LE-STREAM: A LATENCY AND ENERGY-AWARE FRAMEWORK FOR DATA STREAM PROCESSING IN THE INTERNET OF THINGS

Egberto Armando Rabello de Oliveira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Marta Lima de Queirós Mattoso
Flavia Coimbra Delicato

Rio de Janeiro
Novembro de 2021

LE-STREAM: A LATENCY AND ENERGY-AWARE FRAMEWORK FOR DATA STREAM PROCESSING IN THE INTERNET OF THINGS

Egberto Armando Rabello de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Marta Lima de Queirós Mattoso
Flavia Coimbra Delicato

Aprovada por: Prof. Marta Lima de Queirós Mattoso
Prof. Flavia Coimbra Delicato
Prof. Claudio Miceli de Farias
Prof. Paulo de Figueiredo Pires
Prof. Markus Endler

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2021

Oliveira, Egberto Armando Rabello de

LE-Stream: a Latency and Energy-Aware Framework for Data Stream Processing in the Internet of Things/Egberto Armando Rabello de Oliveira. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIV, 61 p.: il.; 29, 7cm.

Orientadores: Marta Lima de Queirós Mattoso

Flavia Coimbra Delicato

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 52 – 61.

1. Internet of things. 2. Data stream processing. 3. Edge computing. 4. Adaptive sampling. 5. Active node selection. I. Mattoso, Marta Lima de Queirós *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

"Rien dans la vie n'est à craindre, tout doit être compris. C'est maintenant le moment de comprendre davantage, afin de craindre moins."
(Marie Courie)

I dedicate this work to my
beloved girls Bella and Michelle.
Without their unconditional love
and support I would NEVER
have made it this far!

Acknowledgements

It is really hard to find words to describe how I feel about getting this far. It is equally impossible to name each person, word or gesture that directly or indirectly influenced this journey. However, I am trying to do my best to record my sincere thanks and I hope I am not being unfair to those I did not name explicitly.

I must start with whom inspired me with her legitimate passion for Science and who made me believe I was capable of persuing a MSc degree. Being my inspiration (in all senses) would be enough to guarantee the first place in my list, but this is not everything. “full support" is still to small to describe how much she was dedicated to my success. She was my eyes when I was not able to see the end of the tunnel and also the only one who never doubted that I would make it to the end. Nothing I say or do in this or any other lifetime can fully express my gratitude for my wife, soul mate and eternal love: Michelle.

I also thank the one who was born along the way and became my inexhaustible source of joy and willpower to face each setback. I fell so many times but I got up and moved on, always thinking about being an example to my sweet little Bella, my reason to exist!

How not to mention the greatest warriors I have ever seen: my beloved parents Earli and Nadson. I am just a mirror that reflects the greatness of those who created me without measuring efforts and, above all, with unrestricted love.

My little sister Alessandra, who is small only in height, but one of the most amazing and strongest women I ever met. Thank you for being my partner, guardian and number 1 fan since day 1.

To Giovanna, Sophie, Gabriela, Ricardo and Alice: jewels that have been entrusted to me and that I will protect and love until my very last breathe. You inspire me to always be the best version of myself. To my heart brothers Guilherme, Juliana, Rafael, Marcele and Rodrigo for the immense honor of being chosen as the godfather of your greatest treasures.

To my blood family whose names it would not be possible to list and also to the family that welcomed me with all the love and affection, here represented by Tânia, José Carlos and Jennifer.

To my big brother Rui for being not only a friend but a real mentor when needed

and also to all my family from the mountains he represents.

To all my teachers and professors for helping me to acquire the only good that cannot be taken away: knowledge.

To professors Flavia and Marta, my advisors in the truest sense of the word. Tireless, dedicated and extremely helpful. I think I have an idea of how difficult it must be to guide such a stubborn“old donkey" like me... Thank you for not giving up on me!

To professor Paulo Pires for the valuable teachings in the IoT labs and the very insightful feedbacks on my MSc seminar together with professor Artur Ziviani, who left us prematurely and certainly represents a huge loss for Science.

To professor Atslands for the very important contribution to the success of this work.

To professor Paulo Eustáquio for the recommendation and support and who will always be one of my greatest academic references.

To my true friends from CPII for understanding my absence and my eventual bad mood: Bernardo, Mariah, Marcelo, Theo, Marianna, Marília, Livia, Maíra, Diego, Douglas, Fernando, Julia, Luana, Clarissa, Steffanie, Bruno, Gabriel and so many others that it's hard to count...

To my partners from Valia, Marcus, Vanessa, Eduardo, Veloso, Bruno, Ismael, Gustavo, Martelo, Enrique, Zanchetta, Claudia and Analu for supporting and covering my absence at the most inappropriate class times.

To my colleagues from Siemens for welcoming me in a new home on the other side of the Atlantic ocean.

Last but not least, I thank God. Not only for the opportunity of going back to the University after a decade, at the largest engineering postgraduate institute in Latin America, but also for filling my path with all these people who gave me strength and support to overcome so many obstacles along the way.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

LE-STREAM: UM FRAMEWORK CIENTE DE LATÊNCIA E ENERGIA PARA PROCESSAMENTO DE FLUXOS DE DADOS NA INTERNET DAS COISAS

Egberto Armando Rabello de Oliveira

Novembro/2021

Orientadores: Marta Lima de Queirós Mattoso
Flavia Coimbra Delicato

Programa: Engenharia de Sistemas e Computação

O processamento de dados em IoT é um desafio devido à sua natureza dinâmica e heterogênea e à enorme quantidade de dados envolvidos. Dados gerados por sensor sofrem de problemas de incerteza e inconsistência, que podem afetar sua precisão. Vários aplicativos IoT são sensíveis à latência, exigindo processamento de dados rápido. Por fim, como os dispositivos IoT costumam ser alimentados por bateria, as tarefas de processamento devem ser executadas de maneira eficiente em termos de energia. Portanto, os desafios no processamento de fluxo de dados englobam três dimensões: precisão, latência e energia. Esta dissertação de mestrado propõe o LE-Stream, um framework para apoiar o processamento de fluxo de dados para sistemas IoT que aborda estas três dimensões em conjunto. O paradigma de computação de borda é utilizado para aproximar o processamento de dados das fontes de dados, minimizando a latência. Uma nova amostragem adaptativa colaborativa combinada com um modelo de previsão de dados em duas etapas reduz o consumo de energia dos dispositivos sensores sem comprometer a precisão dos dados de saída. Um esquema de seleção de nós ativos melhora a distribuição da carga de trabalho entre os dispositivos, abordando também a dimensão da energia, promovendo uma degradação harmoniosa dos seus recursos computacionais.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

LE-STREAM: A LATENCY AND ENERGY-AWARE FRAMEWORK FOR DATA STREAM PROCESSING IN THE INTERNET OF THINGS

Egberto Armando Rabello de Oliveira

November/2021

Advisors: Marta Lima de Queirós Mattoso
Flavia Coimbra Delicato

Department: Systems Engineering and Computer Science

Data processing in IoT is challenging due to its dynamic and heterogeneous nature, and the massive amount of generated data. Sensor data suffers from uncertainty and inconsistency issues, that can affect its accuracy. Several IoT applications are time sensitive, requiring fast data processing. Finally, as IoT devices are often battery powered, processing tasks must be performed in an energy-efficient way. Therefore, there are challenges in data stream processing concerning three dimensions: accuracy, latency and energy. We propose LE-STREAM, a framework to support the data stream processing for IoT systems which jointly addresses these dimensions. It leverages edge computing to bring the data processing closer to the data sources, thus minimizing latency. A novel collaborative adaptive sampling combined with a two-step data prediction model reduce the energy consumption of devices without compromising data accuracy. An active node selection schema improves the workload distribution among devices, also tackling the energy dimension by promoting a graceful degradation of devices resources.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet of things (IoT) is transforming the internet, enabling the communication between every kind of object (things) and creating a vision of "anytime, anywhere, any media, anything" communications [1]. Initially, IoT was mainly obtained by the use of RFIDs, nowadays such a concept has evolved to a broader view that refers to the interconnection of sensors, actuators, smart objects, and wireless sensor networks (WSN)[2]. The growing number of sensors and smart devices led to an explosion of volume, variety and velocity of generated data, empowering a new way of value creation to people and corporations [3]. The processing of these "firehoses" of data from existing and emerging applications poses several challenges and brings novel research opportunities.

The challenges involved in IoT data stream processing may be analyzed at least from two dimensions: (i) the data itself (generated by heterogeneous, distributed and often constrained devices), and (ii) the data processing, i.e. the core activities from the data acquisition to the production of high level knowledge.

Regarding the nature of the data, IoT devices/sensors generate, possibly in a continuous way, a huge amount of data, typically consisting of time-series values, which are sampled over a specific time period, thus characterizing a data stream [4]. Often, there is no control over the order or frequency of streamed data, which is transient or non-persisted. The input rate of a data stream is unpredictable and bursty in nature, ranging from a few bytes to several gigabits per second. In addition, the data is highly heterogeneous, as it is generated by multiple types of devices, in different formats and to feed a wide range of applications, also heterogeneous. Besides the potentially massive volume of data, an IoT environment is also characterized by high dynamism and volatility. In many IoT applications, such as traffic accident monitoring or river flooding prediction, the potential value of data depends on its timely processing, under strict response time requirements. Otherwise, the processing results and actions become less valuable or even worthless. Finally, quality-related data features also need to be considered. According to QIN

et al. [5], quality awareness in IoT considers the following features: (i) uncertainty, (ii) ambiguity and inconsistency, (iii) incompleteness, and most of them are a direct consequence of the data being produced by sensors. Sensors are fail-prone devices. Information and decisions derived from raw data generated by sensors will also be subject to failure [6]. Therefore, identifying errors/inconsistencies on a sensor generated data stream is crucial to improve the accuracy of the data being processed. These errors/inconsistencies are called outliers, which are readings considered outside the regular state of the data being collected. Data points that differ significantly from others in a data set can represent either errors or events of importance to the application [4].

Regarding the data processing, the demand for computational resources capable of processing large volumes of data has historically been an obstacle for creating high volume and/or high speed data processing solutions [7]. Because of the huge availability of resources offered by cloud computing platforms, cloud-based approaches are widely adopted in IoT systems. The data is pushed to the cloud to be processed and the outcome is delivered back to the local system. However, the internet backbone is not always able to meet strict response time requirements to transport a huge amount of data coming at a high speed. This creates a communication bottleneck that leads to proposing non-cloud based alternatives, to handle and process IoT generated data streams[8].

