the communication channels between the layers. An EPA can act as any of the three roles, an event producer at one moment, a consumer at another time, or an event processor for the events that it receives from the sources.

Event Producers Also known as event sources, is the layer in charge of producing or capturing events in the internal or external environment (e.g. sensor values, financial trades) to later be forwarded to the event processing logic layer. Event producers could be a software module, different sensors or even a clock.

Event Processing Logic The event processing logic consists of a CEP engine and continuous queries. The CEP engine is where all the events are analyzed, while the queries are a set of patterns that describe the situations of interest presented in the form of combinations and transformation of low level events with causal, temporal, spacial and other relations. Queries are defined in a declarative SQL-like language. An

example would be a query that detects credit-card frauds, when two transactions are realized in far places, within a short period of time.

Event Consumers Event consumers receive complex events from the Event processing logic in order to deal with detected situations. Typical event consumers could be applications for visualizing events or services in the Cloud.

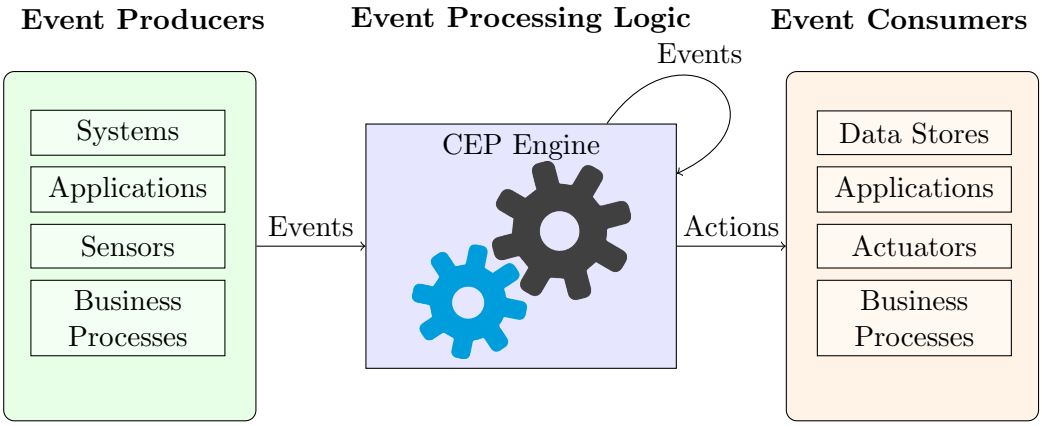


Figure 2.1: Event-Driven Architecture of a CEP System[1]

2.1.1 Main Concepts of Event Processing

There are some fundamental concepts that we have to know at the moment of using a solution for event processing. *Events* that happen in the real world are usually a combination of data that contains different properties such as time, location and sequence. An event could be also described as the absence of something that was expected to happen. Moreover, the word *query* is used to refer a programming representation of an event (an occurrence), for example a query that represents all the transaction events with a value equal or higher to 5000, could be: `SELECT * FROM Transactions[amount >= 5000]`.

Queries that involve functions like aggregation or negation (absence of an event) are not adequate for infinite streams because they can only be answered after the stream ends (Eckert e Bry, 2009). In such cases, queries require an upper and lower bound (range) that constraints the streams, such scope is called a *Window*. Additionally, windows can be classified in time- and count-based. Time-based windows define the bounds as seconds, hours or days, while count-based use the amount of data items. Besides, depending on how window's moves and process information they can be classified in:

[1]Source: Opher Etzion, Peter Niblett. Event Processing in Action. Manning, 2011

1. *Sliding Windows* have a fixed size and process the information in a continuous block of events. They have a lower and upper bounds that advance with time or as data items are inserted into the engine. Every time these windows vary, they process all the data that they contain within their bounds (Eggum, 2014) and release the oldest ones. As an example, you can see in Figure 2.2, where dashed events are the ones leaving the window, and the new events are the ones with an arrow above.

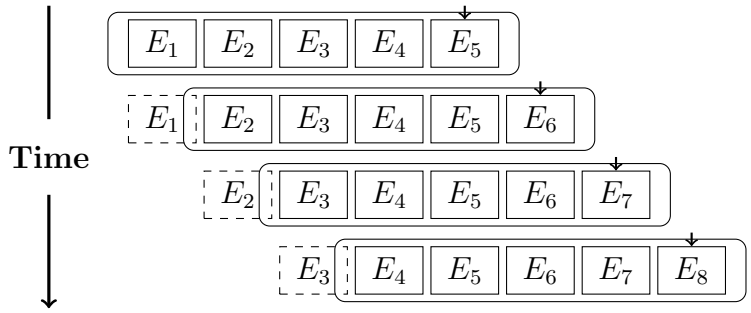


Figure 2.2: Sliding Window

2. *Jumping Windows* waits until their size n is filled with events, to process in a bulk operation all the data items contained in it. The main difference to sliding windows is that jumping windows will never contain a data item that could have resided in the window before the processing (Eggum, 2014). Sliding windows only remove the oldest events. Figure 2.3 shows how jumping windows remove all the contained events after they are completely filled, and then start to include new events.

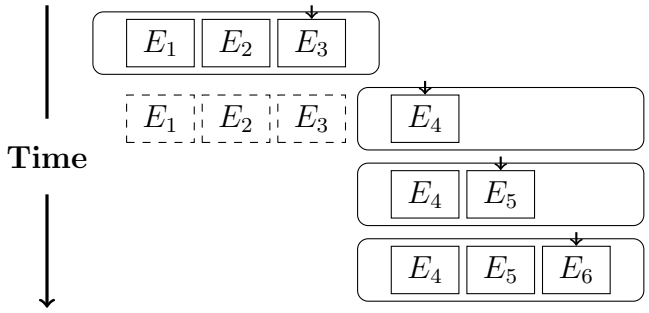


Figure 2.3: Jumping Window

Queries usually join several event items to express a more complex event over time (Composition). Such unions are represented by different relationships between events. Selectivity is the capacity that allows to select a number of events that match certain criteria. Projection can generate new events that contain a sub-set of the properties from previously seen events. Aggregation is

defined as a combination of events and it is related to counting, summarizing, or averaging. Different from projection and selectivity, aggregation needs to see the entire input, so they must be used in combination with windows (Eckert e Bry, 2009).

CEP queries frequently implement complex correlation functions that sometimes can be divided into different sets of sub-queries. The sub-queries can perform certain steps of the correlation separately and provide the results to a CEP root query in the cloud. For example, a CEP root query could be a computational-intensive processing (e.g. correlations of pre-processed sensor data), while the the rest of sub-queries could be simple correlations on collected sensor data (e.g. aggregators in a window of time, filters) (Schmidhäuser, 2014). In our approach mobile devices collect most of the sensor data, processing these sensor data directly in them can improve the performance of the system regarding the possible slow network connections, as well as reducing the device's energy consumption.

2.1.2 Esper

There are several technologies and languages for CEP, but we focus on Esper[2], since it is an open-source software available under GNU GPL license and one of the most widely used CEP engine. It was initially released in 2006, and has a continuous release to this date (2016) from an active community (Eggum, 2014). Esper events allow a rich object representation, since it supports all aspects of object-oriented design as well as dynamic typing. Other CEP technologies force a flat Map-like tuple-set definition of events. It also offers a rich set of parameterizable data windows (expiry policies) while most other engines provide a very small set of very simple rolling, sliding or hopping windows[3].

Esper uses a declarative Event Processing Language (EPL) that derives many properties from the SQL standard, to define the event processing queries used to detect the patterns over the streaming data. An event processing engine analyzes the event streams and executes the matching queries in-memory (Stipkovic et al., 2013). An application that embeds Esper, can employ one or more engine instances with different configurations and queries. However, since mobile devices still possess limited resources like processing power it is recommended that all EPAs share one Esper instance instead of having an own instance for each agent (Dunkel et al., 2013). Finally, and more important,

[2]http://www.espertech.com/esper/documentation.php
[3]http://www.espertech.com/esper/faq_esper.php#comparison

it has been successfully ported to Android with the name of Asper (Eggum, 2014). This allows us to include it as an additional service for the IoT gateways. Thus, enabling a flexible and efficient way of real-time processing of sensor data in mobile environments, where data are obtained either by the sensors embedded in a mobile device, or by the sensors connected to it.

2.2 Scalable Data Distribution Layer Middleware

The concept of the Mobile Hub in principle is independent of the protocol/technology used for Internet connectivity and communication with the cloud. Nevertheless, the current implementation is based on the utilization of a middleware for mobile-cloud communication called the Scalable Data Distribution Layer (SDDL) (David et al., 2013; Vasconcelos et al., 2013; Vasconcelos et al., 2014), which connects mobile nodes [4] with an IP-based wireless data connection to stationary nodes in a wired core network, the SDDL Core. In this core that executes in a cloud, some of the stationary nodes are information and context data processing nodes, others are gateways for communication with the mobile nodes, and yet others are monitoring and control nodes operated by humans, and capable of displaying the mobile nodes current position (or any other context information), managing groups, and sending message to the mobile nodes. The mobile nodes would become the gateways (hubs) of communication for providing Internet connectivity to Smart Objects, as shown in Figure 2.4.

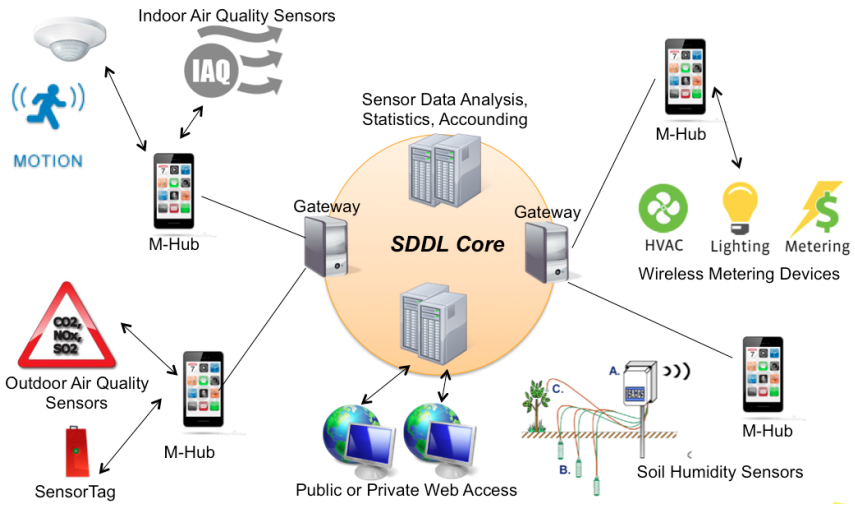


Figure 2.4: Example of M-OBJs sensing using the SDDL and the M-Hubs

[4] A mobile node is the generic term for an end user terminal or a Mobile Hub

SDDL employs two communication protocols: the Data Distribution Service (DDS) Real Time Publish/Subscribe Protocol for the wired communication within the SDDL Core, and the Mobile Reliable UDP (MR-UDP) (Silva e Roriz., 2013) for the inbound and outbound communication between the core network and the mobile nodes. DDS (Pardo-Castellote, 2003) is an OMG standard that specifies a peer-to-peer middleware architecture for real time and high-performance data distribution, with Quality of Service (QoS) contracts between producers and consumers of data (e.g., reliable communication, data persistency, priority lanes, etc.). MR-UDP, on the other hand, is a Reliable-UDP with mechanisms for tolerating intermittent connectivity, dynamic IP address changes of the Mobile nodes and reaching these nodes behind firewalls/NATs. It is used by the mobile nodes to connect with a special type of SDDL Core node called *Gateway (GW)*, of which any number can be deployed in the SDDL Core. Each Gateway maintains one independent MR-UDP connection with each mobile node, and is responsible for translating application-messages from MR-UDP to the intra-SDDL core protocol, and, in the opposite direction, converting SDDL Core messages to MR-UDP messages and delivering them reliably to the corresponding mobile nodes. Any mobile node uses the *ClientLib*, a library used to establish and manage a MR-UDP connection with any of the Gateways and that hides most MR-UDP details and message re-transmission issues from the application layer, and also supports a fully application-transparent handover of the mobile nodes between SDDL Gateways. The M-Hub is thus nothing else but a special kind of mobile node that opportunistically connects to M-OBJs.

The SDDL Core includes several other specialized services in charge of load balancing, data persistency, data stream processing and groupcast communication, whose explanation can be found in papers (David et al., 2013; Vasconcelos et al., 2014). In particular, it is possible to deploy CEP root nodes for the in-network sub-queries on the sensor data collected by the mobile nodes. The interested reader can download a VM with pre-installed SDDL, as well as find examples and tutorials for implementing SDDL-based applications in Java, Android and Lua[5].

[5]http://www.lac-rio.com/dokuwiki/doku.php?id=tutorial

3
Energy Consumption in Mobile Devices

Personal mobile devices (smartphones, phablets and tablets) and mobile Internet are becoming increasingly ubiquitous[1], more affordable, powerful and opportunistic; and intermittent connectivity is becoming a common place for mobile, wearable and embedded technologies that need to constantly share data. However, mobile devices have a limited amount of energy obtained from batteries which haven't evolved as much as other components in the last years, limiting its use to a few tasks over the day.

Usually, there are a lot of applications and services running at the same time in mobile devices, and this can reduce the device's battery life to few hours of operation. This could be due to a lousy management or a high requirement of resources, for example keep using the camera or the GPS even when the application is in background. Additionally, mobile internet services are becoming popular, so wireless data transmission is turning into a major energy consumer on mobile Internet devices (Tarkoma et al., 2014). As an example, according to a review from AnandTech[2], the Motorola Moto X (2013) device can be idle for up to 576 hours, but it can only maintain up to five hours of data access on 4G and eight hours on Wi-Fi. In other words, there is a considerable difference in energy consumption when keeping the mobile device idle (fully awake without active applications) and when it is using the wireless Internet connection (Pentikousis, 2010).

In this chapter we will introduce the basic concepts about the energy consumption in mobile devices. It is important to understand where and how the energy is used: how much of the system's energy is consumed by which parts and under what circumstances, in order to maximize the operation time of mobile devices. This problem is very complicated, since mobile devices consist of various hardware and software systems that work together.

3.1
Mobile Environment Evolution

Nowadays, mobile devices are becoming increasingly complex and distributed. They can now communicate among them and with simpler peripheral devices (e.g. smart watches, a myriad of sensors, and augmented reality

[1] According to eMarketer, in 2014 we already had 1.75 billion smartphone users - almost 24% of the world population!

[2] http://www.anandtech.com/show/7235/moto-x-review/6

devices) by using protocols such as Bluetooth, Bluetooth Low Energy (BLE), and ANT+. They also maintain one or more connections with Internet servers in order to synchronize their platforms and applications with the cloud, producing an environment similar to the one shown in Figure 3.1. This expands the optimization beyond the device (inter-device level), since now improvements have to be put in a local communication context. A more common approach is to offload certain power-hungry tasks that are frequently used to the cloud, to mitigate the battery consumption in the mobile devices (Tarkoma et al., 2014).

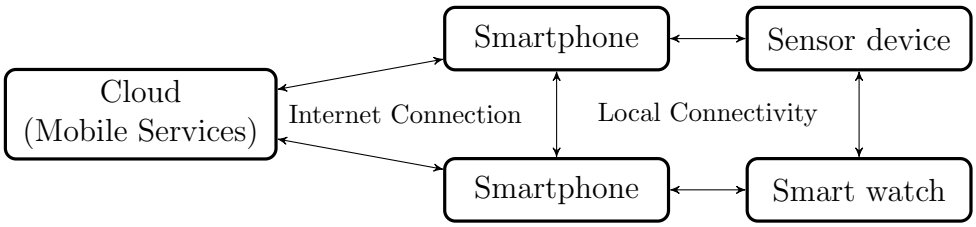


Figure 3.1: Mobile Environment. Adapted from (Tarkoma et al., 2014)

Moreover, mobile devices have evolved into general-purpose computers that get their power from rechargeable batteries. They now include high-resolution resolution digital cameras, and displays that have become larger with a superior quality. New interaction methods, such as touch screens and voice recognition, are also adopted. This environment will open the doors to many new innovative applications that sense and interact with users in different ways. However, it also presents many challenges that includes energy efficiency (Tarkoma et al., 2014).

In contrary to the many new features that mobile devices include now, their battery capacity haven't evolved as much as its energy requirement (see Table 3.1). In terms of watt-hours, battery is increasing linearly as a result of gradual advances in Li-ion batteries since the early 1990s. Year after year, average battery capacity increased 3.52 percent between 1996 and 2008, while clock speed has more than tripled from 2008 to 2013 (Tarkoma et al., 2014). As an example; an Apple iPhone 3G had a clock speed of single-core 412 MHz and a battery of 1219 mAh in 2008, while in 2013 a Samsung Galaxy S4 had a quad-core 1600 MHz clock speed and a battery of 2600 mAh. John Baker imply that "lithium-based cell technologies are likely to represent the pinnacle of cell development in terms of energy efficiency". Unfortunately, he also predicts that only incremental improvements are most likely to occur, resulting in 10-20 percent increases in energy densities (Pentikousis, 2010). So, we should not expect a big increase of battery capacity in the mid-term.

