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

**L.E. Talavera**[1], **M. Endler**[1], **S. Colcher**[1]

[1] Department of Informatics – Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Brazil

{lrios,endler,colcher}@inf.puc-rio.br

***Abstract.*** *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 paper 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.*

## 1. Introduction

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 the Internet of Things (IoT) at a global scale. In other words, extraction of meaningful information of data from tens of billions of sensors and actuators in (near) real-time is still a challenge. In particular, very few studies have focused on the Internet of Mobile Things (IoMT) where a Mobile Object (M-OBJ) may be any movable object (*Thing*) that carries sensors and/or actuators and has some means of wireless connectivity. In this context of general and unrestricted mobility of Smart Objects, different resources (e.g. embedded and wearable sensors) can become available and unavailable at any moment without previous warning.

Nowadays, 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, NFC, ANT+), the latter used by peripheral devices in the edge networks. Furthermore, according to [daCosta 2013], IPv6 does not solve all IoT problems because management, rather than addressing and routing, is the biggest challenge of IoT.

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, that allows an easy specification of patterns that represent different situations. CEP processing combines heterogeneous data from different sources to find patterns and generate high-level events (e.g. a crash or a fraud). Nevertheless, current CEP solutions are server centralized imposing a high energy cost for the communication.

Moreover, recent works propose the use of mobile devices (e.g. smartphones and tablets) as the propagator nodes in IoT, since they can gather the information of the environment and transmit it to the cloud. However, the huge volume of transmissions among nodes (mobile devices) and cloud would drain the device's battery 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 for sole cloud-based processing becomes less suitable [Saleh and Sattler 2013]. For this reasons, many works advocate that it is important to have as much in-network processing as possible, by performing some functions of filtering, aggregation and pre-processing over the data streams before sending any information to the cloud. In fact, recent works has shown that current mobile devices have the sufficient capacity to execute CEP, allowing them to perform such processing [Stipkovic et al. 2013].

In addition, CEP allows the re-configuration of its processing (deploy/un-deploy queries) on-the-fly. This characteristic is very useful for mobile devices since they can be found in different environments with different resources and thus different processing requirements. Our main contribution is to provide a method for dynamic processing of sensor data in the IoT-gateways and show the feasibility of using CEP in mobile devices by suggesting which kind of processing should or shouldn't be executed on them.

The remainder of this paper is structured as follows. In the next section we give an overview of the enabling technologies and main concepts of our approach towards IoMT. Section 3 discusses the energy consumption in mobile devices and its main concerns. In section 4, we present our approach to handle in-network processing and its general architecture. In section 5 we describe and discuss the results of the performance tests. In section 6 related work is discussed. Finally, in section 7, we draw concluding remarks and point to future work.

## 2. Enabling Technologies

This section presents the main concepts and technologies that form the foundations of our approach.

### 2.1. The Mobile Hub

The Mobile Hub (M-Hub) is a general-purpose middleware that enables mobile personal devices (smartphones or tablets) to become the propagator nodes (i.e. gateways to the Internet) for the simpler IoT objects or Mobile Objects (M-OBJ) (sensors/actuators) 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. As its first realization it is implemented for Android and uses Bluetooth Low Energy (*BLE*) as WPAN communication technology. BLE is being widely adopted for IoT and allows a low rate and very low-power communications.

#### 2.1.1. Main Components

The M-Hub consists of three services and one manager, all 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, humidity) 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 a certain time interval.

Messages that are going to be sent 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, network, or a manually entered (in case of a fixed location). The periodicity and duration of all of these three service's actions, is influenced by the device's current energy level (LOW, MEDIUM, HIGH), and is set by the **Energy Manager**, which from time to time sample's the device battery level and checks if it is connected to a power source. The communication among all the services is done using an EventBus[1], which is a Publish-Subscribe (PubSub) event bus optimized for Android that helps to decouple the components, and allows the inclusion of different services without many code modifications.

#### 2.1.2. 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, ANT+, Classic Bluetooth). To this end, it has a protocol for short-range communication with M-OBJs, based in two interfaces that help developers implement the different technologies uniformly. On the one hand, the *Technology Interface*

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

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 that occur on the different technologies and publish them for any interested component. The data published has the structure showed in the Figure 1.