A promising approach recently emerged is Edge Computing [7]. It consists of bringing the data processing activities physically closer to the data sources. Edge computing is potentially useful and has been adopted in several domains such as smart buildings, healthcare, autonomous vehicles, and environmental monitoring. In these applications, data is processed by an edge device such as a smart gateway, to extract meaningful information from it and take necessary actions immediately [8], thus preserving the usability of time-sensitive data. Therefore, with the support of edge computing, some of the activities of a data stream processing workflow can be performed by devices at the edge of the network. Other activities, more demanding in terms of processing, may continue to be carried out in the cloud. Still others can be performed on the sensor device itself, that is, on the data sources since, although restricted in terms of CPU and memory, such sensors are capable of performing less complex processing. In this context, another challenge arises related to data stream processing in IoT, which consists of using the available resources in a rational way. In addition to the restricted processing and communication capabilities, several sensor devices are powered by non-rechargeable batteries. Keeping the sensors working as long as possible is a major challenge in all sensing-based systems, and it has been extensively investigated in the WSN community.

In general, a basic strategy for preserving energy in WSNs consists of: (i) keep-

ing the nodes in hibernation as long as possible and (ii) reducing the data traffic in the network as much as possible. One technique called Adaptive Sampling [9] simultaneously tries to do both by varying the interval between samplings according to the behaviour of the sensed data. If, on the one hand, adaptive sampling techniques are very efficient in reducing power consumption in a WSN, on the other hand they are usually not suitable to applications requiring sampled data to be provided continuously (as a stream). However, this problem can be minimized when Adaptive Sampling is combined with a data prediction model to compute future sensor readings based on past samples.

In WSNs, data prediction techniques usually maintain two instances of a prediction model, one residing at the sink node and the other at the sensor, in the so-called Dual Prediction Scheme (DPS)[10]. Each sensor node runs a model that estimates the next measurement. The sink (or base station) runs exactly the same models for each sensor in the network and makes the same predictions, thus reducing the need of data transmissions from sensors to sink, as long as the difference between the values predicted by the sink and the actual transmitted (from times to times) is below a threshold.

By using a data prediction model, queries generated by the application can obtain data continuously (predicted values), without having to wait for the real data, which will be transmitted at a reduced rate due to the use of the adaptive sampling approach. Yet, despite being able to provide data continuously, DPS is not a good fit to sample physical phenomena exposed to significant sudden and unpredictable variations. This is the case in event detection systems where, after a period of stability, sudden and expressive variations of a physical variable may indicate the occurrence of an important event. In general, DPS methods are more suited to sample predictable physical phenomena, with more smooth variations in time [10]. Since it is very difficult to define a data prediction model capable of tackling sudden and unpredictable events, in this work we propose a novel adaptive sampling approach so that it is possible to capture such events.

Usually, adaptive sampling techniques are based on a centralized (also called synchronized) approach to determine the time intervals between samples taken by a sensor [11]. On these synchronized approaches, the time intervals are calculated from time to time and they are used for all sensors involved in the sensing task. Thus, it is possible that at a given moment of time most or all of these sensors are“sleeping". If a sudden variation occurs when the sensors are sleeping at the same time, this event is not captured. This is a problem we can call “blind window". The size of these blind windows can vary depending on the scenario, adopted technique and its parameters. In a data stream processing scenario, where sudden variations are very common and can be highly relevant to the applications, it is necessary to reduce blind

windows as much as possible. To achieve this goal, we propose a collaborative and non synchronized adaptive sampling strategy, where the time interval is calculated for each sample received and sent back to the sensor node that sampled it. This non synchronized (or decentralized) approach reduces blind windows by distributing different time intervals to each sensor node, preventing many or all of them from sleeping at the very same time.

Despite addressing the problem of blind windows, desynchronizing activities of sensor nodes, in turn, can make the workload of sensor nodes in the network to be uneven, generating what is known in the WSN and IoT fields as a load balancing problem [12]. Among the approaches commonly used to tackle this problem, we highlight the topology control technique known as Active Node Selection [13]. The main goal of an active node selection service is to achieve a more even energy consumption between sensors over time, thus preventing the premature death of overloaded nodes [14]. Given the importance of this aspect, in this work it is also considered as an important power consumption requirement to be met.

The importance of meeting strict response time requirements and of providing accurate data can be critical to many IoT applications, such as traffic accident monitoring or fire suppression system. The slight difference of seconds as well as a tiny margin of error on a sensor reading can be preponderant in the real value of an IoT application's outcome. In addition, increasing the overall lifespan of an IoT system, which is often dependent on devices that are battery powered, is a major issue. Overloading these devices with processing or communication shortens their lifespan. Thus, there is a big challenge to overcome: to meet strict response time requirements and reduce energy consumption of IoT devices (preferably in a balance way among the nodes) without compromising data accuracy.

Relevant works addressing separately strict response times, power consumption and data accuracy requirements can be found in the literature, but to the best of our knowledge, no solution tackling all these three concerns together has been found so far. This makes it difficult to deploy solutions that can efficiently respond to strict response time events in power-constrained environments, such as a forest fire suppression system, a malfunction detection system on small ships, etc. Developing an IoT data stream processing framework which adapts and combines some of these isolated techniques to tackle strict response times, energy consumption and data accuracy together is the main purpose of this work.

In this work we propose LE-Stream, a framework to support activities of a data stream processing workflow in IoT systems. The framework aims at addressing the requirements of strict response times, power consumption and data accuracy together. We adopt the edge computing paradigm to deal with the network bandwidth vs. data production bottleneck, allowing for applications with strict response

time requirements. We use adaptive sampling to reduce the network traffic, and, as a consequence, the power consumption of the sensor nodes. An active node selection schema is responsible for improving the workload distribution among sensor nodes. A data prediction model identifies and removes outliers producing an accurate aggregate output. The main contributions of the proposed framework are:

- To provide an energy-aware data gathering component with adaptive sampling to reduce the network traffic and, as a consequence, the power consumption of the sensor nodes;
- To present an active node selection service capable of improving the workload distribution among sensor nodes and mitigating the load balancing problem generated by the non synchronized adaptive sampling approach.
- To develop a data prediction model which takes readings from multiple sensor nodes over a short predefined window as inputs, applies a density-based clustering algorithm to identify and remove outliers and produces an accurate aggregated output.

The major benefit expected by adopting the proposed framework is being able to deploy long running strict response time processing systems on remote outdoor environments such as forests, open fields and watercrafts. In such environments, there is no access to continuous sources of electricity thus requiring the use of batteries, solar panels or other types of limited power sources.

In addition to this introduction, the remainder of this work is organized as follows. Chapter 2 presents the fundamental concepts and terminology involving IoT and data stream processing. Chapter 3 shows a review on relevant work published tackling strict response time requirements and energy-awareness in data stream processing and IoT. Chapter 4 introduces our framework LE-Stream. Chapter 5 presents an implementation of a workflow using LE-Stream with the purpose of evaluating the proposed framework in terms of impacts on data accuracy, efficiency on reducing energy consumption and data stream processing speed. Finally, Chapter 6 concludes this dissertation by summarizing its findings and sharing proposals for future work.

Chapter 2

Background Concepts

This chapter presents key concepts for understanding the solution proposed as LE-Stream in Chapter 4. First, Internet of Things (IoT) is described along with its system architectures and application protocols. Next, wireless sensor networks (WSN) and their strong relationship with IoT are presented together with the challenges related to energy conservation in these types of networks. Also, cloud, fog and edge computing paradigms are presented, as well as their role and importance in IoT system architectures. Finally, Data Stream Processing (DSP) and its challenges regarding data outliers and outlier detection are discussed in the context of IoT.

2.1 Internet of Things (IoT)

The IoT is a paradigm that leverages the next wave of internet evolution, extending communication between computers to every kind of object (things) and enabling the integration of heterogeneous technologies [1]. The increasing availability of sensors, mobile phones and other devices led to an explosion in the volume, variety and speed of the data generated, requiring analyzes of some kind to add value to such data. This phenomenon is often referred to as Big Data because of the challenges it imposes on existing infrastructure with regard to, for example, data storage, transfer and processing [3].

In this context, new generations of applications are emerging in various domains. In the field of supply chain and urban transportation, smart vehicles and autonomous cars have been dictating new modes of operation. In healthcare, active patient monitoring through wearable devices encourages the adoption of healthy habits and enhances preventive medicine. In addition, when continuous monitoring data is associated with historical patient data, more accurate diagnostics and risk situations can be detected in advance. Under the domain of smart buildings, smart homes, even entire cities are already a reality and transform communities and society, contributing to more comfortable, safe and sustainable environments.

Figure 2.1: An IoT overview emphasizing application domains and their integrations by AL-FUQAHA *et al.* [15]

Social networks have completely changed the relationships between individuals and communication in general. Industry is experiencing increased operating efficiency in a variety of applications, such as preventive machinery maintenance and active environment monitoring, and is often referred to as industry 4.0. As new application domains are explored in the context of IoT, numerous challenges are emerging and many of them are not yet fully addressed [15].

2.2 IoT System Architectures

IoT must be able to interconnect a large number of heterogeneous objects (in the order of billions or trillions) across the Internet, so there is a critical need for a flexible layered architecture [15]. Although the growing number of proposed architectures has not yet converged to a single reference model [16] it can be said that many published works considers at least three distinct physical tiers: things, fog / edge and cloud [15], [17].

tierThe first (lower) tier is called *things* tier, which aims to connect the physical and digital worlds. This tier encompasses multiple types of physical entities (objects) such as buildings, electrical appliances, clothes, animals or vehicles, equipped with sensing or actuating components that are responsible for collecting and sending data

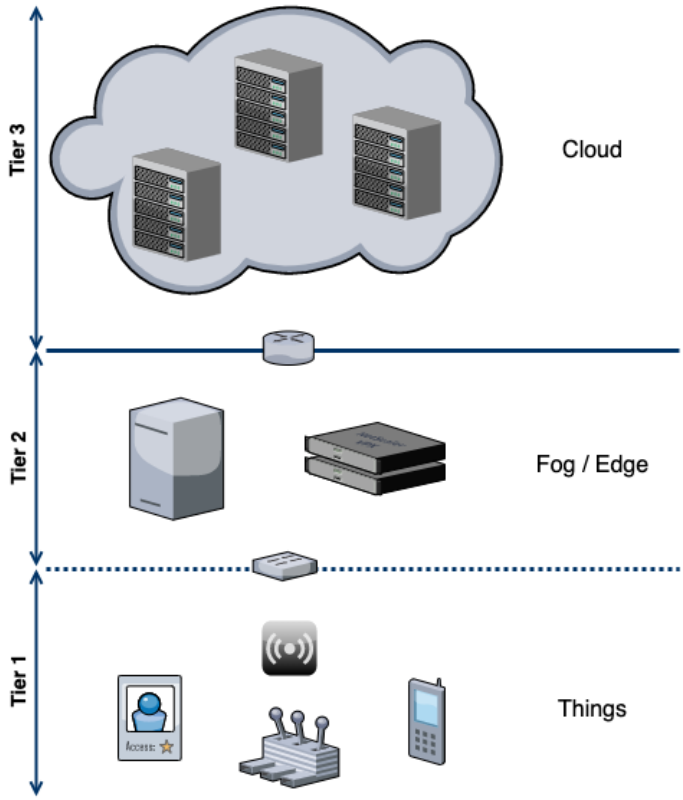

Figure 2.2: A three tiered architecture for an IoT system, adapted from LI *et al.* [17]

to the upper tiers for further processing or acting over the physical entities to change their state [17]. An important aspect in IoT systems is the presence of a WSN in this tier [1]. Commonly a WSN is comprised of resource-limited, often battery-powered, low-computing devices (memory, CPU, disk, etc.) which poses inherent challenges to IoT as already mentioned in Chapter 1.