	1995	2000	2005	2010	2015
Cellular Generation	2G	2.5G - 3G	3.5G	Pre-4G	4G
Standard	GSM	GPRS	HSPA	HSPA, LTE	LTE, LTE-A
Downlink (Mb/s)	0.01	0.1	1	10	100
Display pixels (x 1000)	4	16	64	256	1024
Comms modules	-	-	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth LE, RFID
Battery capacity(Wh)	1	2	3	4	5

Table 3.1: Evolution of mobile phones. Adapted from (Tarkoma et al., 2014)

Network traffic has also grown tremendously over the last few years since the appearance of laptops, smartphones, tablets and other portable devices. On the one hand, technologies such as cellular networks 2G-5G and Wi-Fi can now provide wireless speeds of hundreds of Mbps. In the case of Wi-Fi, it can cover certain areas, such as a room or an office (Tarkoma et al., 2014), while cellular networks can cover bigger areas like a city. Mobile devices switch between them depending on the availability and quality of the wireless networks. On the other hand, Machine-to-Machine (M2M) communication is expected to growth significantly in the next years, because a lot of devices are including short-range wireless technologies to allow mobile devices to communicate with a growing number of sensors and actuators.

3.2 Components of Mobile Devices

A mobile device consists of software and hardware components. The hardware components (see Figure 3.2) are the ones that require energy in order to function (e.g. microprocessors, wireless network interfaces, and cameras). The I/O controllers are implemented by a specific chip, while Wi-Fi, cellular modem, sensors, and other auxiliary components are connected through system buses (e.g. internal USB, or multichannel buffered serial port - McBSP) (Tarkoma et al., 2014).

The number of cores in mobile devices have increased in the past years. Such cores can be categorized in two types: high and low performance, the former handles heavy tasks and the latter background tasks (e.g. email synchronization). An algorithm monitors the load on each CPU and migrates tasks between the higher and lower performance CPUs. Another component is the GPU, that priorizes low power consumption over performance. Mobile GPUs use various on-chip caching techniquest to reduce the memory traffic, which is a requirement for energy-efficient operations (Tarkoma et al., 2014).

The display is yet another of the most energy-consuming components in mobile devices, they now include touch-sensitive full HD screens. It is only surpassed by the wireless network interfaces (WNI), where the energy

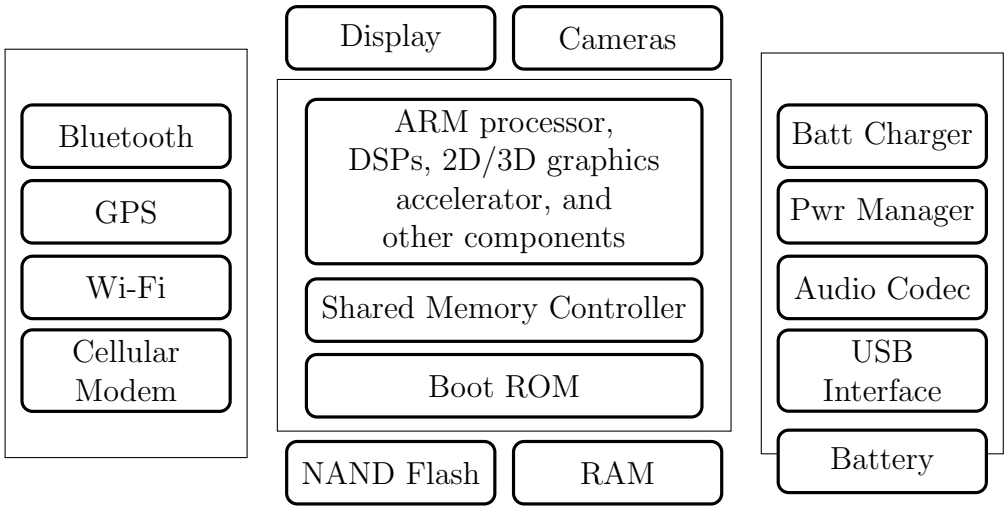


Figure 3.2: Usual hardware structure of mobile devices. Adapted from (Tarkoma et al., 2014)

consumption depends highly on the type of access network used and the workload offered by the applications. Interfaces such as Wi-Fi and Bluetooth have a dynamic power demand, and usually have three states: idle, transmitting and receiving which can be categorized in two different duty cycles, idle and active. The power consumption on the active state is significantly higher than on the idle state. The overall power can be optimized by making the active states shorter, so mobile devices could spend most of their time in idle states (Tarkoma et al., 2014) consuming less energy. Some experiments presented by (Pathak et al., 2012) shows that most of the applications spend a large amount of energy in I/O components such as Wi-Fi, 3G and GPS.

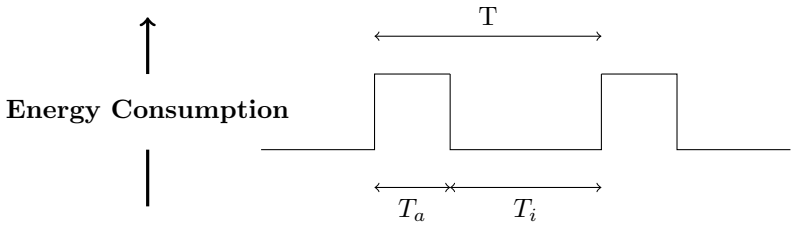


Figure 3.3: Example of a duty cycle

Figure 3.3 shows a total duration of a duty cycle, where T_a is the active state, and T_i the idle state. Moreover, for each data transfer there is a waste of energy produced by the *ramp energy* and the *tail energy*. Ramp energy is referred to the energy required to switch to the active state, while tail energy is a waste of energy (where the interface remains active during a certain amount of time) after the completion of a transfer. Tail and ramp energies are constants that mitigates the waste over larger transfer sizes or frequent successive transfers. For example, with the tail energy, the interfaces avoid a

change of state between close transmissions. Tail energy has different duration times depending on the type of connection (i.e. WiFi, 2G/3G/4G). GSM (2G) has a much smaller *tail time* compared to 3G and WiFi (6 secs vs 12 secs). However, the data transfer in WiFi is significantly more efficient than the other connections (Balasubramanian et al., 2009).

Another high-energy consuming activity in Wi-Fi and Bluetooth is the discovery phase. While in Wi-Fi the discovery looks for Access Points (AP), in Bluetooth it looks for other bluetooth devices. Moreover, Bluetooth Low Energy (BLE) was introduced as a new standard for the version 4.0 of Bluetooth, allowing low rate and very low-power communications. Bluetooth was originally designed for continuous streaming data applications, meanwhile BLE limits the transmissions to small but frequent packages (31 bytes for application payload data, once per second or minute). However, Bluetooth's discovery consumes only slightly more energy than BLE discovery, and BLE connection events seems to draw a high amount of energy, even more than its discovery. BLE communication takes place between a slave and a master device. For all data transfers in BLE, both the slave and the master have to wake up in synchrony to exchange messages in order to consume less energy (Tarkoma et al., 2014).

All these hardware components are feed with energy by Lithium-ion batteries, which have become popular due to their thinness, lightness, and efficiency. But these batteries haven't evolved as much as the other components and have not kept up with Moore's Law. This is due to the chemical nature of batteries and the fact that there are theoretical limits on the amount of energy that can be obtained from the materials (Tarkoma et al., 2014). We have introduced the energy-consuming components in mobile devices, however we will only focus in the WNI, since it is the most energy-hungry activity in IoT-gateways.

3.3 Energy-Saving Strategy

Developing energy-efficient mobile applications is a very challenging task. Different from desktop and laptop computers, where the CPU is likely to be the component that consumes more energy (often more than 60 percent of the total power), in mobile devices there are several components that may consume nearly the same or even more energy than the CPU. For example, an often cited power budget mobile phone streaming a 384 kb/s video includes 1.2 W power drain caused by the WNI, 1 W for the display, and 0.8 W for the CPU and memory operations (Tarkoma et al., 2014). Besides, nowadays

mobile devices possess several sensors that also consume a considerable amount of energy, such as GPS, accelerometer, and gyroscope. In this dissertation we focus on the energy consumption produced by the network interfaces (i.e. Internet communication).

Experiments in (Li e Halfond, 2014), related to the network interface, showed that it is more efficient to send bigger size packages over smaller ones, since this reduces the active state duration of the duty cycle. As an example the energy consumption for downloading 1,000 bytes of data is almost the same as downloading 1 byte. Moreover, they also showed that high memory usage will increase the energy consumption only by a small amount. In their experiments, the average energy consumption per computational step increased 21% for each memory increase of 10,000%. So, buffering several messages to decrease the number of active states during communication, could produce a relevant decrease in the energy consumption depending on the frequency of messages.

Internet A/V streaming is an interesting case in point. If MP3-encoded music is streamed to a mobile device, its network interface must remain active during the entire session. If the user first downloads the songs and then listens to them; during playback the network interface does not have to remain active, which can reduce energy consumption significantly (Pentikousis, 2010). Moreover, cellular networks consume much more energy than Wi-Fi according to (Balasubramanian et al., 2009), because they remain more time in active states.

In this context, with the purpose to prevent the wireless network interfaces (WNI) to remain in prolonged active states, CEP can pre-process all the data collected by the M-OBJs. Thus, the mobile device will only communicate with the cloud when a complex event is detected. CEP queries may not necessarily reduce the amount of transferred data, but instead require a low outbound frequency, since the main objective is to increase the idle states of the WNI. Moreover, it is important to use a monitor to estimate the improvements, usually power values are reported as averages over a specified time window. In order to target the subsystem of interest, we have to turn off all the other features. The challenge is that mobile devices consist of many subsystems that are constantly under interaction.

4
The Mobile Hub

The envisioned architecture for IoT has changed a lot during the last years. Now we are aware that simpler Things or M-OBJs cannot and will not communicate directly to the cloud because they are low-powered devices and Internet communication is an energy-hungry activity. Hence, the architecture illustrated in Figure 4.1 has been proposed by most of the works in the literature. In such architecture, in order to deliver the information gathered by the perception layer (Level 0) to the application layer (Level 2) in the cloud there is a mid-layer for transmission (Level 1). This transmission layer is formed by mid-powered devices called IoT-gateways (e.g. routers, mobile devices) that communicate with the M-OBJs and the cloud using WPAN (e.g. BLE and ANT+) and WLAN (e.g. WiFi) technologies respectively.

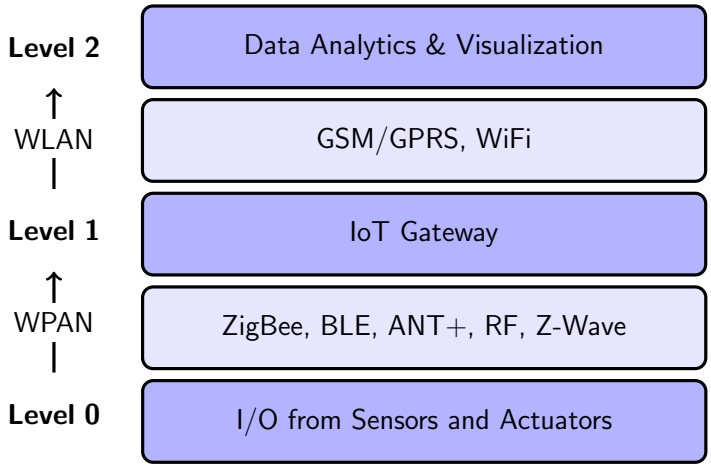


Figure 4.1: Typical IoT Architecture

Moreover, since mobile devices are becoming ubiquitous, cheaper and more powerful, and that disconnections and message loss will be the norm in IoT, where reliable delivery of single messages from/to M-OBJs will be less important, mobile devices are the natural candidates for serving as IoT propagator nodes. This makes us propose the concept of Mobile Hub (M-Hub) as the intermediate between the myriad of different Smart M-OBJs and the long-haul Internet connection. In this regard, IoT-gateways have a high energy consumption for their behaviour as a merely bridge for the data. Thus, we propose a modification to the current IoT architecture by including Fog computing, which allows to move part of the processing from the application layer to the edges of the network. This chapter introduces our two main proposals for communication and local processing.

4.1 Communication Infrastructure

The M-Hub is a general-purpose middleware that enables mobile personal devices to become the propagator nodes (i.e. gateways to the Internet) for simpler IoT objects with only short-range WPAN interfaces. This middleware is responsible for discovering and opportunistically connecting many different simple M-OBJs to the Internet in order to be able to "bridge the gap" between the Internet connection to the cloud and the short-range wireless connections established with M-OBJs. Furthermore, the M-Hub provides context information like the current local time and/or the (approximate) location to the data obtained from the M-OBJs to which it is connected (Talavera et al., 2015). This feature opens up to IoT applications new ways of classifying, filtering or searching data gathered from the M-OBJs. The main mandatory tasks of the M-Hub thus include:

Discovery, monitoring and registration of nearby M-OBJs periodically, the M-Hub scans for nearby M-OBJs announcing their IDs and capabilities. This information about reachable M-OBJs is kept stored in the M-Hub database and eventually forwarded to some service executing in the SDDL core.

Connecting to a M-OBJ depending on the kind of interaction (and the WPAN technology capabilities) a stable communication link may be established with some M-OBJ, over which the M-Hub executes a request-reply protocol.

Protocol Transcoding Data packets and messages from/to M-OBJs may have different formats and encodings. Thus, the M-Hub transcodes them from/to serialized objects, encoded with Protocol Buffers, and transmitted over the MR-UDP connection.

Caching of recent M-OBJ sensor/status information in order to optimize communication over the Mobile Internet, the M-Hub may group several pieces of sensor data from the set of nearby M-OBJs into a single "bulk message" for transmission. In order to do this it stores the most recent (the current) data items obtained from the M-OBJs.

Sending requests to M-OBJs depending on the services/resources offered by the M-OBJ, the M-Hub periodically or sporadically send queries about sensor readings and/or the M-OBJ's current state (e.g. read the current temperature value).

4.1.1 Main Components

The M-Hub consists of four services and one manager executing in background. The **S2PA Service** is responsible of discovering and monitoring nearby M-OBJs that use the supported WPAN technologies. This service keeps a record of the current provided sensors/actuators (e.g. temperature, accelerometer) and publish the sensed information to all the components that require it. One of which is the **Connection Service**, where messages are sent to/from the cloud in a JSON format through an Internet connection. Important messages (e.g. M-OBJ connection/disconnection) are sent immediately to the cloud, while sensor data or low relevance messages (e.g. temperature readings) are grouped, to be transmitted as a bulk message at periodic time intervals.

Messages that are going to be send to the cloud are enriched with context information, like a timestamp and the approximate location. The location is obtained through the **Location Service**, responsible for sampling the M-Hub's current position obtained from different providers like GPS or a manually entered (in case of a fixed location). The periodicity and duration of all of these three service's actions, are influenced by the device's current energy level (LOW, MEDIUM, HIGH), and is set by the **Energy Manager**, which from time to time samples the device's battery level and checks if it is connected to a power source.

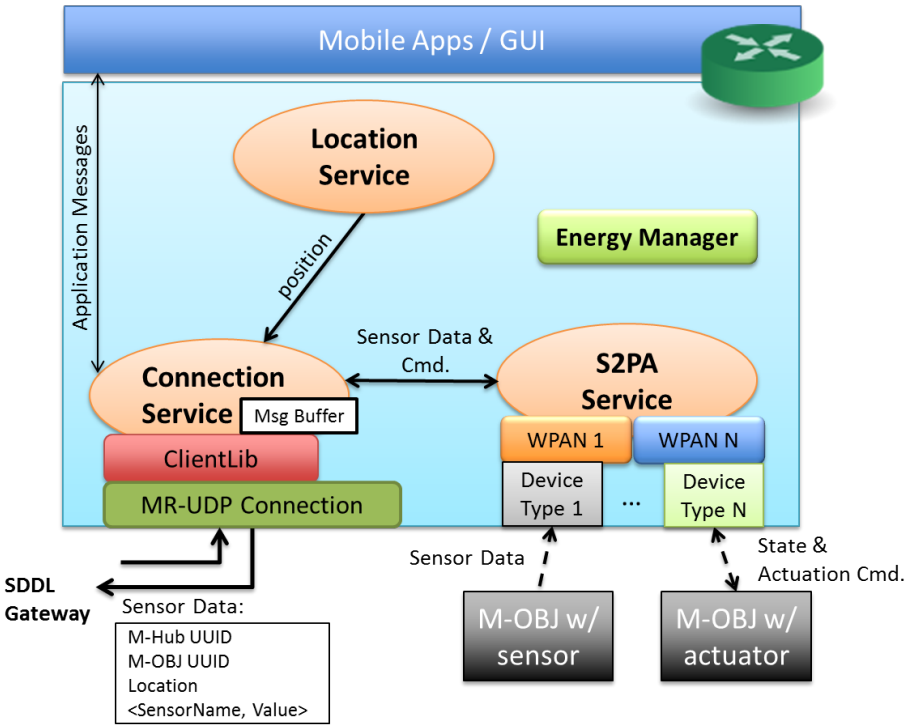


Figure 4.2: Architecture of the Mobile Hub

Figure 4.2 shows the M-Hub architecture and some details of the interactions between the aforementioned components. The communication among all the services is done using an EventBus[1], which is a Publish-Subscribe (Pub-Sub) event bus optimized for Android that helps to decouple the components, and allows the inclusion of different services without many code modifications. *Device Type 1, ..., Device Type N* are modules that handle the information received/sent from/to specific M-OBJs, since each different M-OBJ could possess a different method to transform the raw data (bytes array) to readable information (doubles array). Hence, each of the modules possess a **convert()** method that handles such conversion.