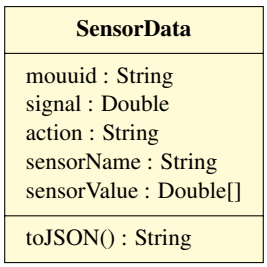


**Figure 1. Basic sensor data structure of the M-Hub**

The Technologies have to include an ID at programming time to be uniquely identified. This ID is also combined with the M-OBJ's mac address to form the Mobile Object Universally Unique Identifier (MOUUID) which helps developers to differentiate all the M-OBJs, even if they are communicating with the M-Hub under different WPAN technologies. As an example, the module for BLE is defined with the ID 1, and a MOUUID under such technology could be 1-B4994C64BA9F.

## 2.2. Complex Event Processing

Complex Event Processing (CEP) is a software technology that helps to correlate high volumes of incoming data items which are considered as events happening in the external world such as changes in a company's stock value, or a simple change in the temperature. A CEP solution provides capabilities of filtering, aggregation, correlation and analysis over continuous streams of data. It can detect high-level events that represent different situations and can trigger actions such as notifications or interactions with business processes. In fact, aggregation can be described, as combining and transforming lower level events (e.g. changes in the temperature and humidity) into higher level events (e.g. a fire, a credit-card fraud) to respond upon them as soon as possible.

In CEP, incoming messages runs through a set of continuous queries to produce derived events or sets of data (complex events), this process is performed by an entity called **Event Processing Agent** (EPA). An EPA may perform different kinds of computation on events, such as filtering and aggregation, in order to detect patterns of events. A set of EPAs and the channels they use to communicate form what is called an Event Processing Network (EPN), that 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, 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, 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 queries are a set of patterns that describe the situations of interest presented in the form of combinations of 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 when several devices are within a certain distance of each other.

**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 the cloud.

There are many flavours technologies and languages for CEP, but we focus on Esper[2], since it is an open-source component available under GNU GPL license and one of the most used CEP engine[3]. Esper events allow a rich domain object representation, since it supports all aspects of object-oriented design as well as dynamic typing, while 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[4]. Moreover, different from other CEP technologies it has been successfully ported to Android with the name of *"Asper"*[Eggum 2014]. This allow us to include it as an additional service for the M-Hub.

Esper uses a declarative Event Processing Language (EPL) that derives many properties from the SQL standard, to define the event processing rules used to detect the patterns over the streaming data. An event-processing engine analyzes the event streams and executes the matching rules 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].

## 3. Energy Consumption in Mobile Devices

Mobile devices are general-purpose computers that get their power to process from batteries which have a limited amount of energy. Table 1 shows some of their main characteristics. Many of such characteristics evolved at an exponential rate except for the battery capacity, limiting its use to day-to-day tasks.

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 event more energy than the CPU. For example, an often cited power budget mobile phone

[2]http://www.espertech.com/esper/documentation.php

[3]http://www.espertech.com/esper/faq_esper.php

[4]http://www.espertech.com/esper/faq_esper.php#comparison

[5]Energy Consumption: Modeling and Optimization. 2014

| | 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 1. Evolution of mobile phones**[5]

streaming a 384 kb/s video has 1.2 W power drain caused by the network interface, 1 W for the display, and 0.8 W for the CPU and memory operations [Tarkoma et al. 2014]. Besides, mobile devices possess several sensors that also consume a considerable amount of energy, such as GPS, accelerometer, and gyroscope. Thus, it is important to understand where and how the energy is used.

Some experiments presented in [Pathak et al. 2012] confirms that most applications spend a large amount of energy in I/O components such as Wi-Fi, 3G and GPS. With the increasing popularity of mobile internet services, wireless data transmission is becoming a major energy consumer on mobile Internet devices [Tarkoma et al. 2014]. For example, according to a review from AnandTech[6], the Motorola Moto X (2013) device can be idle for up to 576 hours, but 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 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].

Wireless communication interfaces such as Wi-Fi and Bluetooth have a dynamic power demand. They usually have three states: idle, transmitting and receiving which can be categorized in two different duty cycles, idle and active (transmissions and receptions). The power consumption on the active state is significantly higher than on the idle state. The overall power can be optimized by making active states shorter, so mobile devices could spend most of their time in idle states [Tarkoma et al. 2014] consuming less energy.