The middle tier is the *fog* or *edge* tier depending on the terminology adopted. Fog / edge nodes are devices which can provide processing and storage resources for running services at the edge of the network [17]. This computing paradigm is a key aspect of the framework proposed in this work and it is further discussed in Chapter 2.6.

The third and upper tier is the *cloud* tier, comprised by the data centers that can provide data-intensive computation and permanent storage of huge, valuable data chunks for users and applications on a pay-as-you-go or utility-like pricing model. The presence of the intermediate fog tier with its edge nodes to handle some processing tasks avoids offloading massive raw data through the network and enables a more efficient usage of cloud resources [17] in addition to a more moderate consumption of network bandwidth [18].

2.3 IoT Application Protocols

Several IoT standards have been proposed to facilitate and simplify the development of IoT systems. These standards range from technologies and protocols to infrastructure, service discovery and application, among others [15]. In the group of protocols for infrastructure are, for example, Z-Wave [19, 20], Bluetooth Low Energy (BLE) [21, 22] and IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) [23]. In the group of service discovery protocols we can highlight Multicast DNS (mDNS) [24] and DNS Service Discovery (DNS-SD) [25]. However, the most relevant group of protocols for the context of this work is of applications, where it stands out Constrained Application Protocol (CoAP) [26], Message Queue Telemetry Transport (MQTT) [27] and Advanced Message Queuing Protocol (AMQP) [28].

Created by The Internet Engineering Task Force (IETF) Constrained RESTful Environments (CoRE) working group, CoAP defines a web transfer protocol based on Representational State Transfer (REST) [29, 30] on top of HTTP functionalities. It uses Uniform Resource Identifiers (URIs) as nouns and HTTP get, post, put, and delete methods as verbs. Unlike REST, CoAP is bound to UDP (not TCP) which makes it more suitable for IoT applications. Furthermore, CoAP modifies some HTTP functionalities to meet the IoT requirements such as low power consumption and operation in the presence of lossy and noisy links. CoAP aims to enable tiny devices with low power, computation and communication capabilities to utilize RESTful interactions [15].

MQTT is a messaging protocol introduced by IBM in 1999 and standardized by OASIS in 2013 [27]. It aims at connecting embedded devices and networks with applications and middleware. MQTT utilizes the publish/subscribe pattern to provide transition flexibility and simplicity of implementation and it is suitable for resource constrained devices that use unreliable or low bandwidth links. It is originally built on top of TCP protocol but one of its specifications, MQTT-SN [27], defines a UDP mapping of MQTT and adds broker support for indexing topic names. MQTT consists of three components: subscriber, publisher and broker. A device subscribes for specific topics in order to be informed by the broker when a publisher posts on topics of interest. In addition, the brokers achieve security by checking authorization of publishers and subscribers [15].

AMQP is an open standard focused on message-oriented environments. It supports reliable communication via message delivery guarantee primitives, including at-most-once, at-least-once and exactly once delivery. AMQP requires a reliable transport protocol such as TCP for message exchanging. Two main components handle communications: exchanges and message queues. Exchanges route the messages to appropriate queues. Routing between exchanges and message queues is

based on predefined rules and conditions. Messages are stored in queues and sent to receivers. Besides this model, AMQP also supports the publish/subscribe pattern.

2.4 Wireless Sensor Networks (WSN)

Advances in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics have enabled the development of low-cost, low-power, multifunctional sensor devices that are small in size and communicate untethered in short distances [2]. A wireless sensor network can be defined as a network of these tiny sensor devices, which are spatially distributed and work cooperatively to communicate information gathered from a monitored entity through wireless links. The data gathered by different nodes is sent to a sink which either uses the data locally or is connected to other networks or the internet through a gateway [31]. Figure 2.3 depicts a typical representation of a WSN.

WSNs are closely related with and are also considered an enabler of IoT [15]. It often comprises the things tier of our three-tiered reference model [17]. Not surprisingly, many of the problems and challenges faced in IoT are also encountered in WSNs. The processing, storage and energy resources available in the sensors are quite limited. In general, sensors are powered by non-rechargeable batteries. Due to their enormous quantity and their installation in hard to reach places, these sensors must operate without human assistance for long periods of time. The constant replacement of sensors that have exhausted their batteries is undesirable and diminishes part of the benefits of using WSNs. Therefore, the operational life span of WSNs is severely limited by the battery capacity of its nodes. Energy saving then becomes a crucial issue in these networks. Therefore, all stages of design and operation of WSNs must take energy consumption into account and seek to optimize it [14].

Nowadays, Internet and its traditional communication protocols such as Hypertext Transfer Protocol (HTTP) [32] are mainly used to transmit information among

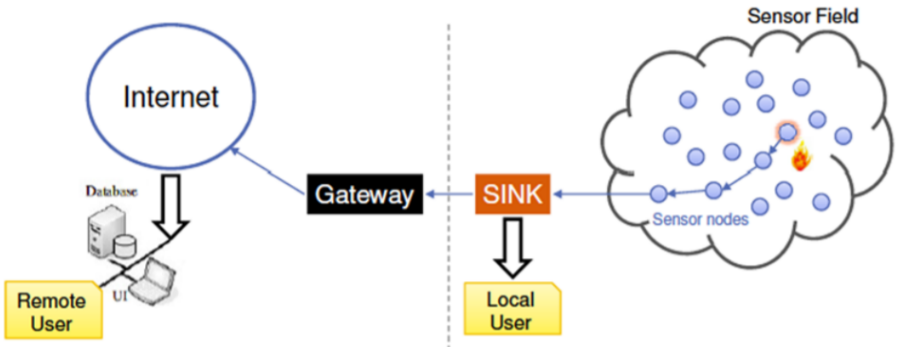

Figure 2.3: A wireless sensor network (WSN) by RAWAT *et al.* [31]

people, while the WSN can realize the short distance communication among the objects by constructing wireless networks in ad-hoc manners. Nevertheless, it is difficult to connect Internet and WSNs due to the lack of uniform standardization in communication protocols and sensing technologies. Also, data from WSNs cannot be transmitted in long distance with the limitation of WSN's transmission protocols. Therefore, with the development of IoT, new components called Gateways have been developed and/or adopted to act as bridges to settle with the heterogeneity between WSNs and the Internet [33].

2.5 Energy Preservation in WSN

Any sensor network must have a lifetime long enough to fulfill the application requirements, which can be of the order of several months, or even years. It is worth reinforcing that the power source for devices on a WSN often consists of a battery. Batteries have a limited energy budget and recharging could be impossible or inconvenient since the nodes may be deployed in a hostile or unpractical environment [9]. It is possible to scavenge energy from the external environment as of using solar cells, for example. However, such external power supply sources often exhibit a non-continuous behavior so that a battery is needed as an energy buffer as well [34]. It leads to a conclusion that energy is a critical resource which must be used very sparingly. Therefore, energy conservation must be seriously taken into consideration when designing systems based on WSN [9].

Experimental measurements have shown that usually data transmission is significantly more expensive in terms of energy consumption than data processing [35]. The energy cost of transmitting a single bit of information is approximately the same as the cost of processing a thousand operations in a typical sensor node [36]. Therefore, exchanging communication operations for computing operations is an effective strategy to reduce energy consumption in a WSN. Moreover, a large amount of energy is consumed by node components (device itself - CPU, radio, etc.) even if they are idle. Power management schemes are used for switching off device components that are not temporarily needed.

Several techniques have been developed to reduce energy consumption on WSNs. According to ANASTASI *et al.* [9], those techniques can be classified, at a very general level, into three main groups: *duty cycling*, *data-driven*, and *mobility*.

Duty cycling techniques focus on two different and complementary approaches:Topology control and power management. The general idea consists of exploiting node redundancy, which is typical in sensor networks, by selecting only a minimum subset of nodes to remain active for perform its tasks. Nodes that are not selected can go to sleep and save energy [9]. There are numerous topology control

techniques available [37, 38]. LE-Stream proposes in Chapter 4.3 a novel approach which can be classified according to RAHMAN *et al.* [13] as an Active Node Selection technique. Sleep/wakeup protocols [39] and MAC protocols with low duty cycle [40, 41] are prominent power management techniques found for WSN.

Data-driven approaches can be divided, according to the problem they address, in data acquisition and data reduction techniques. Data acquisition is mainly focused at reducing the energy spent by the sensing subsystem while data reduction aims at unneeded samples.

Energy efficient data acquisition techniques emerged from the need to tackle the issue of excessive energy consumption in the sensing activity itself. In such cases reducing communications may be not enough. Thus, reducing the number of acquisitions (data samples) is necessary [9]. Adaptive sampling, Hierarchical sampling and Model-based active sampling stand out among the known techniques for Energy efficient data acquisition in WSN literature [42]. Adaptive sampling strategies are very effective on reducing the number of samples by exploiting spatio-temporal correlations between data. However, they are usually not suitable to applications that required sampled data to be provided continuously [42].

Data reduction techniques can be further subdivided into three groups: data compression [43], in-network processing [44] and data prediction. Data prediction strategies are widely discussed by LE BORGNE *et al.* [10] and RISTESKA STOJKOSKA e MAHOSKI [45] with a special attention to time series forecasting techniques. DPS, briefly described in Chapter 1, is the most popular paradigm for time series forecasting in WSN, but it is not very accurate when predicting physical phenomena subject to sudden variations [45]. Data prediction strategies also encompass stochastic and algorithmic approaches [9].

For scenarios where not all nodes of the network are static [46], mobility can also be used as a data-driven approach for reducing energy consumption. In a static sensor network packets follow a multi-hop path towards the sink. Thus, a few paths can be more loaded than others, leading to a situation where nodes closer to the sink relay more packets, being more prone to premature energy depletion. If some nodes are mobile devices, the traffic flow can be altered when those devices are responsible for data collection directly from static nodes. Static nodes wait for the passage of the mobile device and route messages towards it, creating proximity based communication [9]. However, it is important to mention that LE-Stream does not consider the mobility factor on its energy preservation strategy.