The S2PA Service is capable of managing several WPAN technologies by calling generic methods, that are mapped to their respective classes and methods. As soon as a new M-OBJ is detected, it starts getting data from the it, which are then handed over to the *Connection Service*, where they are stored in a buffer (*msg buffer*). The Energy Manager controls the periodicity intervals for all the three Services, depending on the current battery level, and if the M-Hub is plugged or not to the power supply. For example, if energy supply is *LOW*, *MEDIUM* or *HIGH* the S2PA Service will perform the scan (on all WPAN technologies) every 20, 30 and 40 seconds, respectively. These thresholds are configurable for different mobile devices.

4.1.2 Short-Range Sensor, Presence and Actuation API

One of the main purposes of the M-Hub is to handle M-OBJs with different WPAN technologies (e.g. BLE, Classic Bluetooth). To this end, it has a protocol for short-range communication with M-OBJs, based on two interfaces that help developers implement the different technologies uniformly. On the one hand, the *Technology Interface* (shown in Figure 4.3), maps the main capabilities of each WPAN technology to some methods that have to perform the following actions: 1) Discovery of, and connection to M-OBJs, 2) Discovery of services provided by each M-OBJ, 3) Read and write of service attributes (e.g., sensor values, and actuator commands) and 4) Notifications about disconnection of M-OBJs. And on the other hand, the *Technology Listener Interface* is implemented by the S2PA Service to listen to all the important events such as the sensor data, detected by the M-OBJs and reported by each technology in order to publish them for the subscribed components. The data published has the structure showed in the Figure 4.4.

[1]http://greenrobot.github.io/EventBus

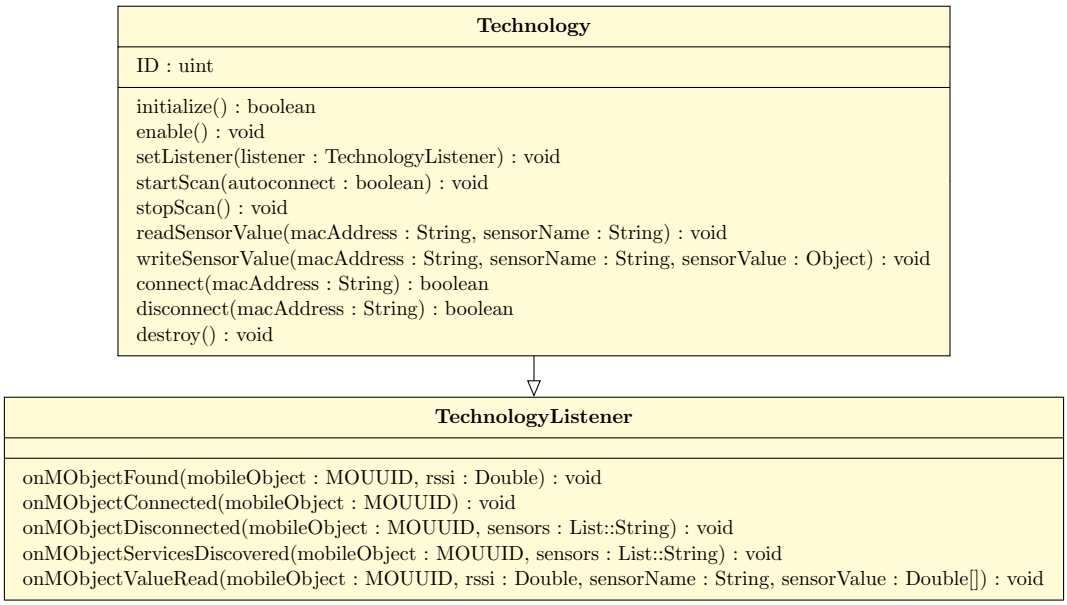


Figure 4.3: Main interfaces used by the M-Hub to communicate with M-OBJs.

Some functions in Figure 4.3 that need to be explained are the followings: **initialize,** verifies if the technology exists on the device and sets it up in order to be used, **readSensorValue,** and **writeSensorValue,** request a read or a write of a sensor respectively, where the *macAdrress* is the unique identifier of the M-OBJ under the technology and the *serviceName* represents the sensor with a string (e.g. “Temperature” and “Humidity”). In case of a write there is an extra parameter sensorValue that will handle commands to actuators.

Moreover, the developers have to include an ID in the different Technology classes to uniquely identify them. This ID is also combined with the M-OBJ’s MAC address to form the Mobile Object Universally Unique Identifier (MOUUID) which help developers to differentiate all the M-OBJs, even if they are communicating with the M-Hub under different WPAN technologies. The current M-Hub implementation only has a module for Bluetooth Low Energy (a.k.a. *Bluetooth Smart - BLE*) defined with the ID 1, and a MOUUID under such technology could be 1-B4994C64BA9F.

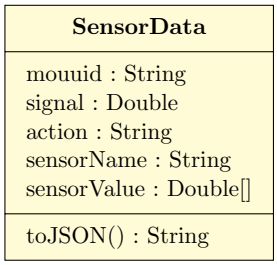


Figure 4.4: Basic sensor data structure of the M-Hub.

Figure 4.4 represents the data received from the M-OBJs, the mouuid identifies uniquely the M-OBJ that produced the sensor data. The signal represents the connection strength of the M-OBJ (e.g. Bluetooth uses dBm[2]) with the M-Hub. The action indicates the type of data in relation with the M-OBJ (i.e. found, connected, disconnected, read), while the sensorName is an optional value that gives a higher description of the data (e.g. temperature, humidity). Finally, the sensorValue only appears when the sensorName is present, and contains the data values (e.g. 25 for the temperature, or [0.5, 0.0, 0.5] for the accelerometer).

4.2 Local Processing Infrastructure

Differently from the traditional Internet where data consumption is primarily discrete (Billet e Issarny, 2014), IoT deals with a continuous processing of data produced by sensors and actuators embedded into mobile devices and M-OBJs. In IoT data usually have value at the their generation time, given the need for fast decision making systems (e.g. Smart Cities, and Ambient Assisted Living), and to reduce bandwidth consumption in order to avoid bottlenecks, IoT should completely decentralize its processing. However, most of the current IoT-gateways only care about the direct transmission of the data acquired from the physical environment (perception layer) to the cloud (Pereira et al., 2013) yielding an architecture with a high cost for data processing.

In this regard, an emerging category of applications are based on the collection of sensing data at a community-wide level, where multiple individuals provide sensing data in order to contribute to the observation of a large scale phenomena (e.g. traffic congestion, environmental pollution) (Skorin-Kapov et al., 2014). Current mobile devices have enough processing power to perform part of the IoT data processing close to the data sources (M-OBJs). Furthermore, communication with the cloud is an expensive operation in terms of energy and network bandwidth consumption which may critically affect the scalability and the sustainability required by the IoT. Thus, a recurrent IoT concern is to reduce the amount of information being sent to the cloud, by detecting and sending only consolidated and pre-processed data that actually matters to applications, like for example a sudden increase in the temperature, or an elevated heart rate.

[2]https://en.wikipedia.org/wiki/DBm

4.2.1 MEPA Service

The Mobile Event Processing Agent (MEPA) is an aditional service that extends the Mobile Hub (M-Hub) with the capacitity of local processing of the data received from the Mobile Objects (M-OBJs) (see Figure 4.6). We believe that Complex Event Processing (CEP) can be used for evaluating the streams of sensor data (looking for certain data patterns) close to the sources. Raw sensor data usually carries only little semantic information, and hence it needs to be correlated and enriched to gain some meaning (Dunkel et al., 2013). Moreover, CEP exhibits characteristics that makes it well suited for processing in mobile devices: it employs in-memory processing which allows (near) real-time operations, and also the ability to correlate heterogeneous data (sensor data) obtained from nearby M-OBJs.

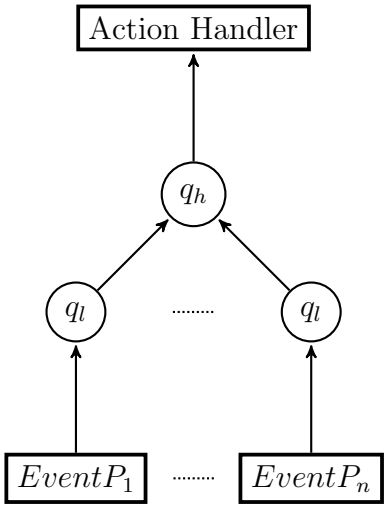


Figure 4.5: Distributed CEP scenario

As an example, Figure 4.5 shows a scenario where q_l are simple correlations (CEP queries) executed in different mobile devices. Such queries process the sensor data received from $EventP_i$ (Event Producers), which could be different M-OBJs (sensors) located in the proximity. The events generated from q_l are sent to a computational-intensive processing CEP query q_h in the cloud, where they are combined producing events that can start different actions (e.g. send a notification, turn on the air conditioner). Such distributed workflow of CEP queries can be used in similar IoT processing applications where it is possible and desired to distribute the processing across the cloud and the edges, in order to scale a system (e.g. smart cities, ambient assisted living).

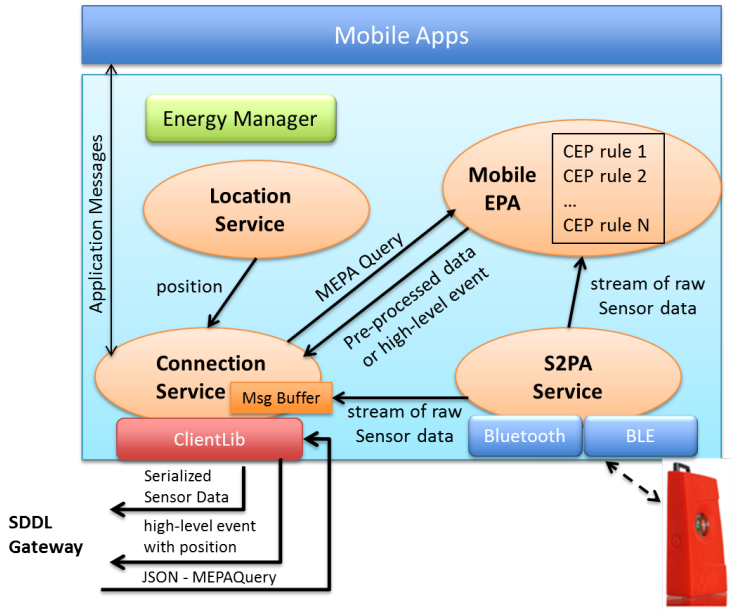


Figure 4.6: Architecture of the M-Hub with the MEPA Service

4.2.2 Dynamic Deployment of CEP Queries

M-Hubs can opportunistically collect data in different environments (e.g. a hospital, a school, the streets) at different moments. Such environments are very likely to have diverse sets of M-OBJs, which provide sensor data to complete different applications. For example, imagine a set of M-OBJs (e.g. sensors for temperature, smoke, motion) distributed around a house to provide information about its current state. When a M-Hub enters to the house, it will need to swap the executing CEP queries to the ones required to process the corresponding sensor data (e.g. detect motion when no one is supposed to be home or a gas leak). A completely different scenario could be described for sensors located in the street or the beach that could help to detect high pollution/radiation levels, or the average environmental temperature. In fact, in certain situations it will need to stop all the executing CEP queries since the sensor data is very infrequent or not necessary at all.

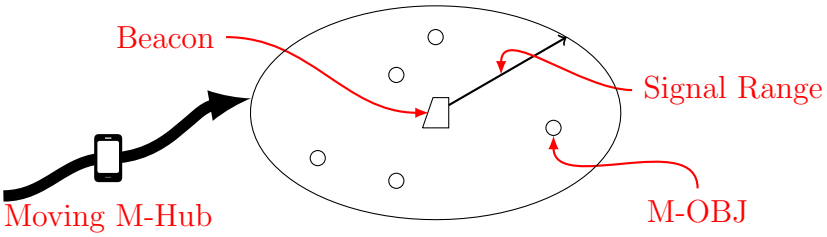


Figure 4.7: Environment defined by a beacon

To identify different environments, we could use for example the GPS position or Beacons[3] (see Figure 4.7). Hence, when a M-Hub enters in the range of a Beacon, or in a predefined area over a GPS position, the installed CEP queries should be swapped with the corresponding ones to the available sensors in the device's vicinity. These sensors may vary as the M-Hub is connected with different M-OBJs during its lifetime and in different situations. Due to this flexibility requirement we included the capability to add/remove CEP queries to/from the M-Hub remotely using the communication link with the cloud.

On the one hand, when processing in the cloud, there is virtually an unlimited processing and energy capacity. Centralized-cloud processing has the ability to analyze collectively data obtained from different mobile devices, and with a low latency of processing since raw sensor data are received as soon as they are collected (depending on the amount of sensor data and available bandwidth). On the other hand, when we include a local pre-processing and bulk transmission in the mobile device, energy is saved (except for very complex computations), there are less probabilities of overcrowd the bandwidth, and the CEP queries can be customized to the types of sensors that are in the vicinity of the mobile devices.

4.2.3 Implementation

In order to enable CEP processing in mobile devices, we have included the Asper CEP engine (Eggum, 2014) to the new MEPA service to process any incoming sensor data. Additionally, to allow the creation of CEP queries to process such data we needed to define a primitive data type. So, for such primitive (event type) we have used the SensorData structure defined in the section Communication Infrastructure (see Figure 4.4), which provides an identifier for the M-OBJ, the sensor's name (e.g. temperature, humidity) as a string and its respective values as a array of doubles.

The MEPA Service keeps a record of the running CEP queries, and the representations of the events as tuples of named values (event type). These records allow to start and stop CEP queries on-the-fly, and to restore them even if a sudden shutdown occurs. As shown in the Figure 4.6, the MEPA Service subscribes to all the messages that are sent from the S2PA and the Connection services, since the former collects the data from the M-OBJs, and the latter receives the commands from the cloud to modify its behavior (e.g. deploy/remove a new CEP query).

[3]Device that advertises its position in a fixed region - https://es.wikipedia.org/wiki/IBeacon

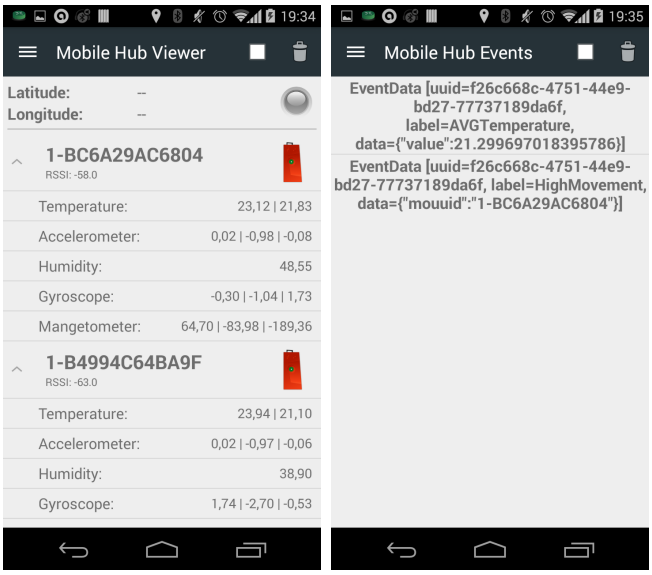


Figure 4.8: Screenshots of the M-Hub viewer for sensor data and events

The *MEPAListener* is a class that implements the notification interface of Asper (UpdateListener). It can initiate an action whenever new events become available. Each one of the CEP queries installed in the MEPA Service require an instance of the MEPAListener in order to listen and react to the detected event patters (which leads to the generation of a new complex event). The MEPAListener may also publishes the complex events (an Event structure that contains the data and the identification label) to any interested component that could be the Connection service, or another CEP query executing in the MEPA Service. For example, Figure 4.8 shows some detected events in a list view (right image), obtained from the M-OBJs in the left image.