## 4. Mobile Processing and Dynamic Deployment

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). Most of the current IoT gateways only care about the direct transmission of data acquired from the physical environment to the cloud[Pereira et al. 2013]. But differently from the traditional Internet where data consumption is primarily discrete[Billet and Issarny 2014], IoT deals with a continuous processing of data produced by sensors and actuators embedded into M-OBJs. Furthermore, communication with the cloud is an expensive operation in terms of energy consumption and network bandwidth which is critical regarding 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 a sudden increase in the temperature, or an elevated heart rate.

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

### 4.1. MEPA Service

Given the need to reduce bandwidth consumption in order to avoid bottlenecks, IoT should completely decentralize its processing. Nowadays mobile devices have enough processing power to perform part of the IoT data processing close to the data sources (M-OBJs). In this context, we believe that Complex Event Processing (CEP) can help to evaluate streams of sensor data by checking for certain data patterns. Some of these patterns could be their timestamps and order of occurrence expressed as queries (e.g. consecutive temperature events with a value increase). 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.

We have included Asper CEP engine [Eggum 2014] in the MEPA Service to process any incoming sensor data. It keeps a record of the running CEP rules, and allows to start and stop them on-the-fly. The sensor data that arrives from M-OBJs are defined as a primitive event type, which contains the sensor's names and their respective values (see Figure 1). As shown in the Figure 2, the MEPA Service is subscribed to all the messages that are sent from the S2PA and Connection services, since the former collects the data from the M-OBJs, and the latter receives commands from the cloud to modify the behavior of the MEPA Service itself (e.g. deploy a new CEP rule). Every time an event pattern is detected (which leads to the generation of a new complex event), it will be published to any interested component that could be the Connection service, or another rule in the MEPA Service.

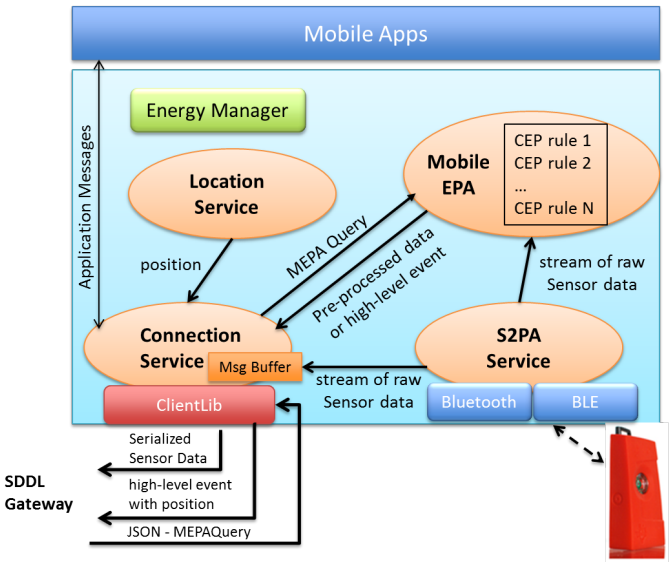


**Figure 2. Architecture of the M-Hub with the MEPA Service**

CEP rules frequently implement complex correlation functions that sometimes can be divided into different sets of sub-queries. These sub-queries can perform certain steps of the correlation separately and provide the results to a root CEP rule in the cloud. For example, a root CEP rule could be a computational-intensive processing, while the rest of sub-queries could be simple correlations on collected sensor data [Schmidhäuser 2014].

Since mobile devices collect most of the sensor data, processing these 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.

### 4.2. Dynamic Deployment of CEP Rules

M-Hubs can opportunistically collect data in different environments (e.g. a hospital, a school) 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, a M-Hub in a hospital could interact with temperature, heart rate and accelerometer sensors attached to the patients. Thus it could detect events such as irregular heart rates, fever, or even the fall of a person. A completely different scenario could be described for sensors located in the street or a the beach that could help to detect high pollution/radiation levels, or the average environmental temperature. Due to this flexibility requirement we included the capability to add/remove CEP rules to/from the M-Hub remotely using the communication link with the cloud. Depending on the location, CEP rules in the MEPA Service could be swapped to fit the M-Hub's current usage context. A software framework implemented around the MEPA Service translates commands (JSON messages) received through the Connection Service into specific instructions to modify the CEP rules executing in the M-Hub. Such instructions are encapsulated as a ***MEPAQuery*** object that contains information about the type of the request (i.e. add/remove/start/stop/clear) and the label used to identify the EPL queries. In the case of an `add` request type, the EPL query will be also included as a string. Wrong rules specifications or any other exception that could happen generates an error message that is sent to the server.