To be able to provide data to consumers continuously and achieve accurate data prediction while monitoring a phenomena subject to sudden variations, LE-Stream introduces a data-driven energy conservation scheme combining a novel collaborative adaptive sampling strategy with a two-step data prediction model in Chapter 4.

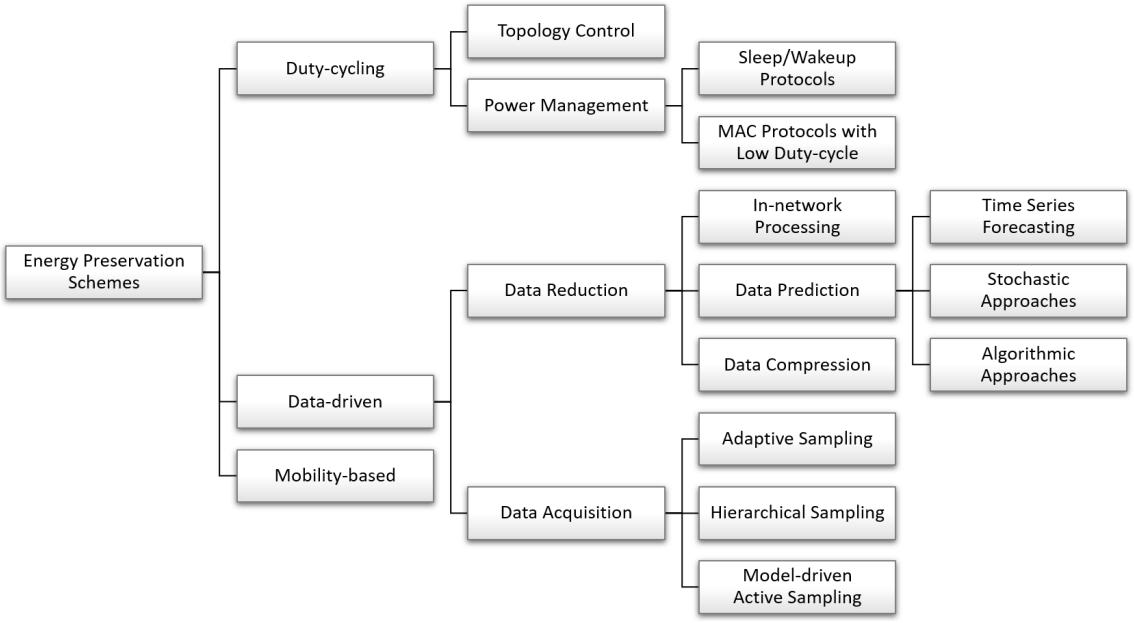

Figure 2.4: Energy preservation schemes for WSN

2.6 Cloud, Fog and Edge Computing

A breakthrough has been made in the area of IoT with the integration with cloud computing, so called Cloud of Sensors [17]. The virtually infinite capacity of computing resources offered on a pay-as-you-go basis by service providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform extends to micro enterprises and self-employed developers all the computing power which was previously only accessible by large companies due to the high investment required [15].

There are several published works where the solution relies on a complete data offloading to the cloud, such as TAHERKORDI *et al.* [18]. However, while efficient for many scenarios, it cannot be said that this is the standard infrastructure solution to all problems (*one size does not fit all*). There are extremely latency sensitive IoT applications where real time response is a strong requirement. For example, in a surveillance system that acts immediately upon image recognition on a TV circuit, a response time in the order of minutes can lead to delayed actions and compromise the overall solution. This need is also very present in fire-fighting applications where the environment is actively monitored and millisecond processing is critical for triggering safety mechanisms to be able to extinguish fire or prevent it from spreading.

The challenge of minimizing network latency and increasing processing speed in the context of IoT has been recognized as major research challenges in the area [8]. As a consequence, the concept of fog computing has been suggested and it is being developed by industry experts and the academic community. Conceptually, fog brings "the cloud" closer to data sources (things), such as sensors and actuators,

with data processing tasks running on a local area network infrastructure [7].

With similar purposes, and in some cases treated only as a matter of terminology, the concept of edge computing also proposes to bring data processing closer to data sources. The soft difference, also highlighted by DAUTOV *et al.* [7] lies in the fact that edge understands the processing of data directly on things (sensors, smart devices, etc.) or even on network edge devices such as routers or switches while fog computing concept can be less restrictive in terms of devices, extending to embedded servers or even small local data centers.

Architectures that comprise fog and / or edge computing layers for data stream processing are becoming frequent and emerging as an interesting research path. For example, MORABITO *et al.* [47] study and compare lightweight virtualization techniques applicable at the edge layer in different IoT application scenarios. FEI *et al.* [48] discuss Machine Learning (ML) techniques as promising to achieve time and mission criticality and also provides guidance on how to deploy ML on cloud and fog scenarios.

2.7 Data Stream Processing

Technological advances have significantly enhanced the data collection capabilities of embedded sensor devices, resulting in more data generation and real-world streaming data. [4]. Data stream processing is a wide area of study that has a strong relationship with the concepts of big data and IoT.

Many definitions for Data Stream can be found in the literature, but two specific ones are worth mentioning: while BABCOCK *et al.* [49] define such concept as *"Multiple, continuous, fast and time-varying streams of data"*, DIAS DE ASSUNÇÃO *et al.* [3] state that data streams are *"Incoming data that reaches a high rate, often being considered big data, thus stressing the communication and computing infrastructure"*. It is noteworthy that speed and variability aspects stand out in the definitions and may drive the architectures and processing solutions in this field.

Data streams have the following characteristics: (i) the fact that they are transient, i.e. not persisted; (ii) there is no control over the order or frequency of the data; (iii) they are potentially unbounded in size and (iv) they often have high volume and (near) real-time processing requirements [49]. These characteristics differentiate this data type significantly from traditional (persistent) data models and require specific processing / management techniques and tools.

Frameworks can be considered as pillars for building data stream processing systems (DSPS) [50]. They usually involve multilayered architectures with loosely coupled components to facilitate maintenance and provide scalability and availability

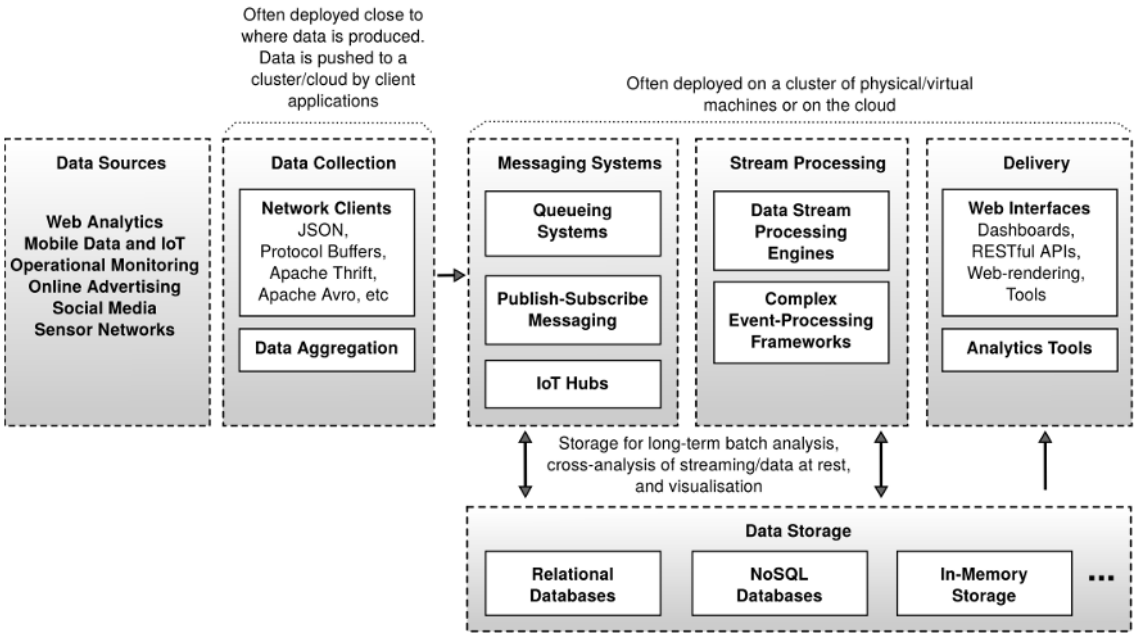

Figure 2.5: Overview of an online data processing workflow by DIAS DE ASSUNÇÃO *et al.* [3]

[3]. In this work we consider the general model proposed by [50]. It includes (i) a data stream ingestion layer, responsible for accepting data into the DSPS; (ii) a data stream processing layer, which preprocesses and analyses data in one or more steps; (iii) a persistence layer that stores, indexes and manages the data and its generated knowledge; (iv) a resource management layer, which coordinates the functions of distributed compute and storage resources; and (v) an output layer that directs the output data to services and applications.

2.8 Data Outliers and Outlier Detection

Another relevant feature present in sensor-generated data, specially in time series, is the presence of *outliers*, which are anomalies belonging to the class of unreliable readings. These are readings that are outside of what is considered a "normal state" of the data being collected. It can be represented by a model, for example. These "abnormal" readings are considered *"events with extremely low probability of occurrence"*. Outliers are elements that differ significantly from others in a dataset, which does not necessarily mean that they represent errors. In fact, outliers can represent important information for the application [4]. Also, in dynamically changing environments, the data distribution can change over time. This is a phenomenon known as concept drift [51]. An example of concept drift is a change in users interests when following an online news stream. While the distribution of the incoming news documents often remains the same, the distribution of the interesting and not interesting

news documents for that user changes.

Outlier detection can play a key role on an IoT data stream processing workflow [48], either to identify and eliminate inconsistent values that compromise data accuracy transmitted to the next task (data cleaning) or to identify events that are important to the use case, such as a peak of hypertension on an IoT-assisted patient or fire on a monitored environment [52], or still identify concept drifts.

Different outlier detection techniques are applied in the context of IoT [53]. Among the applied techniques, we highlight those based on statistical models and, more recently, on data mining techniques mainly based on Artificial Intelligence (AI), more specifically machine learning and deep learning. Some of these techniques are compiled and briefly described by KARKOUCH *et al.* [4], such as anomaly detection with Hierarchical Time Memory (HTM); an estimation-based approach to finding reliable sensors in environmental sensing and an outlier detection algorithm using big data processing and IoT architecture using K-Means clustering.

From an infrastructure perspective, cloud computing and distributed technologies, such as the use of computer clusters or grids, are possible solutions to solve big data problems in general. On the other hand, from a software perspective, most techniques and / or algorithms were not designed under paradigms for parallel and / or distributed computing (as MapReduce, for example). In this context, it can be said that many of the more traditional solutions cannot be directly applied to process the huge volume of IoT data without being redesigned [53].