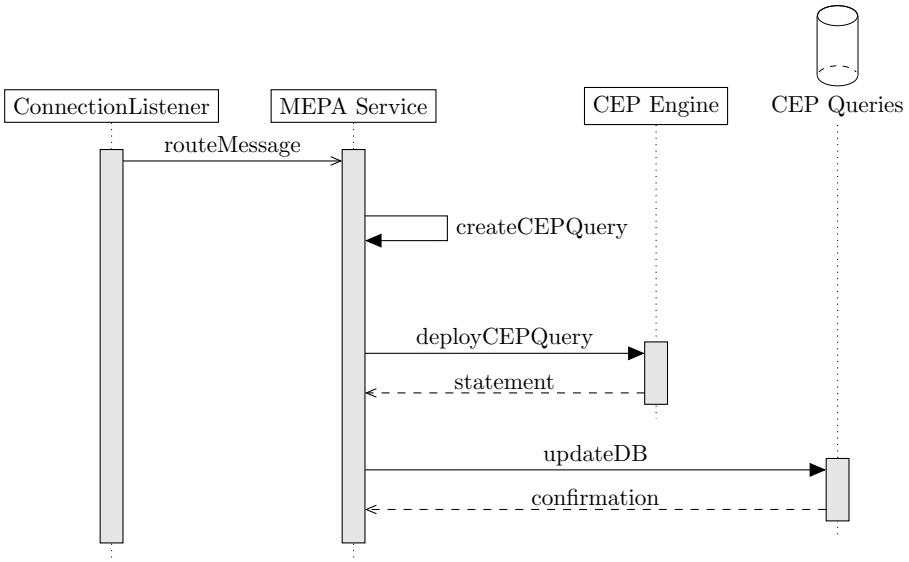


Figure 4.9: Sequence diagram of the deployment of CEP Queries

A software framework implemented around the MEPA Service translates the messages (JSON encoded) received through the Connection Service into specific commands to modify the CEP queries executing in the M-Hub. Figure 4.9 shows the process for the deployment of CEP queries. First messages are received by the *ConnectionListener*, to later be parsed and routed (published) to the *MEPA Service*. Once in the MEPA Service, all the fields are verified before deploying the query in the *CEP Engine*. Finally, the database (*CEP Queries*) is updated with the new query. It is important to keep a record of the CEP queries in order to be able to modify or reload them when necessary (e.g. service reboot).

Code 4.1: A MEPARequest structure

```
{
        "MEPARequest": {
                "type"  : "add",
                "label" : "HighTemperature",
                "target": "global"
                "query" : {
                        "SELECT sensorValue[1] as
                           value FROM SensorData(
                           sensorName='Temperature')
                           WHERE sensorValue[1] > 25"
                }
        }
}
```

The commands are encapsulated as ***MEPARequest*** objects that contain the minimum information that we believe is necessary for this process. The **type** of the request (i.e. add/remove/start/stop/clear), and a text **label** used to identify the Event Processing Language (EPL) queries. In the case of an `add` request type, the MEPARequest will also include a field **query** for the EPL query (encoded as a string). The CEP query is always assumed to be correct, so the MEPARequest is always sent to the M-Hub, see the Code 4.1. However, wrong query specifications or any other exception that could arise (e.g. incorrect syntax in the CEP query, undefined label), generate an error message in the MEPA Service that is sent to the server with a description of the problem.

Code 4.2: An event message

```
{
        "uuid"     : "f26c668c-4751-44e9-bd27-77...",
        "label"    : "HighTemperature",
        "latitude" : -22.978823,
        "longitude": -43.233249,
        "tag"      : "EventData",
        "timestamp": 1456250590,
        "data"     : {
                "value": 21.299697018395786
        }
}
```

The outbound events of the CEP queries (complex events) can be sent to the cloud or to another query executing in the local MEPA Service. The previous characteristic arises for the necessity of using complex events as input for different CEP queries that don't need to be sent to the cloud. So, in order to define the destination of the generated event, the value **target** (shown in Code 4.1) might be set either to local or global. If it is set as local, the event will be received by the CEP engine in the MEPA Service, and as global it will be sent to the cloud.

Moreover, the complex events are sent in a JSON format with a data structure similar to the one presented in the Code 4.2. The *uuid* uniquely identifies the M-Hub, the *tag* indicates that it is an EventData structure and the *label* relates the event with its query. The *latitude* and the *longitude* are optional values, the timestamp is in an Epoch format[4], and finally, the *data* contains all the values obtained from the complex event (defined in the CEP query). As an example, the code in 4.2 shows an outbound event from the CEP query in 4.1 that explicitly selects the $sensorValue[1]$ with the alias "value".

4.3 Summary

One of the most (if not the most) important requirements of IoT is to bridge the gap between the things and the cloud. Hence, we propose the concept of the Mobile Hub (M-Hub) to provide Internet connectivity to nearby Mobile Objects (M-OBJ). While the former can be very simple sensor or actuator devices with no significant processing and storage capacity, the latter are resource-full portable personal devices (smart phones or tablets).

[4]https://en.wikipedia.org/wiki/Unix_time

Several IoT projects are starting to use mobile devices as opportunistic sensor data collectors for the cloud, given their current capabilities (e.g. Internet communication, WPAN technologies). However, energy stored in mobile devices batteries is very limited and the activity that drains more of such energy is the Internet communication. In this regard, sending only pre-processed data to the cloud could reduce the use of the wireless network interfaces, and hence mitigate the energy/bandwidth consumption in mobile devices. CEP is a widely used technology used for processing of continuous streams of data in (near) real time, and that have been proved can be adapted to Android. In section 4.2.1 we explained the advantages of mobile CEP, and gave an example of its use.

Moreover, mobile devices are usually in constant movement and can be located in several environments that provide different types of sensor data. Depending in the sensor types, CEP queries executing in the IoT-gateways should be dynamically swapped with the corresponding ones. Section 4.2.2 explained the advantages and limitations of including a dynamic modification of the local CEP queries. Finally, Section 4.2.3 presented an explanation about our architectural decisions for the implementation of the MEPA Service as a new service for the IoT-gatewats.

5
Performance Experiments and Results

This chapter describes the experiments and measurements realized that aims to answer the research questions of section 1.1. Our intention is to demonstrate that by executing Complex Event Processing (CEP) in the edge devices (mobile devices), it is possible to reduce the communication with the cloud, and thus, save energy/bandwidth in mobile devices. To this end, we tested a prototype running different kinds of CEP queries to measure the number of actions the M-Hub can perform until the battery level decreases by 1%, as well as the total bandwidth usage. Section 5.1 details the setup for our experiments, followed by a presentation of each individual query, while Section 5.6 discusses the results.

5.1
Experimental Setup

For all the experiments we used a Motorola Moto X handheld (model 2013) running Android 4.4.4 KitKat (see Table 5.1) with a good battery integrity as the M-Hub. Both Android runtimes were tested, Dalvik and ART[1]. The devices used as M-OBJs were off-the-shelf SensorTags[2]. Each one of the Sensor Tags has six sensors with different update times: Humidity (>100ms), Temperature (>250ms), Gyroscope (>0.125ms), Accelerometer (>20ms), Magnetometer (>12ms), and Barometric pressure (>2ms).

The WPAN technology used for the communication with the M-OBJs was Bluetooth Low Energy (BLE). It has a characteristic called notifications that allows to get the values from the connected sensors as soon as they are updated. The experiments were made with up to six M-OBJs, since BLE in Android limits the simultaneously connections and notifications to that number. The average size of the messages was 200 bytes for events, and 300 bytes for sensor data. Such sizes could vary depending on the type of sensor or event. However, as explained in Chapter 3, the energy consumption is not very related to the size of the transferred data. The notebook used to run the SDDL Core (Gateway, Query Manager and Web Monitor) for all the experiments was an ASUS Intel(R) Core(TM) i7-4500U CPU 1.80GHz with 5857 MB of RAM, running Arch Linux (Kernel 4.1.11-1-lts). All the tests were executed using a WiFi (IEEE 802.11bgn) connection.

[1]ART and Dalvik - https://source.android.com/devices/tech/dalvik/
[2]Texas Instruments CC2541 Sensor Tag - http://www.ti.com/lit/ml/swru324b/swru324b.pdf

Component	Description
Chipset	Qualcomm MSM8960DT Snapdragon S4 Pro
CPU/GPU	Dual-core 1.7 GHz Krait 300 / Adreno 320
Memory	2 GB RAM
Kernel	3.4.42-g50861a7
Battery	Non-removable Li-Ion 2200 mAh battery

Table 5.1: Moto X Specifications

In order to quickly connect with the M-OBJs and avoid that other components affect the battery, the M-Hub was configured to not use the Location Service, perform a WPAN scan every three seconds, and the scan duration was set at two seconds. For the experiments without CEP processing, the time interval in between consecutive sending of sensor data to the cloud was set to 100ms. Moreover, BLE requires a Service Discovery phase to retrieve the services that the M-OBJs provide, however such time could be quite long for the first connection depending on the amount of sensors/actuators that a device possess. Hence, for our experiments, we skipped this first connection and started testing from the re-connections, to avoid an idle time on the M-Hub that will only reduce the amount of messages processed and sent to the cloud. Further tests that reflect the time that it takes to the M-Hub to connect with M-OBJs using BLE can be found in (Talavera et al., 2015).

Our main concern is the energy and bandwidth consumption caused by the CEP processing and Internet communication in mobile devices. Thus, we conducted a series of experiments deploying different CEP queries with different kinds of processing such as filtering, pattern match, and aggregation. Each test sampled the processing and the cloud communication for one hour with different sets of connected M-OBJs (1, 3, and 6). In order to find out the energy and bandwidth consumption of the mobile device, we built a tool to measure the bandwidth usage (Kb and packets) per application and the mean time that it takes a battery percentage to decrease. Most of the queries included time windows (jumping and sliding), in those cases the length of the windows was set to 10 seconds and 1 minute.

It is also important to mention that some external factors that we couldn't control were present during the experiments, such as some disconnections of the M-OBJs since BLE is still unstable in Android, and some processing or Internet communications from other applications that slightly affected the battery life (e.g. system apps). Most of the other applications in the smartphone were uninstalled with exception of the stock ones which were deactivated. Nevertheless, it was possible to relate the outcome between the observed values to draw strong conclusions.

Table 5.2: No processing - Energy and bandwidth consumption

Dalvik	1	3	6
Bandwidth (kb)	7,108	20,635	42,204
Mean time (s)	578.00	535.83	516.17
Std. Deviation	34.77	28.48	71.94
ART	1	3	6
Bandwidth (kb)	7,262	19,551	37,440
Mean time (s)	727.00	629.39	551.60
Std. Deviation	21.85	48.18	13.35

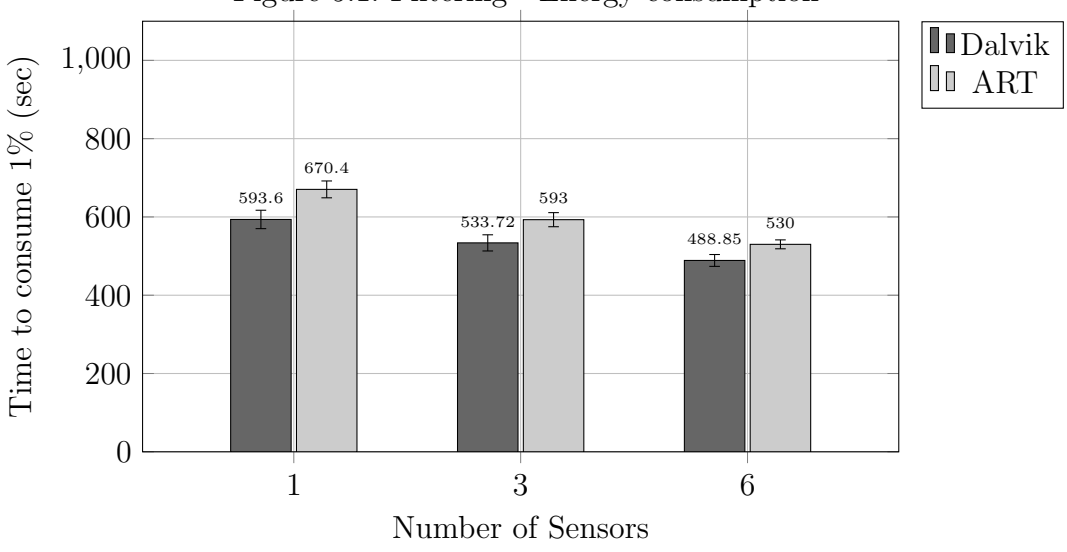
5.2 Filtering Query

We tested how much energy and bandwidth we could save with a simple CEP filtering query. Hence, we created the query 5.1 that filtered the information to just temperature data, but didn't do any further processing. By filtering data we can successfully reduce the amount of transferred information, reducing the possibility of a bottleneck. However, it doesn't reduce the amount of active states of the network interfaces.

Code 5.1: Filtering CEP Query

```
1 SELECT * FROM SensorData(sensorName='Temperature');
```

Figure 5.1: Filtering - Energy consumption



As we can see in Figure 5.1 and Table 5.3, even when we reduced the total bandwidth usage to almost a factor of three, the energy consumption is almost the same as sending all the information to the cloud (see Table 5.2). Thus, if we can't reduce the number of active states for communication, we won't be able to see big reductions in the energy consumption.

Table 5.3: Filtering - Bandwidth consumption

No. of Sensors	Dalvik (kb)	ART (kb)
1	3,146	3,219
3	7,178	7,045
6	12,682	12,865

5.3 Aggregation Queries

In these experiments we intend to show the energy and bandwidth consumption with some CEP queries that included other processing than just filtering, and used different window types (i.e. sliding and jumping). The scope of a stream is called a *window* and defines the lower and upper bounds of the information that is currently seen. Jumping windows wait until their size n is filled with events to only then process all data items in a bulk operation.

Sliding windows have lower and upper bounds that advance with time or as data items are inserted into the engine. Every time these windows vary, they process all the data that they contain within their bounds. Their main difference is that jumping windows will never contain a data item that could have resided in the window before it, while sliding windows exchange the oldest event with the newest event that arrived (Eggum, 2014). Both window types can be classified as time- and count- based. Count-based windows group events by quantity (e.g. 10 events). In order to find out the relation of the energy consumption with different window sizes, we used time-based windows for the tests with 10 seconds and 1 minute of length.

5.3.1 Average Query

The following CEP queries filter the sensor data to process their average value using sliding and jumping windows. Only events that contained the average value were sent to the cloud.

Jumping Windows

The query in Code 5.2 processes the average value by using a jumping window of ten seconds, while the query in Code 5.3 uses a jumping window of one minute.