## 5. Performance Experiments and Results

This section describes the experiments and measurements realized over the CEP processing in mobile devices. We tested a prototype running different kinds of CEP rules 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. For all the experiments we used a Motorola Moto X handheld (model 2013) running Android 4.4.4. Both Android runtimes were tested, Dalvik and ART[7]. The devices used as M-OBJs were off-the-shelf SensorTags[8], and the WPAN technology used for communication was BLE. BLE in Android limits the simultaneous connections to six devices. The type of WLAN used is IEEE 802.11bgn.

To quickly connect with the M-OBJs and avoid that other components affect the battery life, 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. Each test sampled the processing and the cloud communication for one hour with different sets of connected M-OBJs (1, 3, and 6). The average size of the messages was 200 bytes for events, and 300 bytes for sensor data. Finally, 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).

[7]ART and Dalvik - https://source.android.com/devices/tech/dalvik/

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

**Table 2. No Processing - Energy and bandwidth consumption**

| **Dalvik** | 1 | 3 | 6 |
|---|---|---|---|
| Bandwidth (kb) | 7108 | 20635 | 42204 |
| Mean time (s) | 578.00 | 512.57 | 536.17 |
| Std. Deviation | 34.77 | 52.32 | 7.84 |
| **ART** | 1 | 3 | 6 |
| Bandwidth (kb) | 7262 | 19551 | 37440 |
| Mean time (s) | 727.00 | 629.39 | 551.60 |
| Std. Deviation | 21.85 | 48.18 | 13.35 |

### 5.1. Filtering Rule

We tested how much energy and bandwidth we could save with a simple CEP filtering rule. Hence, we created a rule 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. As we can see in Table 3, even when we reduced the total bandwidth usage, the energy consumption is almost the same as sending all the information to the cloud (see Table 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 3. Filtering rule - Energy and bandwidth consumption**

| **Dalvik** | 1 | 3 | 6 |
|---|---|---|---|
| Bandwidth (kb) | 3146 | 7178 | 12682 |
| Mean time (s) | 593.60 | 533.72 | 488.85 |
| Std. Deviation | 39.59 | 38.72 | 31.00 |
| **ART** | 1 | 3 | 6 |
| Bandwidth (kb) | 3210 | 7045 | 12865 |
| Mean time (s) | 670.40 | 593.00 | 530.00 |
| Std. Deviation | 32.12 | 30.36 | 21.48 |

### 5.2. Aggregation Rules

In this experiment we intend to show the energy and bandwidth consumption (see Table 4) with some rules that included other processing than just filtering, and used different CEP 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 process in a bulk operation all the data items contained.

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 only remove the oldest events [Eggum 2014].

Both window types can be classified as time- and count- based. Time-based windows were used for the tests with one minute of length. The CEP rules filter the data to process their average value. Only events that contained the average value were sent to the cloud. The results show that sliding windows don't reduce the active states of the network interfaces. However, jumping windows where the frequency of outbound events depends on the window size, hence bigger windows will decrease the number of active states.

**Table 4. Aggregation rules - Energy and bandwidth consumption**

| | Dalvik | | | ART | | |
|---|---|---|---|---|---|---|
| Jumping | 1 | 3 | 6 | 1 | 3 | 6 |
| Bandwidth (kb) | 71 | 102 | 69 | 81 | 105 | 67 |
| Mean time (s) | 807.50 | 760.67 | 719.80 | 918.00 | 840.67 | 824.25 |
| Std. Deviation | 44.58 | 19.01 | 34.50 | 26.89 | 21.55 | 22.43 |
| Sliding | 1 | 3 | 6 | 1 | 3 | 6 |
| Bandwidth (kb) | 2519 | 6131 | 11355 | 3261 | 5485 | 10240 |
| Mean time (s) | 576.33 | 553.83 | 490.14 | 705.67 | 607.20 | 582.00 |
| Std. Deviation | 49.31 | 45.93 | 26.87 | 27.43 | 32.76 | 26.28 |

### 5.3. Rules with Heavy Processing

In the following experiments we tested a more complex processing, in which the magnitude of the accelerometer sensors [9] was calculated in a window of time. 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 configurations as the previous experiment were used.