Clustering is a widely studied problem in the data mining and AI literature [54]. However, it is more difficult to adapt arbitrary clustering algorithms to the context of DSP [52]. Its potentially unbounded in size feature makes this adaptation especially complex. The clustering problem is generally understood as: *having as input a set of unlabeled patterns, the output of optimal clustering is a partitioning of this set into a given number of clusters, based on a predefined similarity metric* [53].

K-means is certainly one of the best-known clustering and also the starting point for a number of variations tailored for stream processing [53]. However, the number of clusters is an input parameter, making such an algorithm unsuitable for some situations. AGGARWAL [52] highlights the micro-clustering technique as more effective and versatile than K-means for the context of data streams, and cites density-based techniques such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) as promissing solutions to the problem of clustering in an IoT scenario. Techniques like this, in turn, are able to determine the number of clusters as an output.

The classification problem is perhaps one of the most widely studied in the context of data mining on streams (stream mining) [54], and it is hampered by the

variability / evolutionary characteristic of data streams, sometimes referred to in the literature as concept drifts [55]. Therefore, effective algorithms need to be designed to take into account the principle of temporary locality, i.e. access to the same resource two or more times over a short period of time [52]. Unlike clustering, which does not presuppose prior knowledge to guide the partitioning process, classification presupposes some prior knowledge of the monitored problem or phenomenon to guide the partitioning process and build a set of classifiers to represent the possible pattern distribution. Formally, a classification algorithm can be defined as follows: Given a labeled dataset and an unlabeled dataset, the labeled dataset is used to train the classifier (or prediction function) which categorizes (or classifies) the unlabeled one.

Artificial neural networks (ANN), Decision Trees, K-nearest Neighbor, naive Bayesian classification, Adaboost and Support Vector Machines (SVM) are some of the widely adopted classification algorithms [53]. Very Fast Decision Trees (VFDT), On Demand classification, Ensemble-Based classification, and Compression-Based methods are some variants and adaptations studied for the context of data streams [52].

In the field of computational intelligence, classification is an example of a supervised learning algorithm, while clustering is an unsupervised learning one. The combination of different approaches, designing hybrid models, is also a practice that can lead to applicable and possibly promising solutions [53].

Chapter 3

Related Work

Assuring low response times is an important requirement in IoT and DSP applications [56]. A recent survey on solutions for latency-aware processing in data streams [57] identifies in-memory computing, support to non-structured or semi-structured data, low latency, and the usage of machine learning algorithms as key challenges in this field. The authors conclude that there is a lack of flexibility in the available solutions since they are too specific for the use cases they were designed to tackle. In addition, the survey states that cloud-based approaches still represent the majority of current solutions analyzed. The accuracy of the output data is always a major concern while the energy consumption does not appear among the challenges or objectives of the evaluated solutions for latency-aware DSP.

Another recent survey on IoT architecture challenges [58] highlights energy efficiency and latency-awareness (real-time or near real-time responses) as one of the major concerns when designing IoT systems. Quality awareness can also be inferred as a critical aspect for IoT architectures since concerns on data integrity and accuracy are present in most described approaches. The study, in turn, does not explicitly correlate energy and time efficiency on the analyzed solutions. In order to find this correlation, we cross checked the publications addressing energy and time efficiency. From the 29 works listed on [58], only three proposals aim to tackle both energy and time efficiency simultaneously, but they are not suitable for edge based processing running on constrained devices. The solutions proposed in [59] and [60] are cloud-based DSP solutions that rely on fully offloading the collected data to the cloud. This kind of approach leads to a communication bottleneck, making it hard or even impossible to meet strict low latency requirements. The architecture described in [61] is an effective latency-aware stream processing solution but it relies on a robust infrastructure of servers at the edge. It is not designed for constrained gateway devices, and it does not address the energy efficiency of gateways or sensor nodes. OSCAR [62] is not related to data stream processing. It is an energy-efficient architecture for real-time communication (machine-to-machine and

multicast) focusing on security.

Despite relevant proposals addressing separately the issues of low latency, data accuracy or power consumption can be found in the field of data stream processing for IoT, to the best of our knowledge, no solution tackling these three concerns simultaneous has been found so far. This makes it difficult to deploy solutions that can efficiently respond to real-time events in power-constrained environments, such as a forest fire suppression system, a malfunction detection system on small ships, etc.

IRESE [8] is an outlier (so called “rare-event") detection system that applies machine learning directly on gateways to identify events on audio data streams. The authors claimed contributions are (i) a sharp technological focus on edge computing, (ii) to strictly consider limitations of data stream analytics, and (iii) detecting rare-events without prior knowledge. Although processing efficiency is proved by using devices constrained on computing power (CPU, memory), no concern regarding energy consumption is mentioned. Therefore, this solution might not be feasible in environments with limited power sources, which is a major concern in our work. In addition, IRESE was designed to handle a particular type of data: audio streams. Our proposal is more agnostic and not restricted to a single data type or use case.

In [63], the authors introduce a high-dimensional data cleaning method for mobile edge nodes in the context of WSN. It is an adaptive mechanism that combines machine learning techniques with the edge computing paradigm to optimize the cleaning model in near real-time. The authors highlight the importance of the data cleaning task on the Industrial IoT scenario and how the edge computing approach can significantly reduce the energy consumption of sensor nodes and accelerate the cleaning speed of abnormal data. The authors propose a new algorithm to clean data using the mobile edge nodes called Angle-based Outlier Detection (ABOD). It is important to underline that the authors in [63] introduce the edge computing paradigm in their solution with an energy consumption concern, unlike what happens in [8] where the authors concern latency-aware processing. This aspect reinforces how representative edge computing can be on the DSP for the IoT field.

Dual prediction techniques are presented in [64], [65] and [66]. The proposals in those papers are combinations of adaptive sampling with data prediction models based on exponential time series. The core idea builds on a set of lightweight calculations performed at the WSN nodes. These computations allow the sensor nodes to deliver a function that allows predicting sensor readings in the time interval between the current and the next reading. Data prediction models are inserted in this context to avoid jeopardizing the accuracy of the data generated by the decrease in the sampling frequency. The main goal of such a combined approach is to find a good balance between energy consumption and the accuracy of the data produced by a

WSN. A common feature observed in works that present dual prediction schema is that their prediction models are applied individually by each sensor. The framework proposed in this paper differs from these works since it considers the readings of a group of sensors. It not only uses an aggregate function to predict data, but it also identifies and eliminates incorrect readings from its computations.

In [67] the authors introduce an efficient data collection framework for WSN. The framework reduces the activities of the nodes to decrease the overall energy consumption of the network nodes, but it keeps the desired redundancy for the sensing requirements. For example: if an asset needs to be monitored by 25 sensors each second to ensure measurement accuracy, their proposal guarantees the number of sensor readings required even by systematically disabling sensors over time. Their contributions are (i) reduce energy consumption by keeping as many nodes as possible on sleep mode and also avoiding data transmission for the active nodes whenever possible, (ii) provide a dynamic way to define the dependency correlation between nodes, and (iii) propose a heuristic algorithm for the selection of active nodes. DSP capability is not mentioned. Time intervals in the order of hours are assumed to "virtualize" the data using several statistical methods to perform computations. Bringing this approach "as is" to a DSP scenario would not be reasonable. With data coming at a high rate, these significant time intervals for computations would not identify sudden variations in the monitored phenomena. Our proposal similarly takes the idea of creating logical representations of physical sensors but relying on smaller adaptive time intervals and quick computations, more suited to use cases of DSP.

In [68], the authors proposed a four-layer infrastructure with distributed stream processing systems (DSPS) for IoT applications. Their evaluation results recommend using the Apache Nifi as the stream processing system in their infrastructure. Thus, according to the authors, the infrastructure performs with minimum latency and high throughput for IoT applications. For this, the authors evaluated the performance of well-known stream engines, namely Apache Nifi, Storm, Apex, Spark Streaming and Flink. They adopted five metrics in their evaluations: startup time, response time, throughput, jitter and scalability. Energy and data accuracy were not evaluated.

In work [69], a four-layer system architecture is proposed to process data in the edge. The layers are: (i) IoT data sensing, (ii) edge processing, (iii) data analysis and reasoning, and (iv) user application. The authors propose semantic data enrichment as a key enabler to further analysis and reasoning over IoT data streams. Thus, the edge layer has two hierarchical processing sub-layers: pre-processing and data enrichment. Data is pre-processed, and then the semantic enrichment is performed over the qualified selected data. Based on their results, the proposal is proved to be

efficient in latency and response time. Their evaluation results show the challenges of processing an IoT data stream in an edge computing layer related to efficiency, memory usage, and RTT (Round Trip Time).

In [70], the authors proposed an architecture conceptually based on latency-aware processing powered by machine learning and storm technology to process large amounts of data. Storm allows quickly processing a massive volume of data with low latency. The architecture consists of layers for data streams integrating, filtering, latency-aware processing, storage, and visualization with low latency. Ideally, their architecture must provide a limitless number of users to set up new features while handling traffic changes without interruption.

A distributed stream data processing architecture is proposed in [71] for edge environments. The architecture is based on Apache NiFi stream processing framework with support for clustering edge nodes in run-time. The devices are clustered and a shared pool of contributed resources is used to process computational tasks offloaded by peers. The architecture enables data-intensive applications to be deployed and performed at the edge with low latency.

Related work presented in this chapter propose effective solutions to address low latency, energy, or accuracy awareness, but none of them tackle all these three concerns simultaneously. Combining these three requirements in the same solution is complex because the approach used to solve one problem can negatively impact the solution of another problem. For example, statistical methods based on intensive computing can efficiently solve the lack of data accuracy, but they demand a high energy consumption from the devices. Our contribution consists of combining approaches such as those described, promoting the necessary adaptations to meet the three requirements jointly. Table 3.1 summarizes the papers and the respective requirements addressed by them, and also lists the proposals implemented in real devices or not.

The analysis of the related work presented so far aimed on proposals that address the requirements of low latency, energy efficiency and high data accuracy, focus of our paper. However, there are frameworks for analytics of sensing-based data aimed at other requirements. Among them, data security and privacy are relevant concerns, mainly in application domains such as healthcare. As the authors of the survey [72] point out, edge computing brings new security and privacy challenges when applied to data analytics. Besides inheriting some security issues from cloud computing, some distinctive features of edge computing, such as geographic distribution, heterogeneity and resource constrained devices, add further challenges. In this context, the authors in [73] proposed a smart security framework for VANETs equipped with edge computing nodes and 5G technology to enhance the capabilities of communication and computation in the modern smart city envi-

Table 3.1: Related Work.