Code 5.2: Average CEP Queries (Jumping 10 seconds)

```
1 SELECT avg(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time_batch(10 sec);
```

Code 5.3: Average CEP Queries (Jumping 1 minute)

```
1 SELECT avg(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time_batch(1 minute);
```

Figure 5.2: Average CEP Query (Jumping 10 seconds) - Energy consumption

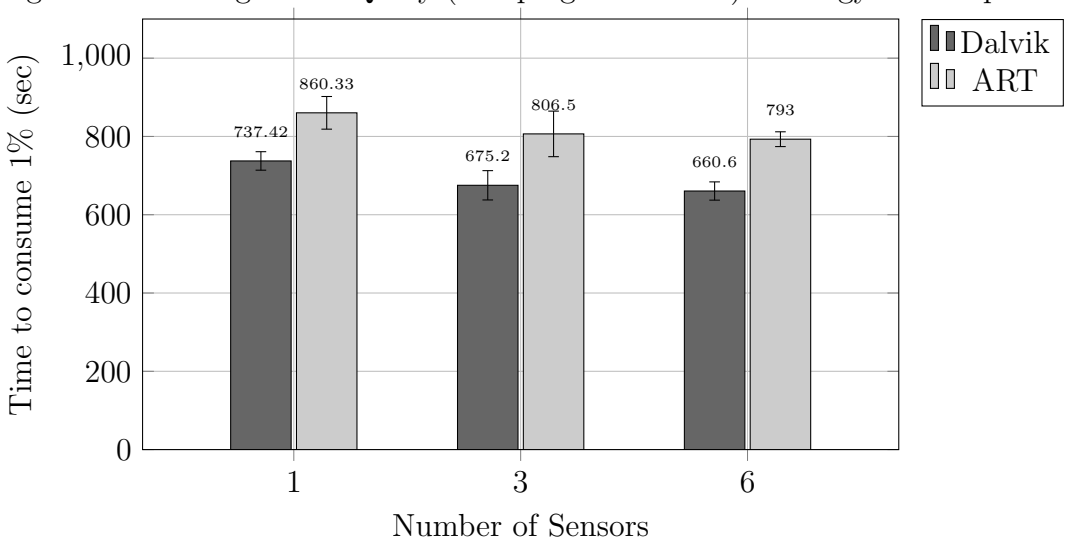


Figure 5.3: Average CEP Query (Jumping 1 minute) - Energy consumption

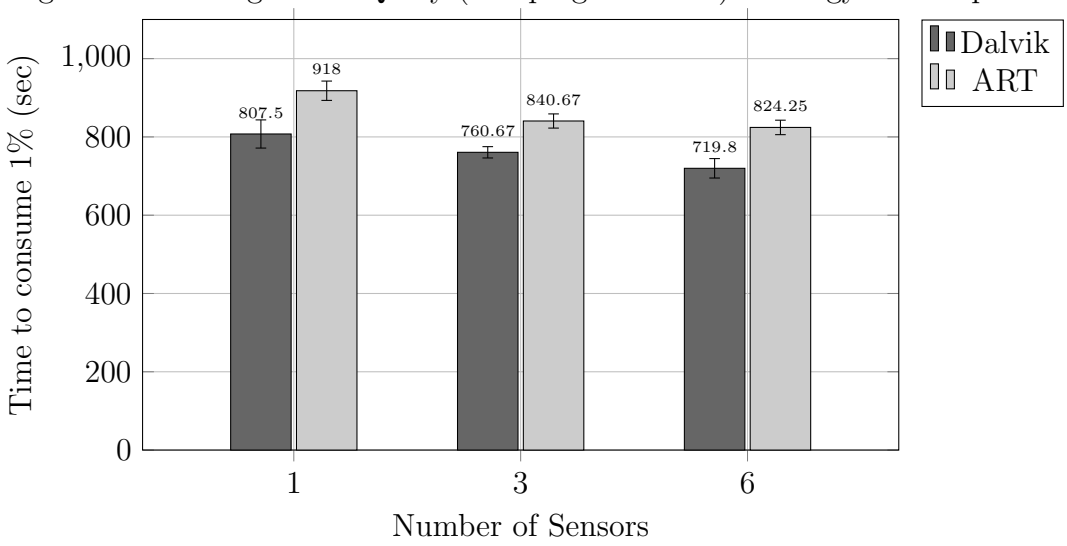


Table 5.4: Average CEP Query (Jumping) - Bandwidth consumption

10 seconds	1	3	6
Dalvik (kb)	267	320	250
ART (kb)	254	312	268
1 minute	1	3	6
Dalvik (kb)	71	102	69
ART (kb)	81	105	67

Figures 5.2 and 5.3 plot the energy consumption results showing an improvement in comparison with the previous experiments, in special with the results in Table 5.2 where there is no processing involved. Moreover, Table 5.4 also presents a high reduction in the bandwidth consumption. Thus, we imply that since bigger jumping windows reduce the frequency of outbound events, they could also increase the idle time of the network interfaces. If the frequency is low enough, it is possible to greatly reduce the energy/bandwidth consumption.

Sliding Windows

Queries in the Codes 5.4 and 5.5 use the same processing logic as the queries in Section 5.3.1 but using sliding windows instead of jumping windows.

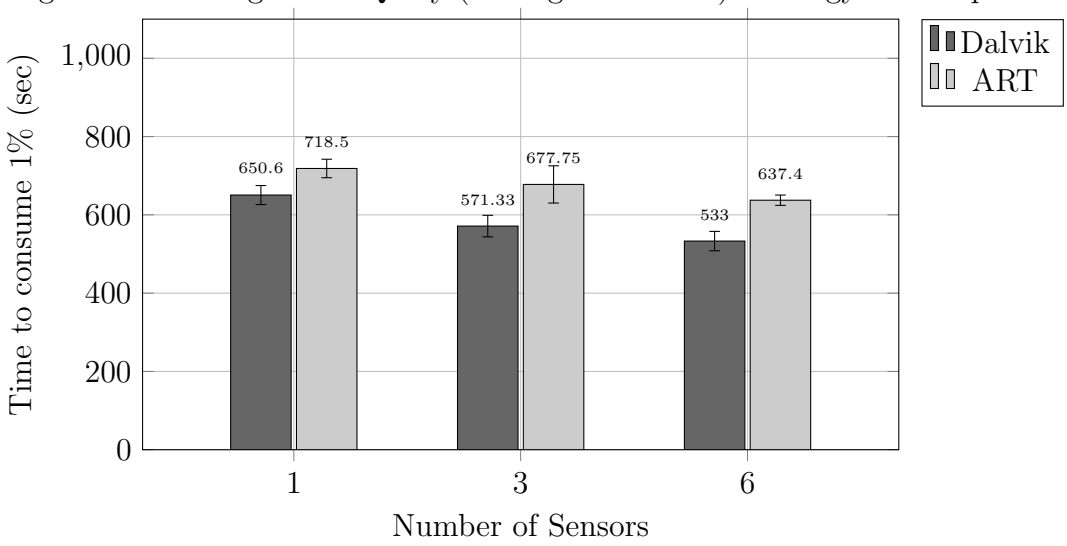
Code 5.4: Average CEP Queries (Sliding 10 seconds)

```
1 SELECT avg(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time(10 sec);
```

Code 5.5: Average CEP Queries (Sliding 1 minute)

```
1 SELECT avg(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time(1 minute);
```

Figure 5.4: Average CEP Query (Sliding 10 seconds) - Energy consumption



The results in Table 5.5 show a decrease in the bandwidth consumption in comparison with Table 5.2 (no processing), however results in section 5.3.1 (jumping windows) are still better. Moreover, the results in Figures 5.4 and 5.5 show that similar to the CEP filtering query, sliding windows don't reduce the active states of the network interfaces. Differently from jumping windows where the frequency of outbound events depends on the window size, in

Figure 5.5: Average CEP Query (Sliding 1 minute) - Energy consumption

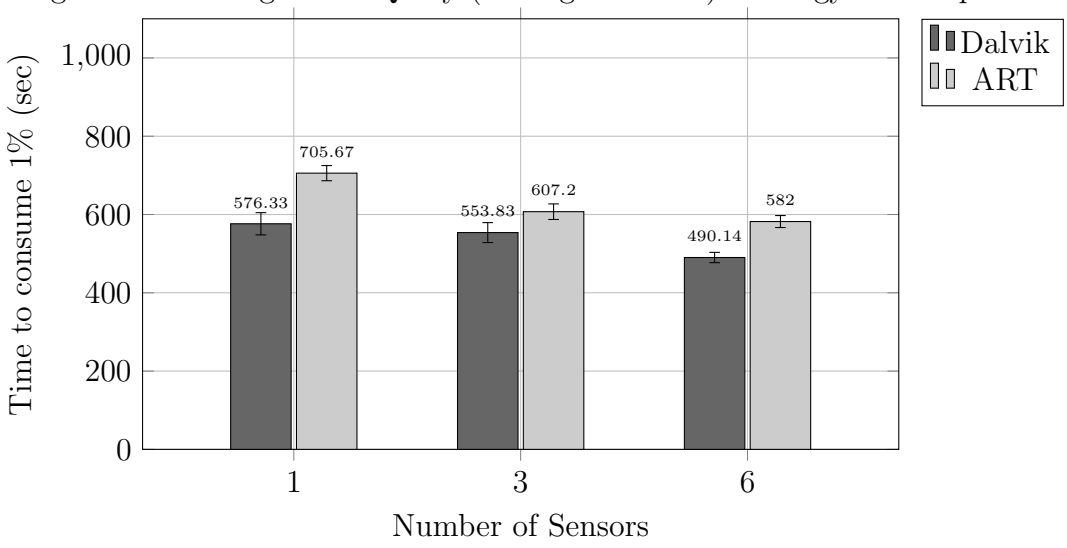


Table 5.5: Average CEP Query (Sliding) - Bandwidth consumption

10 seconds	1	3	6
Dalvik (kb)	3,214	6,047	10,715
ART (kb)	3,091	6,146	9,263
1 minute	1	3	6
Dalvik (kb)	2,519	6,131	11,355
ART (kb)	3,261	5,485	10,240

sliding windows the frequency remains constant and dependent on the arrival frequency of new data since each new item in the window causes a new outbound event. As the size of the sliding windows is increased, the energy savings are reduced since more data within the window must be processed by the MEPA service, and thus the network interface states remains frequent.

5.3.2 Maximum Query

The previous section presented experiments concerning the difference between using jumping and sliding windows, in regard to the energy consumption. However, we also want to show the impact of different kinds of processing (other than the average) using the same window types and sizes as the previous Section 5.3.1. Having said that, the following CEP queries search for the maximum value over a window of time instead of the average. Same configurations as the previous experiments were used.

Jumping Windows

Query in Code 5.6 process the maximum value using a jumping window of ten seconds, while query in 5.7 uses a jumping window of one minute.

Code 5.6: Maximum CEP Queries (Jumping 10 seconds)

```
1 SELECT max(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time_batch(10 sec);
```

Code 5.7: Maximum CEP Queries (Jumping 1 minute)

```
1 SELECT max(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time_batch(1 minute);
```

Figure 5.6: Maximum CEP Query (Jumping 10 seconds) - Energy consumption

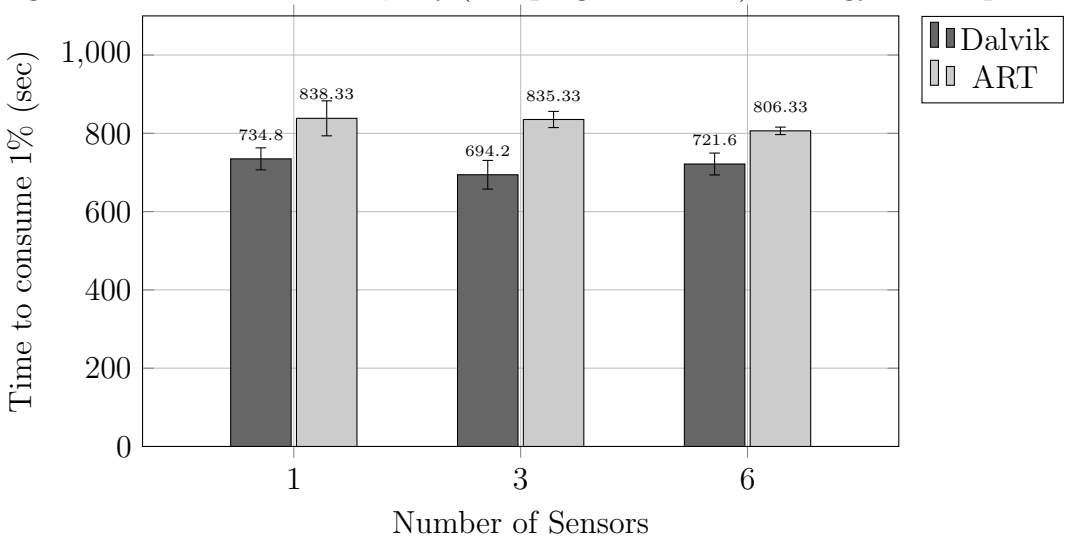


Figure 5.7: Maximum CEP Query (Jumping 1 minute) - Energy consumption

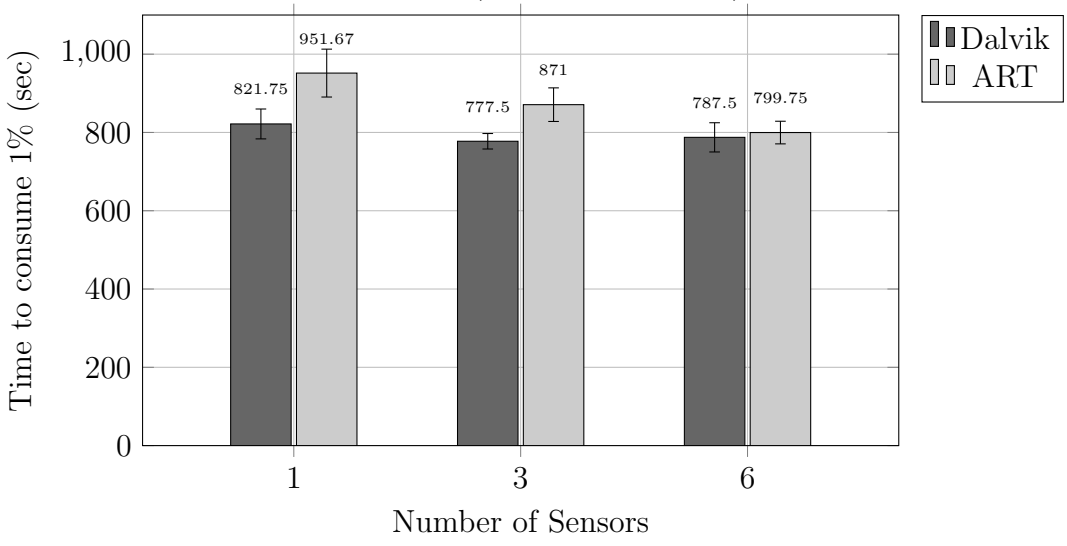


Table 5.6: Maximum CEP Query (Jumping) - Bandwidth consumption

10 seconds	1	3	6
Dalvik (kb)	266	347	264
ART (kb)	278	296	256
1 minute	1	3	6
Dalvik (kb)	68	92	64
ART (kb)	71	78	65

Sliding Windows

Queries in the Codes 5.8 and 5.9 process the maximum value as well, but using sliding windows of ten seconds and one minute respectively.

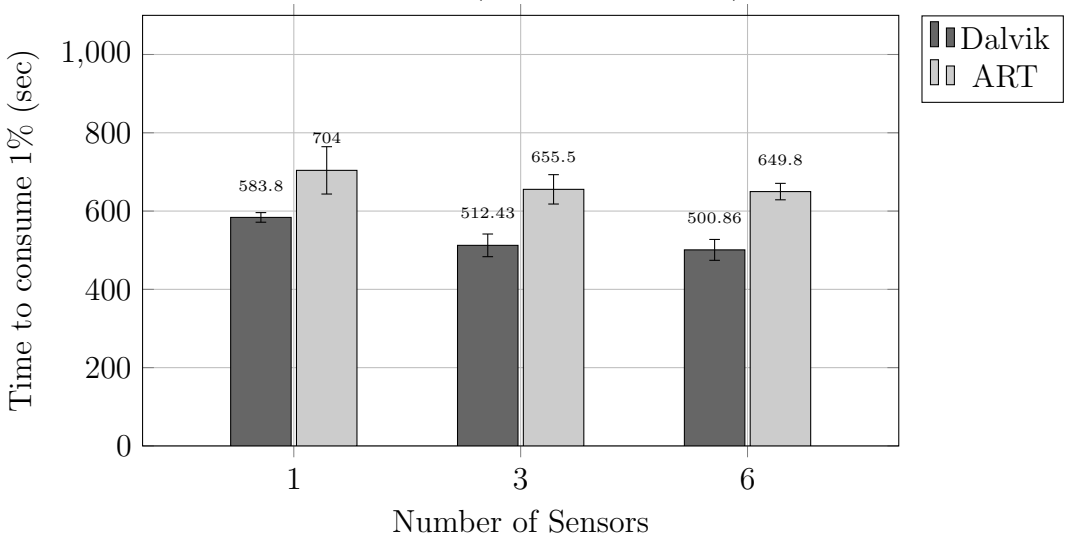
Code 5.8: Maximum CEP Queries (Sliding 10 seconds)

```
1 SELECT max(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time(10 sec);
```

Code 5.9: Maximum CEP Queries (Sliding 1 minute)

```
1 SELECT max(sensorValue[0]) FROM SensorData
2         (sensorName='Humidity').win:time(1 minute);
```

Figure 5.8: Maximum CEP Query (Sliding 10 seconds) - Energy consumption



The results using jumping windows (Figures 5.6, 5.7 and Table 5.6) and sliding windows (Figures 5.8, 5.9 and Table 5.7) are very similar to the ones presented in section 5.3.1 that processed the average value. Even though the maximum and the average CEP queries have a different kind of processing, the difference in their results is not much. This is due to the fact that only major differences will appear if the outbound frequency varies significantly, which is not the case.