**Table 5. Heavy processing rules - Energy and bandwidth consumption**

| | Dalvik | | | ART | | |
|---|---|---|---|---|---|---|
| Jumping | 1 | 3 | 6 | 1 | 3 | 6 |
| Bandwidth (kb) | 78 | 148 | 62 | 74 | 105 | 67 |
| Mean time (s) | 721.20 | 672.80 | 649.20 | 951.46 | 862.67 | 728.5 |
| Std. Deviation | 53.10 | 57.05 | 31.73 | 120.97 | 40.46 | 29.05 |
| Sliding | 1 | 3 | 6 | 1 | 3 | 6 |
| Bandwidth (kb) | 3026 | 4933 | 10630 | 2472 | 5330 | 10240 |
| Mean time (s) | 691.00 | 519.00 | 480.14 | 682.00 | 566.60 | 491.20 |
| Std. Deviation | 32.79 | 18.65 | 31.14 | 53.76 | 64.65 | 11.61 |

Rules 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. Nevertheless, queries like the ones presented in the former experiments (Table 5) can still be executed if the frequency of outbound events is low enough to compensate the energy consumption for the processing.

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

### 5.4. Two Rules (Aggregation and Pattern Match)

Here we tested two different rules 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 rule 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 rules executing at the same time. Table 6 presents the energy and bandwidth measurements. Results show that even if we have two rules, if they have a low rate of outbound events, it is still possible to reduce the energy consumption in comparison with Table 2.

**Table 6. Two rules - Energy and bandwidth consumption**

| **Dalvik** | 1 | 3 | 6 |
|---|---|---|---|
| Bandwidth (kb) | 258 | 389 | 298 |
| Mean time (s) | 750.25 | 733.25 | 789.50 |
| Std. Deviation | 26.86 | 20.22 | 52.67 |
| **ART** | 1 | 3 | 6 |
| Bandwidth (kb) | 270 | 326 | 264 |
| Mean time (s) | 895.00 | 814.67 | 867.00 |
| Std. Deviation | 16.09 | 37.00 | 10.44 |

### 5.5. Discussion

The experiments show that in most of the cases the MEPA service can significantly decrease the energy and bandwidth consumption compared with sending all the data to the cloud. Moreover, the results indicate that CEP jumping windows rules 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.

Nevertheless, it is also possible to use the output of the CEP rules that use sliding windows as the input for another rule $X$ in the MEPA Service. In such case, if the rule $X$ is the one that communicates with the cloud and can significantly reduce the outbound frequency, energy consumption can still be reduced. Moreover, 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 rules, 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.

We can conclude that the impact of having one or more rules depends on the frequency of detected events. 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 propose the use of average Things (moderately powerful Things) as the enablers of the Internet of Things (IoT). These devices could act as temporary IP routers and opportunistic context providers for simpler Things. Nevertheless, only recent works concern about the amount of transferred data and energy consumption on the IoT-gateways that are usually energy-constrained devices. [Billet and Issarny 2014] let the sensor networks perform as much in-network processing as possible before sending any data to a proxy or the cloud. However, in-network processing should be restricted to low-latency operations and depends on the available resources in the current location of the IoT-gateways.

[Stipkovic et al. 2013, Dunkel et al. 2013] propose the use of Complex Event Processing (CEP) directly in mobile devices, and avoids the use of the cloud. They reduce transmission traffic, save resources, and private data remains only on the mobile devices (e.g. GPS). However, it can't handle situations where a global view of the data is required, for example aggregation of data obtained from different mobile devices in different geographic locations. [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 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. Their proposal only covers the architecture and representation of the queries. They don't consider a dynamic deployment of CEP rules, nor have any evaluation (energy/bandwidth) over the system.

[Chen et al. 2014] propose an architecture for distributed CEP to meet the needs of real-time streaming of information processing. Their approach is based on static environments where gateways are always connected to the same devices. It doesn't provide 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] propose a system that provides an adaptation of CEP Rules. To achieve this, they propose a context-aware (resources, situation) system, which will dynamically deploy/un-deploy rules to the mobile devices or server. By doing this, better recommendations will be send, better actions will be taken and battery will be saved, since it won't be necessary to have unused rules running on the mobile devices. The problem is that they still send all the information 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], they make use of the DDS (Data Distribution Service) to exchange data/events (sensor data) among nodes, and use CEP (Complex Event Processing) to create useful information to the users. 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 rules can be changed depending on the requirements of the applications. Nevertheless, they don't consider that the mobile devices can act as data sources and node processors at the same time. Saleh et al. [Saleh and Sattler 2013] solution proposes to distribute CEP rules to the network as a graph, where each node will communicate with each other using

pub/sub. A sensor catalog makes possible to find information about the different sensor devices, such as the computational power, memory size, communication cost and available sensors that will be used at the moment of decide which CEP rules will go to which nodes, or to the server.