Proposal	Energy	Latency	Data accuracy	Implemented in real devices
[59]	Yes	No	No	N/A
[60]	Yes	No	No	N/A
[61]	No	Yes	No	Yes
[62]	Yes	Yes	No	Yes
[8]	No	Yes	Yes	Yes
[63]	Yes	Yes	Yes	No
[64]	Yes	No	Yes	No
[65]	Yes	No	Yes	Yes
[66]	Yes	No	Yes	No
[67]	Yes	No	Yes	No
[68]	No	Yes	No	Yes
[69]	No	Yes	No	Yes
[70]	No	Yes	No	No
[71]	No	Yes	No	Yes
LE-Stream	Yes	Yes	Yes	Yes

ronment. The focus of their work was on communication issues, but they provided an energy-efficient secure system with minimum delay for processing data for Intelligent Transportation Systems. The authors in [74] claim that preserving the privacy of sensitive data for data aggregation applications in fog-based IoT systems is a major concern. To address this challenge, they propose APPA: a device-oriented Anonymous Privacy-Preserving scheme with Authentication for data aggregation applications in fog-enhanced IoT, which also supports multi-authority to manage smart devices and fog nodes locally. A comprehensive security analysis showed that the proposed scheme can achieve security and privacy-preservation properties in the fog-enhanced IoT systems, with low computational complexity and communication overhead. Finally, in a recent work [75], the authors tried to address the big data analytics approach while maintain privacy of healthcare databases for future knowledge discovery. Their proposal targets healthcare applications, a domain in which data privacy is of major relevance.

Chapter 4

LE-Stream: a Latency and Energy-Aware Framework for Data Stream Processing in IoT

The purpose of this work is to develop a time and energy-aware data stream processing framework for IoT. This chapter introduces the proposed framework, from a top level overview of its architecture to the detailed specification of its components.

4.1 Overview and Architecture

There are many tools and platforms for ingesting, processing, storing, and managing data streams, making it a difficult task for domain specialists to select the right combination to perform their analysis. The authors in [50] identified the main components of a modern data stream processing system (DSPS), which can be integrated into a framework. A data stream processing framework can be considered as a cornerstone for guiding the building of DSPS, addressing all the activities involved in the stream processing workflow [50]. This work is inspired on the general model for a DSPS framework proposed by [50] and described in Chapter 2.7. It also considers the three tiered IoT architecture proposed by [17] and presented in Chapter 2.2. We propose a DSP framework (depicted in Fig. 4.1) to be deployed at the things and edge tiers. Its goal is to provide strict response time and energy-aware data processing for IoT streamed data.

In the Fig. 4.1, at the things tier (bottom), each rectangle represents a physical entity (PE) which denotes a physical quantity to be monitored. Each circle represents a physical sensor (S). We assume that a group of sensors will always be present to sense the same PE and provide the necessary tolerance in case of failures. At the edge tier (middle), each ellipse represents a logical entity (LE), which con-

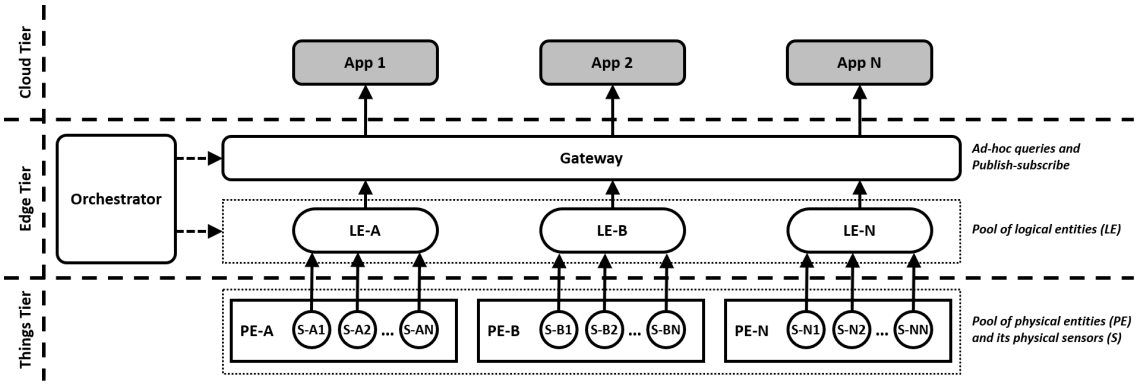

Figure 4.1: Schematic view of the proposed framework for strict response time and energy-aware data stream processing.

sists of a logical abstraction of a PE. The rounded-edged rectangle represents the Gateway which provides a Representational State Transfer Application Program Interface (REST API) [30] to allow communication between LEs and consumers. The Orchestrator is represented by a rounded-edged square and it consists of a software component which instantiates each LE and the Gateway. The sensor nodes at the things tier push readings to LEs via Constrained Application Protocol (CoAP) [26]. At the cloud tier (top), rounded-edged gray rectangles represent applications that consume data from LEs using the REST API provided by the Gateway. The proposed framework supports both synchronous queries and asynchronous patterns such as publish/subscribe [76].

Figure 4.2 represents the components of LE-Stream and their interactions through the operations provided by their respective interfaces. The adoption of decoupled components and the possibility of hosting them using lightweight virtualization techniques (such as containers) make the proposed framework potentially scalable in terms of physical entities, sensors and applications. Figure 4.3 illustrates

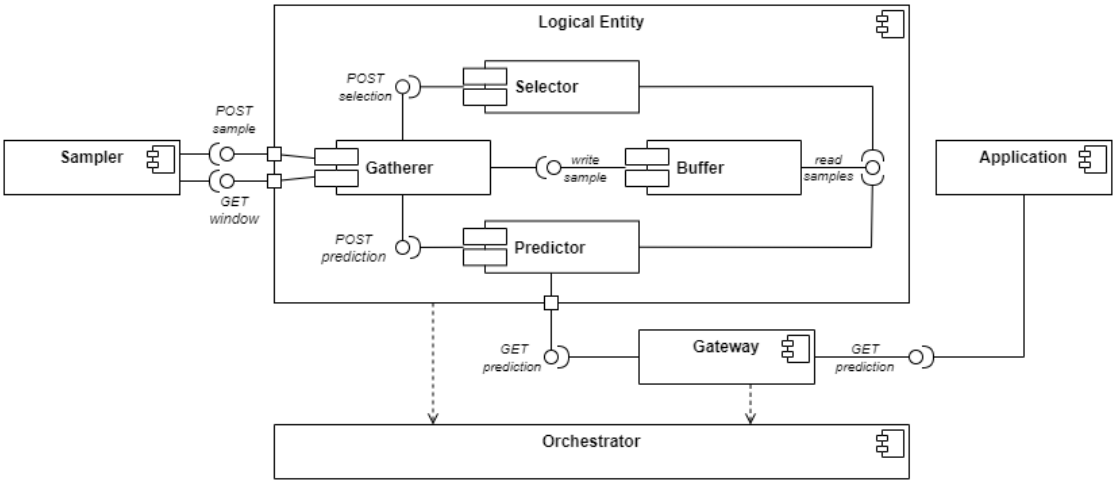

Figure 4.2: LE-Stream components, their interfaces and respective operations.

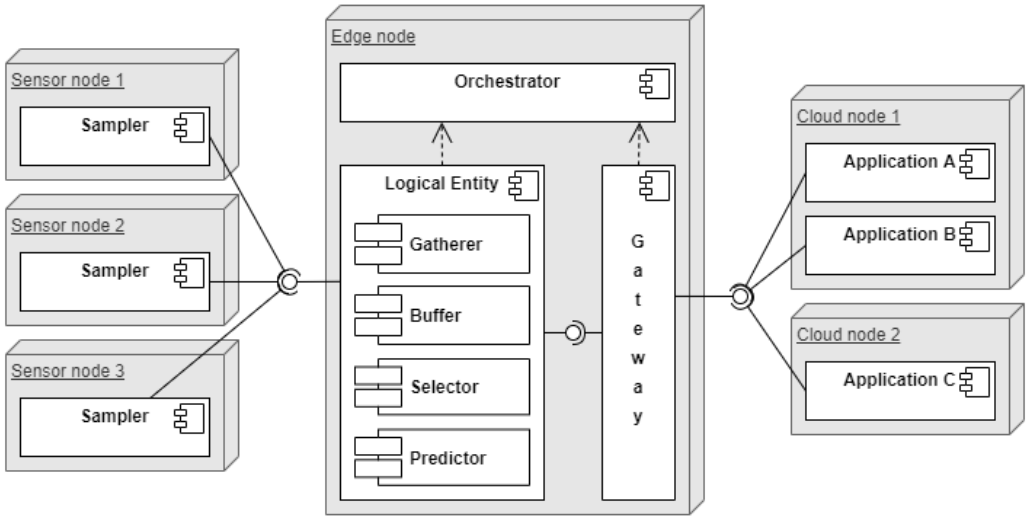

Figure 4.3: Nodes and components involved on a hypothetical scenario of a single entity monitored with three physical sensors and serving different applications.

a use case of a single entity monitored by three physical sensors and serving three different applications. We use a simplified scenario to allow the graphic representation of the components. It is important to mention that the same edge node can have several instances of LEs, each one associated with a set of physical sensors. The description of system operation will always consider the point of view of a single edge node with its associated physical nodes. In the current version of the proposal, collaboration between edge nodes is not considered, but this can be done in the future to obtain a broader view of the phenomena monitored in the environment.

The software components that encompass the proposed framework are described as follows, along with their correspondence to the layers for data stream processing systems proposed by [50]:

- *Sampler* - it is a CoAP client that runs on the sensor nodes and it represents the data ingestion layer. Its function is to sample the physical entity, send the data to the correspondent LE and receive a time interval as a response from the LE. This time interval is used to put the sensor node in a sleep state. When the sensor node goes back to the active state, this process is repeated.

- *Orchestrator* - it is a service that runs on the edge node to manage the lifecycle of resources by coordinating the provisioning of the LEs and the Gateway.

- *Logical Entity (LE)* - it consists of a set of decoupled modules (described below) that run on the edge node and work together to provide an abstract view of a monitored entity or phenomena:

- *Gatherer* - a CoAP server that listens to requests on a specific UDP port. Its function is to receive data sampled from sensor nodes and respond back with a value of time interval (elapsed time until the next data sampling). Section 4.2 describes how these time intervals are calculated.
- *Buffer* - an in-memory data store that keeps the data in the random access memory (RAM) is important [77] to contribute on comply with strict response time requirements for processing requests within milliseconds. This is used to persist sensor readings for a short predefined time.
- *Selector* - Since the time interval between samplings is variable, the workload distribution between the sensor nodes tends to be uneven. A local coordination mechanism is necessary to better distribute this workload [78]. This is a background service that keeps statistics about sensor nodes activity by reading the buffer on a predefined schedule. These statistics are used to identify overloaded sensor nodes which are pushed as a list to the Gatherer which will define longer time intervals for those sensor nodes identified as overloaded. Section 4.3 describes how these statistics are obtained and processed and how they contribute to improving the workload distribution between sensor nodes.
- *Predictor* - this is another background service responsible to periodically retrieve data from the buffer, identify and discharge incorrect readings and output a calculated value based on a predefined aggregation function. The latest prediction computed is pushed to the Gatherer component, to be used on the time interval calculations, and also served via CoAP requests to the Gateway component. This is the only framework component where the choice of algorithms is open to the developer according to the use case of his/her application as of described in Section 4.4.