Figure 5.9: Maximum CEP Query (Sliding 1 minute) - Energy consumption

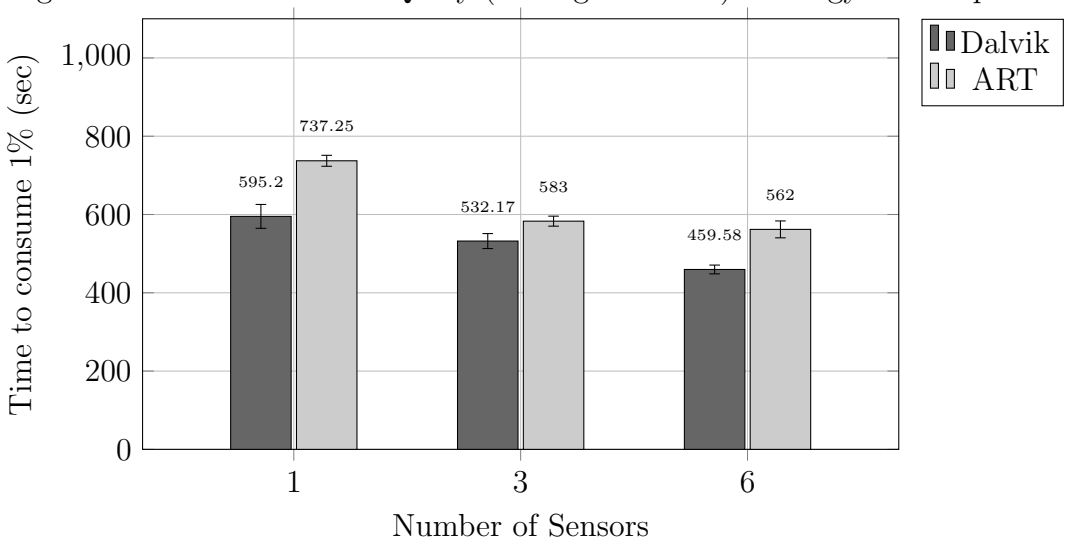


Table 5.7: Maximum CEP Query (Sliding) - Bandwidth consumption

10 seconds	1	3	6
Dalvik (kb)	3,393	6,115	11,745
ART (kb)	2,452	6,148	8,553
1 minute	1	3	6
Dalvik (kb)	2,860	6,184	10,512
ART (kb)	2,775	5,535	8,141

5.4 Aggregation Queries with Heavy Processing

In the following experiments we tested a more complex processing, in which the magnitude of the accelerometer sensors[3] was calculated in a window of time. These experiments were performed to show in a more clearly way, the relation between processing and outbound events frequency in regard to the energy consumption. The accelerometer used can measure acceleration in three directions simultaneously. The magnitude of the accelerometer was defined as $||a|| = \sqrt{a_0^2 + a_1^2 + a_2^2}$. Similar to the previous experiments, we used different window types and varied their size.

5.4.1 Jumping Windows

The query in Code 5.10 process the magnitude value by using a jumping window of ten seconds, while the query in Code 5.11 uses a jumping window of one minute.

[3]The accelerometer is a device that measures the acceleration in a specific direction from gravity and movement.

Code 5.10: Heavy Processing CEP Query (Jumping 10 seconds)

```
SELECT Math.sqrt(
        Math.pow(avg(sensorValue[0]), 2.0) +
        Math.pow(avg(sensorValue[1]), 2.0) +
        Math.pow(avg(sensorValue[2]), 2.0)
) as value FROM SensorData(sensorName='Accelerometer')
                .win:time_batch(10 sec);
```

Code 5.11: Heavy Processing CEP Query (Jumping 1 minute)

```
SELECT Math.sqrt(
        Math.pow(avg(sensorValue[0]), 2.0) +
        Math.pow(avg(sensorValue[1]), 2.0) +
        Math.pow(avg(sensorValue[2]), 2.0)
) as value FROM SensorData(sensorName='Accelerometer')
                .win:time_batch(1 minute);
```


Figure 5.10: Heavy Processing (Jumping 10 seconds) - Energy consumption

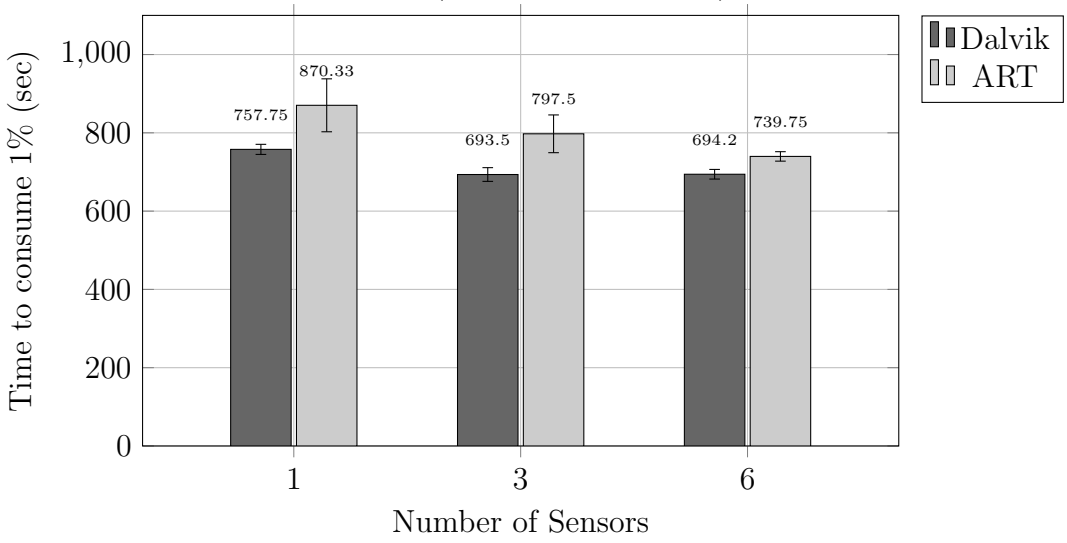
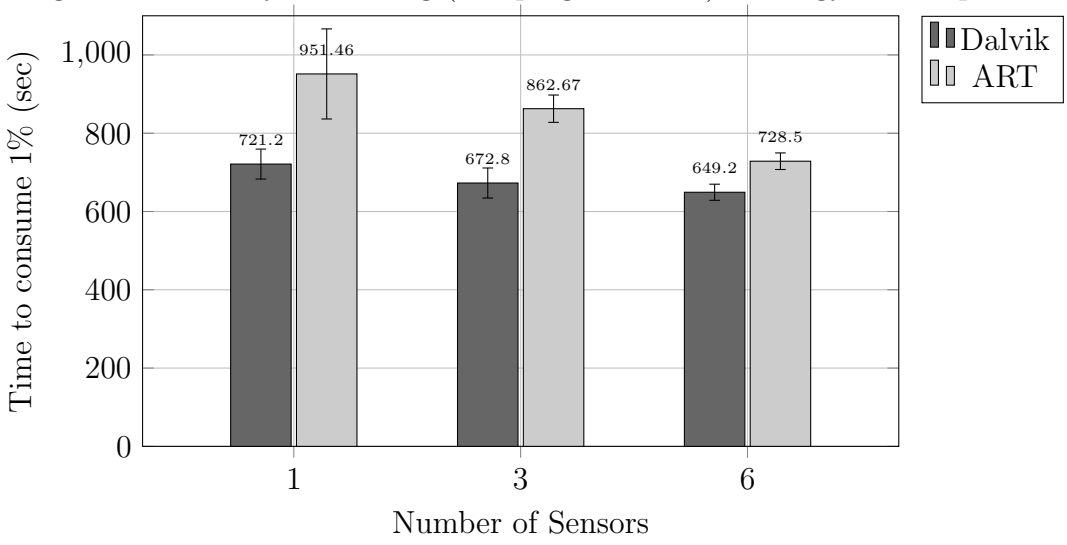


Table 5.8: Heavy Processing (Jumping) - Bandwidth consumption

10 seconds	1	3	6
Dalvik (kb)	268	331	255
ART (kb)	266	316	245
1 minute	1	3	6
Dalvik (kb)	78	148	62
ART (kb)	74	82	92

The results presented for the energy (Figures 5.10 and 5.11) and the bandwidth (Table 5.8) consumption are very similar to the ones in section 5.3, that made a simpler processing using jumping windows as well.

Figure 5.11: Heavy Processing (Jumping 1 minute) - Energy consumption



Queries with a more complex processing can consume a reasonable amount of energy depending on their outbound frequency. Hence, using complex CEP queries in mobile devices could be counterproductive if the events have a high rate of occurrence. Furthermore, heavy processing could also affect other applications execution and thus, user's experience. In such cases we have to consider divide the complex queries in two or more sub-queries, in order to send the most heavy part to the cloud. Nevertheless, queries like the ones presented in the former experiments can still be executed if the frequency of outbound events is low enough to compensate the energy consumption for the processing.

5.4.2 Sliding Windows

Queries in the Codes 5.12 and 5.13 also process the magnitude of the accelerometer data, but using sliding windows of ten seconds and one minute. Sliding windows increase the processing load and the outbound events frequency.

Code 5.12: Heavy Processing CEP Query (Sliding 10 seconds)

```
1 SELECT Math.sqrt(
2         Math.pow(avg(sensorValue[0]), 2.0) +
3         Math.pow(avg(sensorValue[1]), 2.0) +
4         Math.pow(avg(sensorValue[2]), 2.0)
5 ) as value FROM SensorData(sensorName='Accelerometer')
6                 .win:time(10 sec);
```

Code 5.13: Heavy Processing CEP Query (Sliding 1 minute)

```
SELECT Math.sqrt(
         Math.pow(avg(sensorValue[0]), 2.0) +
         Math.pow(avg(sensorValue[1]), 2.0) +
         Math.pow(avg(sensorValue[2]), 2.0)
) as value FROM SensorData(sensorName='Accelerometer')
                  .win:time(1 minute);
```

Figure 5.12: Heavy Processing (Sliding 10 seconds) - Energy consumption

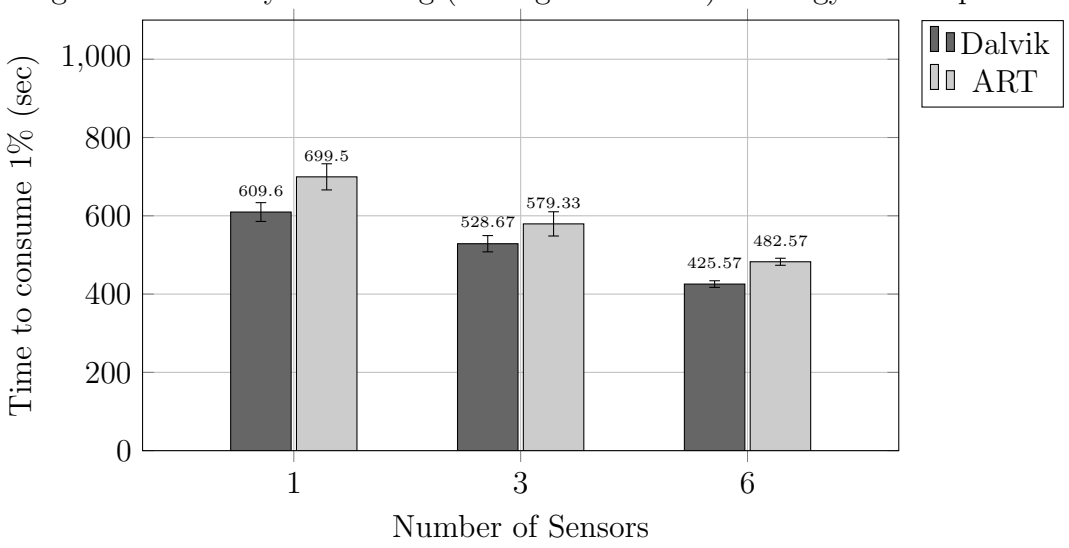
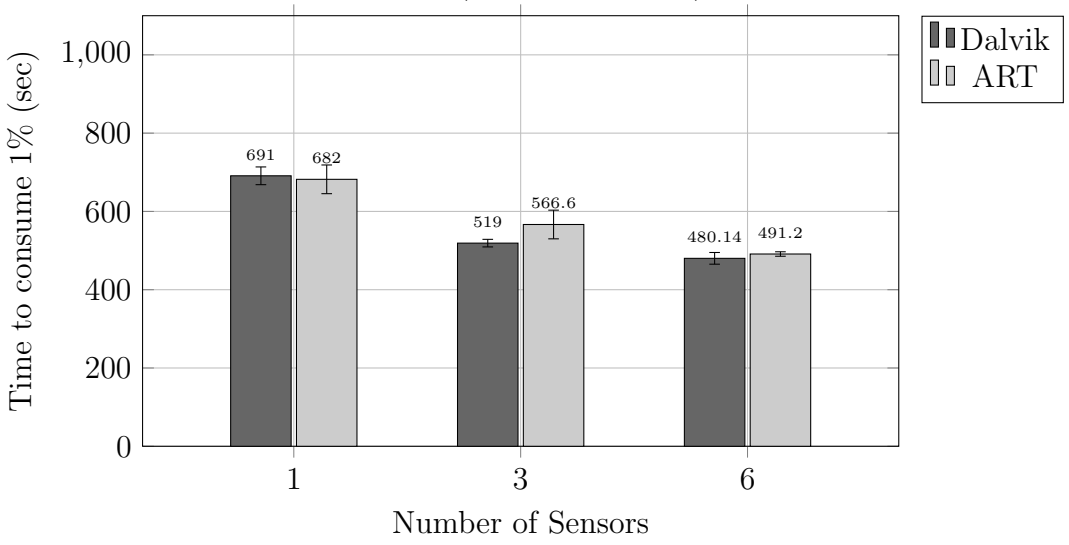


Figure 5.13: Heavy Processing (Sliding 1 minute) - Energy consumption



The results in Figures 5.12 and 5.13 present a higher energy consumption than the simpler aggregation experiments. This is due to the heavier processing, but mainly to the fact that sliding windows include a higher outbound frequency of events, which increased the use of the network interfaces.

Table 5.9: Heavy Processing (Sliding) - Bandwidth consumption

10 seconds	1	3	6
Dalvik (kb)	2,998	5,764	10,997
ART (kb)	2,766	5,912	11,345
1 minute	1	3	6
Dalvik (kb)	3,026	4,933	10,630
ART (kb)	2,472	5,330	11,113

5.5 Aggregation and Pattern Match Queries

Here we tested two different queries (see Codes 5.14 and 5.15) executing at the same time. The first one processed the average temperature in a jumping window of 10 seconds, while the second expected four consecutive temperature values, each one higher than the previous and the first one higher than 20. In the case of the second query we don't have control over the number of outbound events, since it depends on the sequence of temperature events that fulfill the pattern. This experiment is intended to show the energy consumption with two queries executing at the same time. Figure 5.14 and Table 5.10 present the energy and bandwidth measurements. Results show that even with two CEP queries executing, if they have a low rate of outbound events, it is still possible to see a reduction in the energy consumption.

Code 5.14: Aggregation CEP Query

```
1 SELECT avg(sensorValue[0]) as value FROM SensorData
2  (sensorName='Temperature').win:time_batch(10 sec);
```

Code 5.15: Pattern Match CEP Query

```
1 SELECT * FROM SensorData(sensorName='Temperature')
    match_recognize (
2  measures
3          A as temp1, B as temp2, C as temp3, D as temp4
4  pattern (A B C D) define
5          A as A.sensorValue[0] > 20,
6          B as (A.sensorValue[0] < B.sensorValue[0]),
7          C as (B.sensorValue[0] < C.sensorValue[0]),
8          D as (C.sensorValue[0] < D.sensorValue[0])
9 );
```

Figure 5.14: Aggregation and Pattern Match - Energy consumption

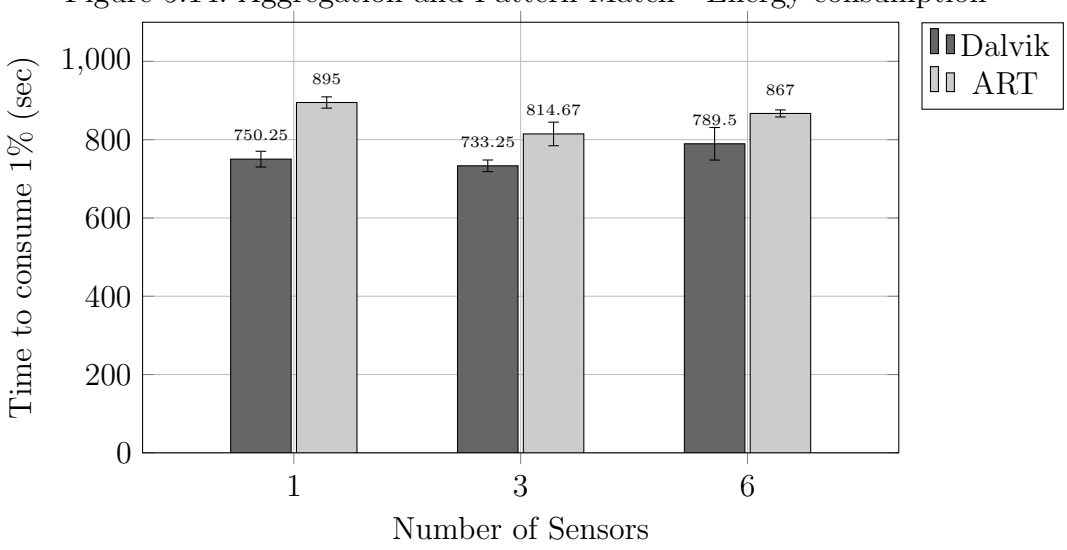


Table 5.10: Aggregation and Pattern Match - Bandwidth consumption

No. of Sensors	Dalvik (kb)	ART (kb)
1	258	270
3	389	326
6	298	264

5.6 Discussion

One of the most important aspects of CEP is that it helps to abstract away the complexities of data processing, allowing to replace code with expressive queries. Such queries can represent different kinds of data stream processing that may be adequate or not to be executed in mobile devices. In fact, there could be situations where no processing is desired at all. The experiments showed that in most of the cases (with a continuous generation of sensor data) the MEPA service can significantly decrease the energy and bandwidth consumption compared with sending all the data to the cloud. The graphic in Figure 5.15 presents a comparison of bandwidth consumption from all the experiments with six M-OBJs, however the pattern is similar with the experiments with one and three M-OBJs.