Although there are many approaches that try to include CEP in mobile devices, they don't explore its capacity for re-configuration. Moreover, none of the previous works provides a benchmark about the energy/bandwidth consumption of the use of CEP in mobile devices, which are some of the most important aspects to consider when including processing in IoT sensor-gateways.

## 7. Conclusion

We have presented an IoMT sensor gateway with dynamic local processing of sensor data using conventional smartphones. In fact, the popularity and characteristics of CEP, such as its on-the-fly reconfiguration and fast detection of events, led us to use it for the local processing on mobile devices. Using a prototype implementation and BLE SensorTag devices we did performance experiments 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 initial experiments are quite encouraging and show that CEP rules that represent events with a low frequency of occurrence are more adequate for mobile devices.

In spite of the encouraging preliminary results, we are aware that our current prototype is only "scratching the surface of IoMT", and much interesting research, software development and applications can be derived from this work. 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, in case a M-Hub is unable to establish an Internet connection. Moreover, we also plan to study 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.

### Acknowledgements

The Mobile Hub project is supported by the *PUC-Rio Microsoft Open Source Alliance*, and partially supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

## References

Billet, B. and Issarny, V. (2014). Dioptase: a distributed data streaming middleware for the future web of things. volume 5. Springer London.

Chen, C. Y., Fu, J. H., Sung, T., Wang, P.-F., Jou, E., and Feng, M.-W. (2014). Complex event processing for the internet of things and its applications. In *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pages 1144–1149.

daCosta, F. (2013). *Rethinking the Internet of Things*. Apress.

Dunkel, J., Bruns, R., and Stipkovic, S. (2013). Event-based smartphone sensor processing for ambient assisted living. In *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, pages 1–6.

Eggum, M. (2014). *Smartphone Assisted Complex Event Processing*. dissertation, University of Oslo.

Govindarajan, N., Simmhan, Y., Jamadagni, N., and Misra, P. (2014). Event processing across edge and the cloud for internet of things applications. In *Proceedings of the 20th International Conference on Management of Data*, pages 101–104. Computer Society of India.

Kim, D., Lee, J. H., Cheol, R., Jo, J. C., and You, Y. D. (2009). 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*, pages 645–646.

Pathak, A., Hu, Y. C., and Zhang, M. (2012). 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*, pages 29–42.

Pentikousis, K. (2010). In search of energy-efficient mobile networking. In *Communications Magazine, IEEE*, pages 95–103.

Pereira, P. P., Eliasson, J., Kyusakov, R., Delsing, J., Raayatinezhad, A., and Johansson, M. (2013). Enabling cloud connectivity for mobile internet of things applications. In *Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, SOSE '13, pages 518–526, Washington, DC, USA. IEEE Computer Society.

Saleh, O. and Sattler, K.-U. (2013). Distributed complex event processing in sensor networks. In *Mobile Data Management (MDM), 2013 IEEE 14th International Conference on*, volume 2, pages 23–26.

Schmidhäuser, S. (2014). *Dynamic Operator Splitting in Mobile CEP Scenarios*. PhD thesis, University of Stuttgart.

Stipkovic, S., Bruns, R., and Dunkel, J. (2013). Pervasive computing by mobile complex event processing. In *e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on*, pages 318–323.

Stojanovic, N., Stojanovic, L., Xu, Y., and Stajic, B. (2014). Mobile cep in real-time big data processing: Challenges and opportunities. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, pages 256–265. ACM.

Talavera, L., Endler, M., Vasconcelos, I., Vasconcelos, R., Cunha, M., and Da Silva E.Silva, F. (2015). 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*, pages 123–128.

Tarkoma, S., M., S., E., L., and Y., X. (2014). In *Smartphone Energy Consumption: Modeling and Optimization*.