- *Gateway* - it is an HTTP server [30] responsible for mediating the communication between LEs and applications. It consumes the predictions from the Predictor via CoAP and translates them to HTTP to serve web applications.

4.2 Adaptive Sampling Strategy

An adaptive sampling strategy consists on dynamically varying the time interval between samplings to somehow follow the variability of the sampled physical entity. Whenever there is little or no variation, the sample interval can be increased. Whenever there is a great variation, the sample interval must be reduced. The purpose of an adaptive sampling method is to reduce the number of samples per unit of time

to the maximum, aiming at reducing the energy consumption of the sensor node as much as possible. However, this strategy needs to be applied carefully so as not compromising the accuracy of the sample series [9].

When combined with a data prediction model, it is possible to calculate the difference between the sampled and predicted values. Thus, it is possible for a domain specialist to configure a tolerance threshold which shall be used to drive the adaptive sampling strategy. A so called greedy adaptive sampling strategy increases the sampling interval at each sampling until the computed prediction exceeds a predefined threshold, denoting a tolerance in terms of difference between the real and predicted values. Such tolerance can be an absolute value or a percentage corresponding to an acceptable margin of error between the predicted and sampled values for the physical quantity being sensed. When the threshold value is exceeded, the sampling interval is reduced so that the predicted and sampled values become closer again. Also, a minimum and a maximum threshold values can be defined to the value of the time interval itself. For example, it may be required that these time intervals are never less than 1s or greater than 30s [64].

The adaptive sampling strategy adopted on the proposed framework is similar to the one described in DPCAS [64], also proposed by our research group. It uses the concepts of the TCP congestion control algorithm to adjust the sampling interval. The strategy is based on the TCP CUBIC protocol [79], where the size of the windows vary according to a cubic function. The equations of the adaptive sampling method are as follows:

$$W_i = C(t - K)^3 + W_{max} \tag{4.1}$$

$$K = (\beta W_{max}/C)^{1/3}, 0 < \beta \leq 1 \tag{4.2}$$

Where W_i represents the sample interval calculated at the i-th sampling, which will be used as the sensor hibernation time until the next sampling, C is a scale factor known as a CUBIC parameter (typically 0.4), t is the elapsed time since the last reduction of the sample interval, W_{max} is the sample interval immediately before the last reduction of the sample interval and β a multiplicative reduction factor (typically 0.2). The factor K, described in Equation 4.2, is updated only when an event of reduction of the sample interval occurs. An event of reduction of the sampling interval occurs whenever the difference (δ) between the sampled value (Y_i) and its respective prediction (F_i) exceeds the tolerance limit of the application (ε), that is, whenever:

$$|Y_i - F_i| = \delta > \varepsilon \tag{4.3}$$

In addition, the application can also impose a minimum (S_{min}) and maximum (S_{max})

limit for sampling interval variation, that is:

$$S_{min} \leq W_i \leq S_{max} \tag{4.4}$$

In our proposal, we assume the presence of multiple sensors sampling the same physical entity, in a so called multi-sensed entity scenario. This is a strategy that aims to provide redundancy between the sensors involved in the sensing task so that a failure in the sampling of one sensor can be covered by the samplings of other sensors. In the adaptive sampling model proposed by DPCAS, each sensor node acts in a completely autonomous way. It samples the physical entity and calculates the time window until the next sampling based on its own samples. However, as the cubic function used to compute the time intervals is the same for all sensor nodes, the sensing rate is very close or the same for all nodes. Therefore, there is some synchronization between the sensor node activities since the time windows increase and decrease almost simultaneously. All the sensors would always be sampling at the very same time. Similarly, all the sensors would also be in a sleep state at the same time. This aspect creates a gap that we call a “blind window": if a sudden variation occurs when all sensors are sleeping, this variation will only be identified when the sensors wake up.

In such a multi-sensed entity scenario, all the sensors monitoring the same entity must not be in a sleep state simultaneously. This opens up an opportunity that we explore by proposing a collaborative strategy: to distribute different sampling intervals between multiple sensor nodes which are monitoring the same area or phenomenon. It aims at desynchronizing sensor nodes activities to reduce these blind windows and make the adaptive sampling model more responsive to sudden changes in the data stream.

To enable this collaborative and unsynchronized approach, the calculations of time intervals between samplings must be carried out at the edge node and not at the sensor nodes. The edge is the only node that communicates with all sensor nodes and it is also responsible for computing data predictions. Only the edge node has all the information required to calculate sampling intervals. In this way, the gatherer component is responsible for these computations by performing the steps represented on Figure 4.4.

To better understand how the proposed adaptive sampling works, let’s consider a basic use case where a group of five sensor nodes sense the temperature of a room. There is no tolerance limit for the temperature sampled (ε), which means that the reduction of the sample interval must occur every time the sampled temperature differs from the previous one. In addition, the minimum (S_{min}) and maximum (S_{max}) limits for the time interval between samples are 0s and 30s respectively. It

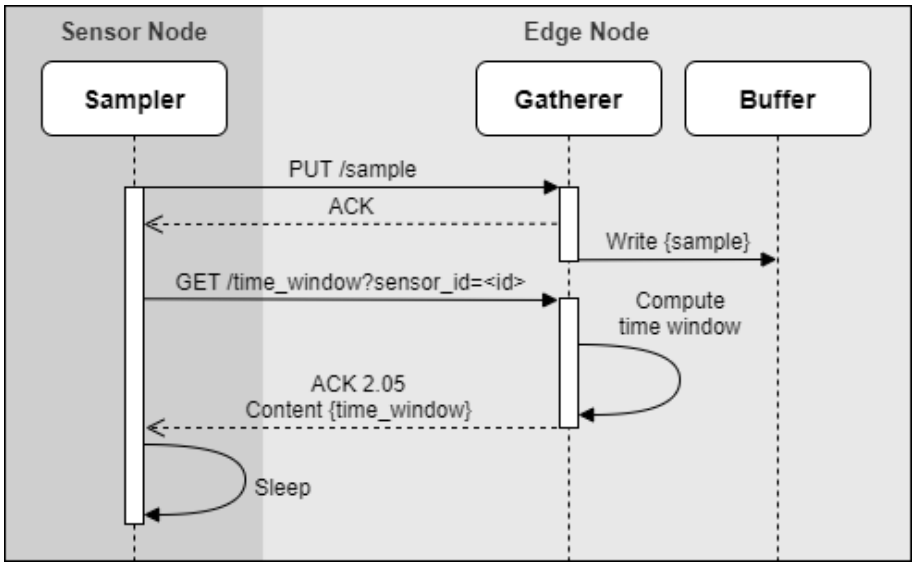

Figure 4.4: Sequence of events and interactions between software components in the sampling activity.

is also important to state how the real temperature behaves on this simulation. For sake of simplicity, let's consider the three stages described below:

1. Cool stage: during the first 30 seconds, the temperature stays steady in 25 °C.

2. Warming stage: from the second 31 to 90 it increases by 1 °C each 2 seconds.

3. Hot stage: from the second 31 on wards it stabilizes again in 55 °C.

Table 4.1 represents all the sampling events that occur on the first 2 minutes of the simulation in a chronological order. The first column indicates the time where the sampling happens since the beginning of the simulation. The second column identifies the sensor responsible for sampling the temperature which is registered on the third column. The fourth column indicates the calculated sampling interval (W_i) and the last column indicates the time when this sensor must take the next sample. The idea of how an adaptive sampling strategy works can be obtained in Figure 4.5. The orange line indicates how W_i varies according to the behaviour of the samples temperature (blue line). When the temperature is stable, W_i tends to increase until it reaches S_{max}. When the temperature changes, W_i decreases until the temperature gets stable again (or until it reaches S_{min}, which is not represented on this graph since it does not occur on the respective simulation). This dynamic behaviour of W_i can be understood as the general idea of an adaptive sampling strategy and it is applied to all the related works we found around this subject ([64], [65] and [66]). What makes our proposal different from the others can be understood by analyzing the values in the second column of Table 4.1. While the temperature is stable, the order of sensors sampling the temperature follows an expected sequence

Table 4.1: Simulation of the proposed adaptive sampling on a simple scenario.

Time (s)	Sensor	Temperature (°C)	W_i (s)	Next Sampling (s)
0,00	1	25	0,40	0,40
0,01	2	25	3,20	3,21
0,02	3	25	10,80	10,82
0,03	4	25	25,60	25,63
0,04	5	25	30,00	30,04
0,40	1	25	30,00	30,40
3,21	2	25	30,00	33,21
10,82	3	25	30,00	40,82
25,63	4	25	30,00	55,63
30,04	5	26	24,00	54,04
30,40	1	26	28,74	59,14
33,21	2	27	22,99	56,20
40,82	3	30	18,39	59,21
54,04	5	37	14,71	68,75
55,63	4	37	17,87	73,50
56,20	2	38	14,29	70,49
59,14	1	39	11,44	70,58
59,21	3	39	13,98	73,19
68,75	5	44	11,18	79,93
70,49	2	45	8,94	79,43
70,58	1	45	11,00	81,58
73,19	3	46	8,80	81,99
73,50	4	46	10,82	84,32
79,43	2	49	8,65	88,08
79,93	5	49	10,64	90,57
81,58	1	50	8,51	90,09
81,99	3	50	10,48	92,47
84,32	4	52	8,38	92,70
88,08	2	54	6,71	94,79
90,09	1	55	5,36	95,45
90,57	5	55	6,66	97,23
92,47	3	55	6,76	99,23
92,70	4	55	8,06	100,76
94,79	2	55	12,98	107,77
95,45	1	55	23,90	119,35
97,23	5	55	30,00	127,23
99,23	3	55	30,00	129,23
100,76	4	55	30,00	130,76
107,77	2	55	30,00	137,77
119,35	1	55	30,00	149,35

(check the first 10 and last 10 entries of the table). As the temperature, and W_i as a consequence, varies, the order of sensors taking samples becomes random. This random behaviour is intentional. The goal is to avoid making all sensors inactive at the same time. It aims at increasing the ability to identify sudden variations of the measured physical quantity (temperature in the given example) while keeping the reduced energy consumption of the sensor nodes.