Moreover, the results in sections 5.3 and 5.4 indicate that jumping windows CEP queries are better suited to be executed on mobile devices than sliding windows, since sliding windows impose a significant use of CPU, and have a high rate of outbound events. If the data produced by sliding windows is directly send to the cloud, it will generate a significant use of the Internet connection (limiting the available bandwidth), and reduce the idle states of the network interfaces. Additionally, if the communication is being made by using

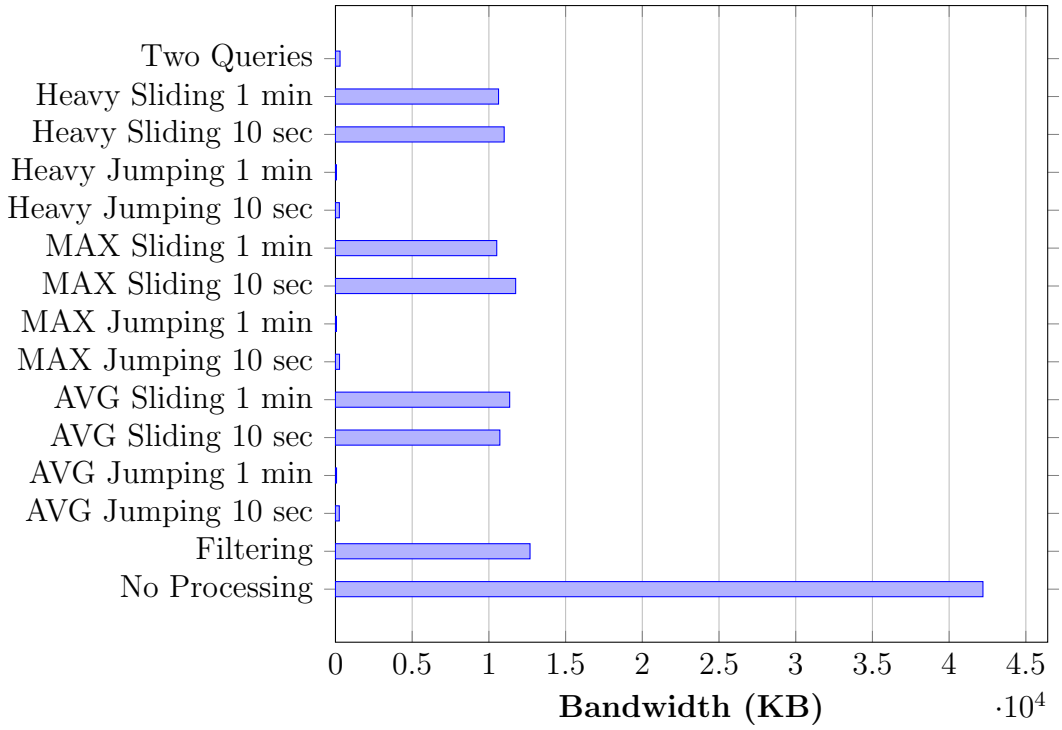


Figure 5.15: Six CEP Queries - Bandwidth consumption

mobile networks, it could generate elevated costs to the users. For example, the results for a heavy processing query using a jumping window of one minute shown in Figure 5.11 are much better, in terms of energy consumption, than the ones using a sliding window of the same size in Figure 5.13. To see the bigger picture, Figure 5.16 presents the energy consumption for all the experiments with six M-OBJs, which has a similar pattern to experiments with one and three M-OBJs.

Nevertheless, there are other scenarios where it is possible to use the output of the CEP queries that use sliding windows as the input for another query X in the MEPA Service. In such case, if the query X is the one that communicates with the cloud and can significantly reduce the outbound frequency, energy consumption can still be reduced. It could be also possible to use sliding windows when the frequency of incoming sensor data is low enough to avoid using the Internet communication constantly. Moreover, as it is shown in Figures 5.2 and 5.2, the bigger the size of the jumping windows, the more energy consumption that can be reduced. Since it will allow to increase the time the network interface remains in idle state. In the case of pattern match query, we have no control over the amount of generated events, thus the expected frequency of the pattern will determine if they are suitable as mobile processing.

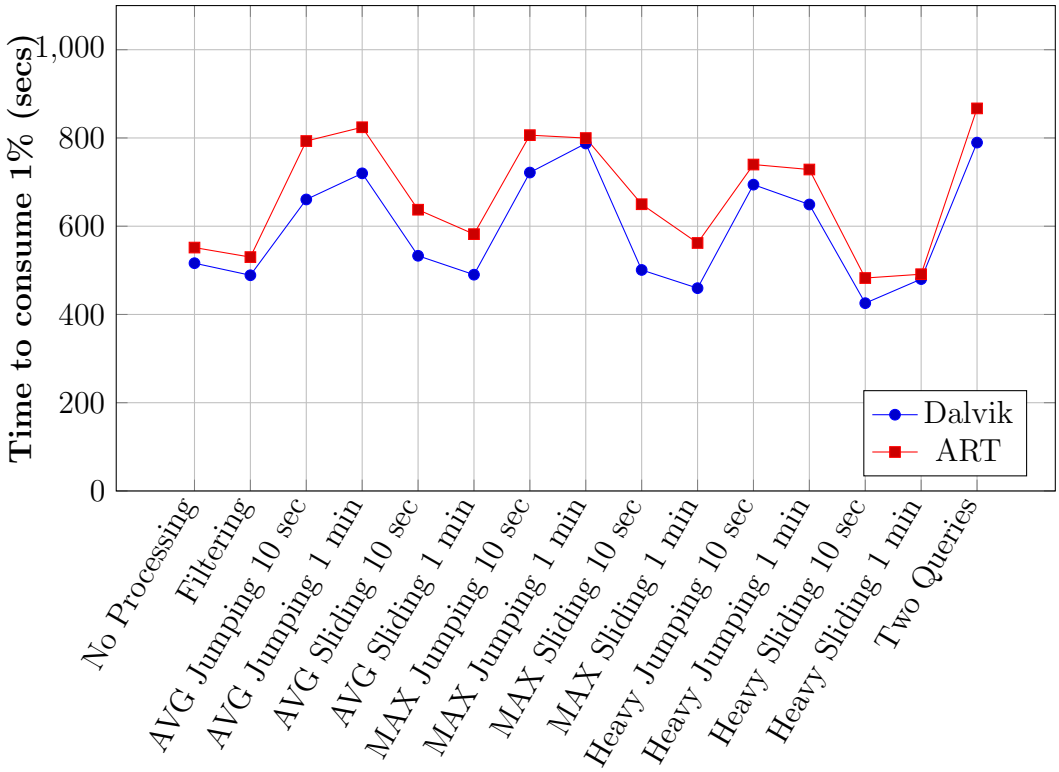


Figure 5.16: Six CEP Queries - Energy consumption

Finally, we can conclude that the impact of having one or more queries depends on the frequency of detected events (see results in Section 5.5). As we explained before, energy savings in wireless communication are strongly-related to reducing the duty cycles (active states) of the network interfaces (specially the 2G/3G/4G interfaces (Carroll e Heiser, 2010)). Hence, local processing should as much as possible reduce the frequency and the number of message transmissions to the cloud. However, if the complexity of the processing is high, and the probing of the sensor data transmission is infrequent, then sending them to the cloud could be better (but only as bulk messages with many sensor data). In the results we can also see that using ART the decrease of energy is mitigated. The use of ART also resolves many performance related issues that was previously seen with Dalvik (Eggum, 2014).

6
Related Work

Several works in both industry and academia (Zachariah et al., 2015; Pereira et al., 2013; Billet e Issarny, 2014; Min et al., 2014; Chung et al., 2014) propose the use of average things (moderately powerful things, e.g. arduino[1], edison[2]) and personal mobile devices (smartphones and tablets) as the enablers of the Internet of Things (IoT). These devices could act as temporary IP routers and opportunistic context providers (e.g. provide location, local time) for simpler Things. Nevertheless, only recent works are concerned with the amount of transferred data and energy consumption of the IoT-gateways, which are usually energy-constrained devices. (Billet e Issarny, 2014) argue that WSAN- and Web-based techniques need to be integrated within a fully-distributed streaming middleware that is able to run directly in every type of average thing. So, it will allow the sensor network to perform as much in-network processing as possible before sending any data to a proxy or the cloud. However, in-network processing depends on the available resources (sensors/actuators) in the current location of the IoT-gateways, for example the sensors deployed in a university could be different from the ones in a hospital. Hence, it imposes an important requirement of adaptability of the processing of the streaming data.

Novel proposals (Stipkovic et al., 2013; Dunkel et al., 2013; Saleh e Sattler, 2013; Schilling et al., 2010; Govindarajan et al., 2014; Chen et al., 2014; Stojanovic et al., 2014; Kim et al., 2009) follow the idea of Fog Computing (Bonomi et al., 2012), which is a cloud close to the ground, typically, but not exclusively located at the edges of the network. The main characteristics of Fog Computing are low latency and location awareness, mobility, wide-spread geographical distribution, very large number of nodes, predominant role of wireless access, strong presence of streaming and real time applications, and heterogeneity. Contrary to the more centralized Cloud, the Fog needs to communicate directly with mobile devices in order to provide services, creating a requirement for a widely spread deployment. Large-scale sensor networks will constantly monitor a wide variety of environments (heterogeneity), requiring distributing computing and storage resources. It is very important for Fog application to involve (near) real-time interactions instead of batch processing (Bonomi et al., 2012). Most of Fog's characteristics are important requirements

[1]https://www.arduino.cc/
[2]http://www.intel.com/content/www/us/en/do-it-yourself/edison.html

for IoT, for example fog nodes can provide real-time processing, location and context awareness, while the Cloud provides global centralization.

(Stipkovic et al., 2013; Dunkel et al., 2013) argue that recently several works have proposed the use of Complex Event Processing (CEP) to process sensor data in backend servers, using mobile devices only as event sources. Hence, they propose the use of CEP directly in mobile devices, avoiding the use of the cloud in order to reduce transmission load, save resources, and ensure privacy of data since it remains only on the mobile devices (e.g. GPS). They mentioned that this is a novel field of application since few years ago smartphones weren't as powerful as they are today, so the idea of having a mobile CEP was discarded. Moreover, they imply that CEP is a software technology which is very well suited for IoT specific processing in mobile devices since it has in-memory processing with low latency which enables (near) real-time operations and correlates heterogeneous data.

Edge devices are starting to have significant processing capabilities, in this regard (Govindarajan et al., 2014) introduce a new approach where CEP processing is distributed among edge nodes (e.g. wireless sensors, smartphones) and the Cloud (VMs). The pipeline of CEP queries is represented by a graph, where the vertices are the sets of queries, and the edges the event streams that connect the output of a query to the input of the next query. They make use of CEP because it has a fast detection on high traffic event streams (heterogeneous data), allowing to execute different actions in (near) real-time. Moreover, the need for a distributed CEP processing solution arises from the performance impact of the systems as they scale to a town or a city, where there will appear a different number and types of sensors. Their proposal only covers the system's architecture and representation of CEP queries. They don't consider a dynamic deployment of CEP queries, nor have any evaluation over the system.

(Chen et al., 2014) propose an architecture for distributed CEP to meet the needs of real-time streaming of information processing. The authors advocate that traditional centralized CEP is unsuitable for IoT because devices have to send large amounts of raw data to central servers, and network quality could not be always good enough to provide sufficient bandwidth. Thus, they imply that if the data can be pre-processed at the the gateways side, and only sent the intrinsic information to the back-end servers, then not only the transfer speed will be increased but also the processing burden will be reduced. To simplify the use of their system, the authors included a web interface, where each user will have their own workspace to deploy their CEP queries, save them and create their own applications. An application may contain many query sets,

where each query set is called as Event Processing Network (EPN). However, their work doesn't have support for mobile IoT, where gateways can be moved to different environments (e.g. hospital, university), and thus be connected with different sets of sensors.

(Stojanovic et al., 2014) implies that with the increase of data collected by mobile devices, there is a need to include local processing to detect real-time situations. Hence, they propose a system that will handle heterogeneous data with the use of CEP in both mobile devices and server. Their system will also adapt the CEP queries to different use cases (context-aware), because it is not the same to monitor a person who is running, to a person sitting in a living room. Moreover, since there are several CEP technologies with different processing languages, they also included a translator of CEP logics to work with different engines. By doing this, better recommendations will be sent, better actions will be taken and battery will be saved, since it won't be necessary to have unused queries running on the mobile devices or cloud. As an example; if some sensor became unavailable, and the sensor data cannot be replaced, the query that uses such information will be un-deployed, in order to save battery. Nevertheless, the authors still allow all the information to be sent to the server in order to increase their complex knowledge (historical data), moreover they only cares about situations related to health and fitness.

In the work of (Kim et al., 2009), the main concern is the execution of CEP processing only in the cloud, so the authors moved the processing to the mobile devices in order to reduce the latency for the responses. Authors make use of the Data Distribution Service (DDS) (Pardo-Castellote, 2003) to exchange data/events (sensor data) among nodes, and use CEP to create useful information to the users (e.g. the combination of data as heartbeat rate and blood pressure could represent the quality of blood circulation). First data is collected from various devices (publishers, e.g. sensors, GPS, cameras) and transmitted to each user (subscribers, e.g. PDA, smartphone) through the DDS network. The data is processed in each mobile device depending on the user's demands, and delivered to the different applications. CEP queries can be changed depending on the requirements of the applications.

There is an increasing number of sensor networks that provide data (events) that can be useful for a lot of different applications that goes from health to disaster detection (e.g. air quality detection, toxic substances in the environment). Moreover, the vast majority of sensor network's solutions only cares about the transmission of the events to the cloud (central processing entity). Such approaches are not adequate for resource-limited systems, it would deplete the node's resources (e.g. energy), and overload the network

with unnecessary information. In this regard, (Saleh e Sattler, 2013) propose to distribute CEP queries among the network as a graph, where each node communicates with each other using a publish/subscribe mechanism. A sensor catalog will make possible to find information about the different sensor devices, such as their computational power, memory size, communication cost and availability. These information will be used at the moment of decide which CEP rules go to which nodes (e.g. server, sensors). Thus, it will reduce the amount of useless information sent among nodes and to the cloud, and for instance, reduce the energy consumption of the system.

Authors in (Schilling et al., 2010) state that "Almost none of the distributed CEP approaches proposed in the scientific literature has made it into industrial applications so far". Current solutions are centralized because important context information (related to business processes) is usually located in the data centers. Thus, they propose an architecture, where CEP queries can be dynamically deployed to the different nodes in the cloud. They use a meta language to model the CEP queries, in order to adapt them to the different CEP implementations that support such processing. Their system can benefit from placing latency-relevant queries on nodes with good network connection, and independent queries on any node within the network to reduce the overall detection time by using parallel computation. CEP queries are deployed according to their requirements, for example some rules could have some restrictions on the computational power required on the machine they shall run on. The system has been developed and created in collaboration with the IBM Research and Development Laboratory Böblingen, where it is currently running.

6.1 Discussion

To compare the presented works we used the following criteria: Current IoT solutions are server and mobile based, meaning that the data processing is executed in the cloud or in the edge networks. Newer approaches proposed an **Hybrid Architecture**, where the processing is distributed between the cloud and the edge devices. This can be regarded as a kind of preprocessing step to mitigate the transmissions to the cloud to reduce energy and bandwidth consumption. Moreoever, a key technology considered by several works is **Complex Event Processing (CEP)**, which allows to abstract the operations that form the event processing logic, and thus separate it from the application logic. Additionally, event processing frequently deals with events that occur, or could occur, in the real world (Etzion e Niblett, 2010).

Another criteria is the use of **Mobile Devices (MD)** as IoT gateways for transparent access to the Internet and cloud-powered applications. Nowadays, mobile devices possess a sufficient processing power to include CEP for local processing. Furthermore, mobile devices can be found in several environments (e.g. universities, hospitals, houses) with different resources (i.e. sensors, actuators), and hence different CEP queries may need to be swapped with the ones that are already executing (**processing Adaptability**). Finally, energy is scarce resource in mobile devices, therefore, not any CEP query can be installed in them (**Energy awareness**). It is important to provide a benchmark that could work as a guideline for the interested people that want to include CEP for in-network processing, to decide which kind of CEP queries should be executed in mobile devices.

Authors	**Hybrid**	**CEP**	**MD**	**Adaptability**	**Energy**
(Billet e Issarny, 2014)	X			X	
(Stipkovic et al., 2013)		X	X		
(Govindarajan et al., 2014)	X	X	X		
(Chen et al., 2014)	X	X	X		
(Stojanovic et al., 2014)	X	X	X	X	
(Kim et al., 2009)		X	X	X	
(Saleh e Sattler, 2013)	X	X			
(Schilling et al., 2010)		X		X	

Table 6.1: Comparison of related works

(Stipkovic et al., 2013; Dunkel et al., 2013) approach is based on Ambient Assisted Living (AAL), and don't handle adaptability of CEP queries. Moreover, they leave all the processing to the mobile devices, avoiding the use of the cloud and thus, it can't handle situations where a global view of the data is required. For example, when aggregation of data is required from different mobile devices in different geographic locations. (Govindarajan et al., 2014) proposal only covers the architecture and representation of the CEP queries. They don't consider a dynamic deployment of queries, nor have any evaluation over the system. (Chen et al., 2014) approach is based on static environments where gateways are always connected to the same devices. It doesn't have support for mobile IoT, where gateways can be moved to different environments (e.g. hospital, university), and thus be connected with different sets of sensors.

The work that handles the most issues as our work does is (Stojanovic et al., 2014). However, they only care about situations related to health and fitness, and don't present any benchmark/experiments of their system (e.g. throughput, energy) but instead they only present a software infrastructure. (Kim et al., 2009) don't consider that mobile devices can act as data sources

and node processors at the same time. All sensor data is sent to the mobile devices where the processing takes place. (Saleh e Sattler, 2013) propose the use of CEP on the sensor nodes, differently from ours where we intent to include CEP processing in more powerful devices (smartphone, tablets) that act as gateway for the sensor nodes. In fact, authors don't consider mobility, where CEP queries need to be deployed dynamically depending on the current available resources and relevant situations. (Schilling et al., 2010) don't consider processing at the edges, but only distribute it on the cloud. Nevertheless, the work have a dynamic exchange of rules and events among the different nodes on the network.

Although there are many approaches that try to include CEP in mobile devices, almost none of them explores its capacity for adaptation (reconfiguration). However, even though we propose processing adaptability, and created an environment that can allow such behavior, since our focus is on energy and bandwidth consumption, we don't have any experiments that can validate this characteristic. A similar situation occurs for the works that also include adaptability, none of them have a validation for this characteristic. Moreover, none of the previous works provide a benchmark about the energy/bandwidth impact of the use of CEP in mobile devices, which we believe (as explained in chapter 3) are some of the most important aspects at the moment of including it in the IoT-gateways.

7
Conclusion and Future Work

The popularity and proliferation of mobile handheld devices (smartphones, tablets) have led to propose their use as IoT gateways. Mobile devices are starting to have substantial processing capabilities, and a myriad of features such as short-range wireless communication. Hence, several works have started to research how these devices with Internet connectivity can act as temporary Internet access providers to simpler things (sensors/actuators) that only have short range wireless interfaces. Most research however, focuses only on using mobile devices as sources of data (sensor and user data) to be processed by a cloud service, leaving aside their processing power and energy consumption for Internet communication.

In this regard, this dissertation presents an IoMT sensor gateway solution with dynamic local processing of sensor data using conventional mobile devices. It is an extension of the *Mobile Hub* which is a mobile middleware that allows any smart thing/object with some short-range communication protocol (WPAN technology) to be opportunistically connected to the Internet. A prototype was implemented for Android; it communicates with a mobile communication middleware called Scalable Data Distribution Layer (SDDL), and has Bluetooth Low Energy (BLE) as the showcase WPAN. BLE turned out to be an excellent choice as a first WPAN technology for its high efficiency, connectivity, and low energy consumption. Moreover, BLE is being widely adopted for smart objects, and is being supported by most smartphone makers.

In order to gain some meaning, the enormous volumes and varieties of continuous data streams are usually correlated in the cloud since it has a virtually unlimited processing, energy capacity and the ability to analyze collectively data obtained from different mobile devices. Nevertheless, WMAN and WLAN network interfaces (e.g. WiFi and specially 2G/3G/4G) are strong battery-draining components in mobile devices, where their energy consumption is largely defined by their frequency of active states. Using handhelds only to transmit such data would drain their batteries and overcrowd the bandwidth with naive information. Thus, local processing should as much as possible reduce the frequency and the number of message transmissions to the cloud. In this regard, the characteristics of Complex Event Processing (CEP), allow an easy specification of event patterns, on-the-fly reconfiguration and fast detection of events over continuous data streams. In fact, recent studies have shown that it is possible to use CEP in mobile devices, which led us

include it for local processing in our IoT-gateway (the M-Hub).

Furthermore, we argue that since mobile devices have limited memory and processing capacity (in comparison with powerful cloud servers), it is recommended that they only execute the CEP queries that are corresponding to the currently available sensors in the device's vicinity. Such sensors may vary as the M-Hub is connected with different M-OBJs during its lifetime and in different situations. Thus, since CEP allows an on-the-fly reconfiguration, we built a software framework around CEP to decouple the processing queries from the application. It will allow the M-Hub to deploy/un-deploy CEP queries using the connection link with the cloud. However, processing also consumes a reasonable amount of energy. If the CEP processing doesn't manage to increase the idle intervals between transmissions, it will result terrible for the activity time of mobile devices. In order to address this problem, we did performance experiments using BLE SensorTag devices that measured and compared the energy and bandwidth consumption between sending all the sensor data, and only pre-processed data to the cloud. The results obtained from these experiments are quite encouraging and show that CEP queries that represent events with a low frequency of occurrence (except for very complex computations) are more adequate for mobile devices. This is due to the fact, that low frequency situations will reduce the active states of wireless network interfaces. Nevertheless, if the complexity of processing is high and the probing of the sensor data transmission is infrequent, then sending it to the cloud could be better (but only as bulk messages with many sensor data).

Finally, most of the related works focus in the still limited processing capacity of mobile devices in comparison with powerful workstation machines. Hence, their experiments are mainly about the current throughput supported by mobile CEP. Nevertheless, IoT gateways are not supposed to process the same amount of information as cloud servers. Besides, since Dalvik and ART were separately studied, we can conclude that Android is becoming a more adequate environment for CEP, since ART showed an improvement in the battery consumption in comparison with Dalvik. The results presented in this dissertation can be used as a benchmark for the interested researchers by showing the impact that CEP has in the activity time of mobile devices.

7.1 Future Work

Although, the presented results have proved the efficiency provided by using CEP to pre-process sensor data in IoT gateways, we are aware that interesting improvements, research, software development and applications

can be derived from this work. Thus, this section intends to present some of the future directions that this work may take. In particular, our future work includes: a way for balancing the CEP processing among cloud and mobile nodes, investigate the problems and possible approaches inter M-Hub handover protocols aiming to deliver detected events to nearby M-Hubs, in case a M-Hub is unable to establish an Internet connection.

As explained before, energy consumption is a critical issue in mobile devices towards their adoption by end-users as the generic propagator devices for IoMT. Users wants to extend the operation time of their devices as much as possible. Hence, an enhanced energy manager could be included to automatically start and stop queries depending on certain aspects of mobile devices (e.g. battery level, number of running applications, type of network connection). Such energy manager could be build by using the MEPA Service, with queries that analyze the status of the mobile device in order to modify other services behavior (e.g. the connection and location services).

Additionally, given that our approach can deploy/un-deploy CEP queries from the cloud, it is important to limit the mobile CEP operators by avoiding to deploy queries with a high-outbound frequency (e.g. sliding windows) in the M-Hub. Another possible study, includes means of sending commands to M-OBJs with actuators, and thus support any Internet-wide remote control of smart things, such as home appliances where an event can start an action locally without the need to sent any information to the cloud. In fact, it brings other important challenges such as conflict resolution, which arises when multiple applications attempt to actuate over the same device in opposing ways (Teixeira et al., 2011).

8
Bibliography

BALASUBRAMANIAN, N.; BALASUBRAMANIAN, A.; VENKATARAMANI, A. Energy consumption in mobile phones: A measurement study and implications for network applications. In: **Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference**. New York, NY, USA: ACM, 2009. (IMC '09), p. 280–293. ISBN 978-1-60558-771-4. Disponível em: <http://doi.acm.org/10.1145/1644893.1644927>.

BILLET, B.; ISSARNY, V. Dioptase: a distributed data streaming middleware for the future web of things. **Journal of Internet Services and Applications**, Springer London, v. 5, n. 1, 2014. ISSN 1867-4828. Disponível em: <http://dx.doi.org/10.1186/s13174-014-0013-1>.

BONOMI, F. et al. Fog Computing and Its Role in the Internet of Things. In: **Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing**. New York, NY, USA: ACM, 2012. (MCC '12), p. 13–16. ISBN 978-1-4503-1519-7. Disponível em: <http://doi.acm.org/10.1145/2342509.2342513>.

CARROLL, A.; HEISER, G. An analysis of power consumption in a smartphone. In: **Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference**. Berkeley, CA, USA: USENIX Association, 2010. (USENIXATC'10), p. 21–21. Disponível em: <http://dl.acm.org/citation.cfm?id=1855840.1855861>.

CHEN, C. Y. et al. Complex Event Processing for the Internet of Things and its Applications. In: **Automation Science and Engineering (CASE), 2014 IEEE International Conference on**. [S.l.: s.n.], 2014. p. 1144–1149.

CHUNG, T.-Y. et al. Design and implementation of light-weight smart home gateway for social web of things. In: **Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on**. [S.l.: s.n.], 2014. p. 425–430.

DACOSTA, F. **Rethinking the Internet of Things: A Scalable Approach to Connecting Everything**. 1st. ed. Berkely, CA, USA: Apress, 2013. ISBN 1430257407, 9781430257400.

DAVID, L. et al. A DDS-based middleware for scalable tracking, communication and collaboration of mobile nodes. **Journal of Internet Services and Applications**, Springer London, v. 4, n. 1, 2013. ISSN 1867-4828. Disponível em: <http://dx.doi.org/10.1186/1869-0238-4-16>.

DUNKEL, J.; BRUNS, R.; STIPKOVIC, S. Event-based smartphone sensor processing for ambient assisted living. In: **Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on**. [S.l.: s.n.], 2013. p. 1–6.

ECKERT, M.; BRY, F. Complex Event Processing (CEP). **Informatik-Spektrum**, v. 32, n. 2, p. 163–167, 2009.

EGGUM, M. **Smartphone Assisted Complex Event Processing**. Tese (dissertation) — University of Oslo, 2014. Disponível em: <https://www.duo.uio.no/bitstream/handle/10852/41663/Marcel-Eggum—Thesis.pdf>.

ETZION, O.; NIBLETT, P. **Event Processing in Action**. 1st. ed. Greenwich, CT, USA: Manning Publications Co., 2010. ISBN 1935182218, 9781935182214.

GOVINDARAJAN, N. et al. Event processing across edge and the cloud for internet of things applications. In: **Proceedings of the 20th International Conference on Management of Data**. Mumbai, India, India: Computer Society of India, 2014. (COMAD '14), p. 101–104. Disponível em: <http://dl.acm.org/citation.cfm?id=2726970.2726985>.

GUBBI, J. et al. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. **Future Generation Computer Systems**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 29, n. 7, p. 1645–1660, set. 2013. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2013.01.010>.

JAEIN, K. et al. A study on cep performance in mobile embedded system. In: **ICT Convergence (ICTC), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 49–50.

KIM, D. et al. Embedded cep engine used in dds-based mobile devices for differentiated services for customers. In: **Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on**. [S.l.: s.n.], 2009. p. 645–646.

LI, D.; HALFOND, W. G. J. An investigation into energy-saving programming practices for android smartphone app development. In: **Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS)**. [S.l.: s.n.], 2014.

MAYER, R. Real-time distributed complex event processing for big data scenarios. In: **Distributed Event-Based Systems (DEBS) Ph. D. Forum**. [S.l.: s.n.], 2013.

MIN, D. et al. Design and implementation of heterogeneous iot gateway based on dynamic priority scheduling algorithm. **Transactions of the Institute of Measurement and Control**, SAGE Publications, v. 36, n. 7, p. 924–931, 2014.

PARDO-CASTELLOTE, G. OMG Data-Distribution Service: architectural overview. In: **Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on**. [S.l.: s.n.], 2003. p. 200–206.

PARDO-CASTELLOTE, G. OMG Data-Distribution Service: architectural overview. In: **Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on**. [S.l.: s.n.], 2003. p. 200–206.

PATHAK, A.; HU, Y. C.; ZHANG, M. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In: **Proceedings of the 7th ACM European Conference on Computer Systems**. New York, NY, USA: ACM, 2012. (EuroSys '12), p. 29–42. ISBN 978-1-4503-1223-3. Disponível em: <http://doi.acm.org/10.1145/2168836.2168841>.

PENTIKOUSIS, K. In search of energy-efficient mobile networking. **Comunications Magazine, IEEE**, v. 48, n. 1, p. 95–103, January 2010. ISSN 0163-6804.

PEREIRA, P. P. et al. Enabling Cloud Connectivity for Mobile Internet of Things Applications. In: **Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering**. Washington, DC, USA: IEEE Computer Society, 2013. (SOSE '13), p. 518–526. ISBN 978-0-7695-4944-6. Disponível em: <http://dx.doi.org/10.1109/SOSE.2013.33>.

SALEH, O.; SATTLER, K.-U. Distributed Complex Event Processing in Sensor Networks. In: **Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management - Volume 02**. Washington, DC, USA: IEEE Computer Society, 2013. (MDM '13), p. 23–26. ISBN 978-0-7695-4973-6. Disponível em: <http://dx.doi.org/10.1109/MDM.2013.60>.

SCHILLING, B. et al. Distributed heterogeneous event processing: Enhancing scalability and interoperability of cep in an industrial context. In: **Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems**. New York, NY, USA: ACM, 2010. (DEBS '10), p. 150–159. ISBN 978-1-60558-927-5. Disponível em: <http://doi.acm.org/10.1145/1827418.1827453>.

SCHMIDHäUSER, S. **Dynamic operator splitting in mobile CEP scenarios**. 2014. Monograph (Informatic Student), University of Stuttgart. Disponível em: <http://elib.uni-stuttgart.de/opus/volltexte/2014/9766>.

SILVA, M. E. L. D.; RORIZ., M. Mr-udp: Yet another reliable user datagram protocol, now for mobile nodes. **Monografias em Ciência da Computação, nr**, Institute for Informatics, Pontifical Catholic University of Rio de Janeiro, v. 1200, p. 06–13, 2013. ISSN 0103-9741.

SKORIN-KAPOV, L. et al. Energy efficient and quality-driven continuous sensor management for mobile iot applications. In: **Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on**. [S.l.: s.n.], 2014. p. 397–406.

STIPKOVIC, S.; BRUNS, R.; DUNKEL, J. Pervasive computing by mobile complex event processing. In: **e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on**. [S.l.: s.n.], 2013. p. 318–323.

STOJANOVIC, N. et al. Mobile CEP in Real-time Big Data Processing: Challenges and Opportunities. In: **Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems**. New York, NY, USA: ACM, 2014. (DEBS '14), p. 256–265. ISBN 978-1-4503-2737-4. Disponível em: <http://doi.acm.org/10.1145/2611286.2611311>.

TALAVERA, L. et al. The Mobile Hub concept: Enabling applications for the Internet of Mobile Things. In: **Pervasive Computing and Communication Workshops (PerCom Workshops), 2015 IEEE International Conference on**. [S.l.: s.n.], 2015. p. 123–128.

TARKOMA, S. et al. **Smartphone Energy Consumption: Modeling and Optimization**. Cambridge University Press, 2014. ISBN 9781107042339. Disponível em: <https://books.google.fi/books?id=ai0DBAAAQBAJ>.

TEIXEIRA, T. et al. Service oriented middleware for the internet of things: A perspective. In: **Proceedings of the 4th European Conference on Towards a Service-based Internet**. Berlin, Heidelberg: Springer-Verlag, 2011. (ServiceWave'11), p. 220–229. ISBN 978-3-642-24754-5. Disponível em: <http://dl.acm.org/citation.cfm?id=2050869.2050893>.

VASCONCELOS, R. O. et al. Autonomous load balancing of data stream processing and mobile communications in scalable data distribution systems. **International Journal On Advances in Intelligent Systems (IARIA)**, Citeseer, v. 6, n. 3,4, p. 300–317, 2013. ISSN 1942-2679.

VASCONCELOS, R. O.; Nery e Silva, L.; ENDLER, M. Towards efficient group management and communication for large-scale mobile applications. In: **Pervasive**

Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on. [S.l.: s.n.], 2014. p. 551–556.

ZACHARIAH, T. et al. The internet of things has a gateway problem. In: **Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications**. New York, NY, USA: ACM, 2015. (HotMobile '15), p. 27–32. ISBN 978-1-4503-3391-7. Disponível em: <http://doi.acm.org/10.1145/2699343.2699344>.