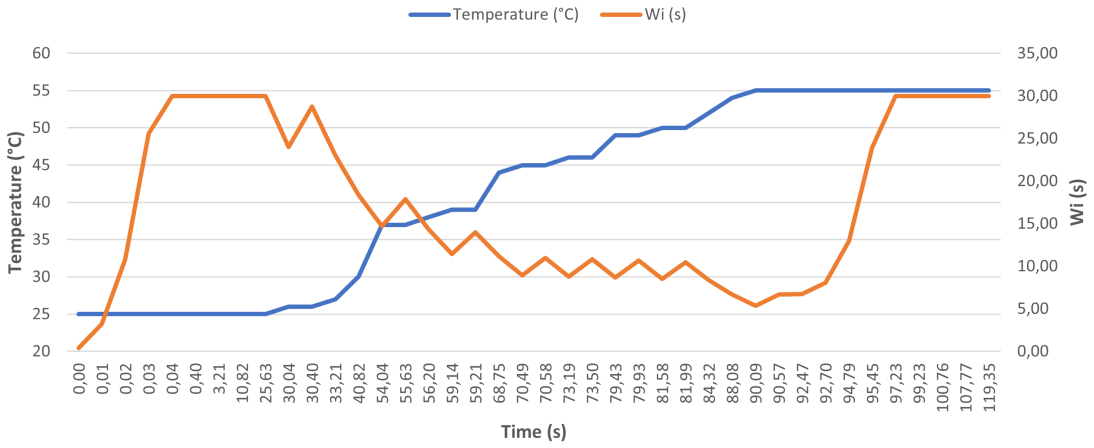

Figure 4.5: An example of an adaptive sampling strategy applied to a simple use case.

4.3 Active Node Selection

Although the proposed collaborative and unsynchronized adaptive sampling approach has a good potential to solve the “blind window" problem described in the previous chapter, on the other hand, it brings a potential issue of producing a poor distribution of the workload among the sensor nodes. Let’s aggregate the data from Table 4.1 into Table 4.2 to check how many samples were taken and the average W_i per sensor node:

Table 4.2: Summary of samples taken and average W_i per sensor node.

Sensor	Samples Taken	Average W_i (s)
1	9	16.59
2	9	15.31
3	8	16.15
4	7	18.68
5	7	18.17

Even for a very simple and short simulation it is possible to see the difference on the workload distribution between the sensor nodes. It can be said that sensors

1 and 2 sampled 28.57% more (9 samples / 7 samples) and slept (Average W_i) from 8.67% (16.59 / 18.17) to 18.03% (15.31 / 18.68) less than sensors 4 and 5. This difference is even greater for cases involving more sensors and longer sensing periods as described in Chapter 5.

The poor load balancing between sensors in a network is a problem widely studied in the WSN field. In several deployments of such networks, especially in scenarios covering small to medium-sized geographical areas, there is only one sink node, responsible for concentrating the data from all sensors. Considering the use of multi-hop transmissions, the closer to the sink node, the greater the workload of the nodes, which, in addition to their own data, will be responsible for transmitting those of the other nodes in the network. This produces fast depletion of energy from these nodes near the sink. The consequence of poor balancing is the early exhaustion of some nodes, which can bring two main issues: (i) in case such nodes serve as traffic forwarders for others, the network may be partitioned, and (ii) part of the monitored region will lose sensing coverage, compromising the accuracy of the event detection process and the monitoring of the phenomenon of interest.

To alleviate this unbalanced workload distribution, the Selector component is a service running on an edge node which acts as a local workload coordinator. Its role is to measure the workload distribution among all sensor nodes involved in the sensing activity by checking the Buffer from time to time and inform the Gatherer component which nodes are the most overloaded. These steps are illustrated as a sequence diagram on Figure 4.6 and described in detail in the next subsections.

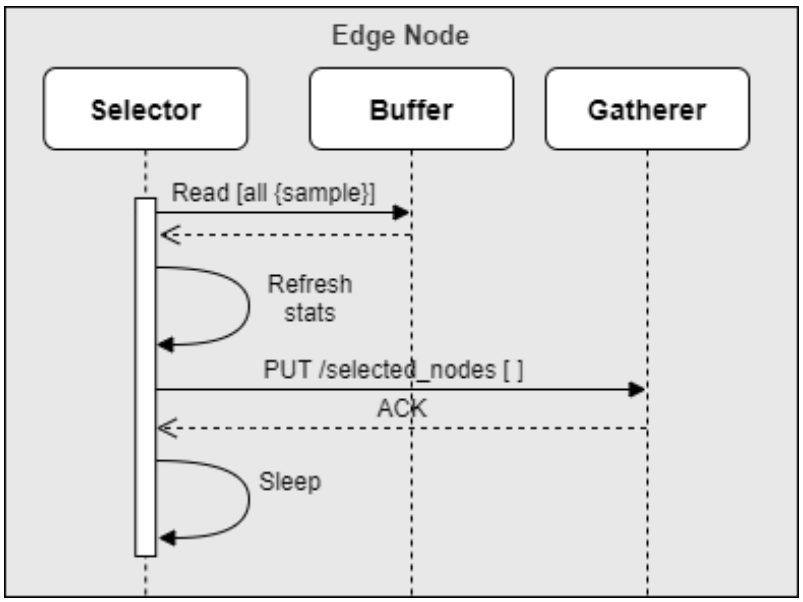

Figure 4.6: Sequence of events and interactions between software components in the active node selection activity.

4.3.1 Setting Workload Statistics

To be able to identify a situation of uneven workload distribution, the Selector component must calculate and store statistics (aggregations) on the number of samples taken per sensor node. The Selector component itself is capable of performing this calculation by reading the buffer periodically. It is essential that this consumption of the information in the buffer happens with a frequency lower than the lifetime (TTL) of the readings in the buffer. For instance, if each sample stays in the buffer for 45 seconds, it is recommended that the active nodes selector queries the buffer at least every 40 seconds to ensure that no samples will be lost. The Selector's **sleep time** is an adjustable parameter on LE-Stream. In the same way that there is a concern with the loss of samples, there must be a concern with not counting the same sample more than once. This problem can be solved by always keeping a copy of the keys present in the buffer on the previous reading. Thus, readings that have already been counted in the aggregations can be disregarded. Algorithm 1 and Figure 4.7 show how the workload statistics are maintained by checking the Buffer from time to time.

Algorithm 1: UpdateStatistics() Function

Result: Updates statistics (global) and returns a list of new keys read
Input: $buffer$: Data available at the Buffer
Input: $lastKeys$: list of last keys read
Output: $newKeys$: updated list of last keys read

```
/* Define scope and initialize variables                    */
global statistics[]
newKeys[] ← ∅
/* Scan Buffer data                                         */
foreach sample ∈ buffer do
    /* Mark key as read                                     */
    newKeys.append(sample.key)
    /* Increment sensorId count for new keys                */
    if sample.key ∉ lastKeys then
        statistics[sample.sensorId] ← statistics[sample.sensorId] + 1
    end
end
```

4.3.2 Evaluating Workload Skewness

The experiments carried out and described on Chapter 5 allowed stating that some level of unbalance will always exist. However, it can be large or small. The tolerable threshold on the workload difference between the sensor nodes may vary depending on the use case. From the aggregated data, so called statistics, the Selector com-

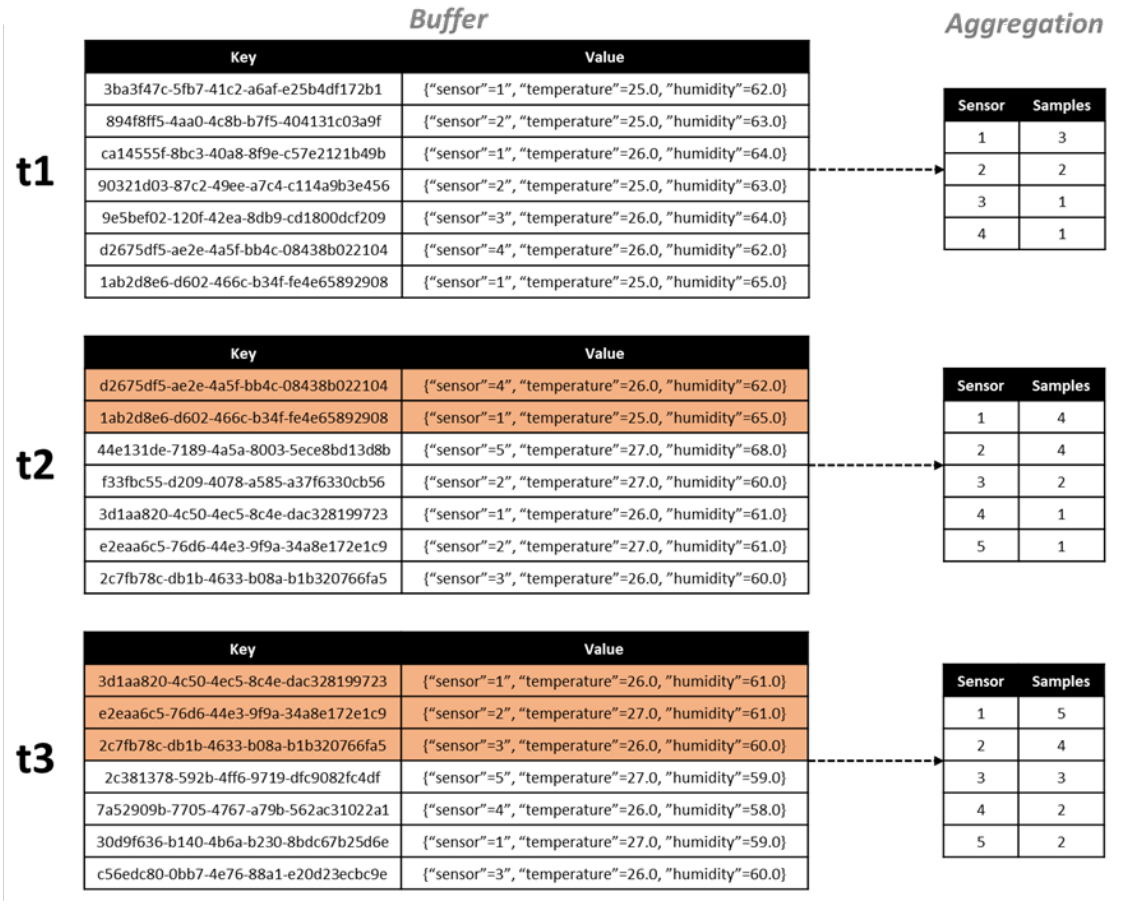

Figure 4.7: How the Sampler sets workload statistics / aggregations by reading the Buffer.

ponent is able to know how many samples were taken by each sensor node. Thus, it is possible to use the statistical metric of the standard deviation to evaluate the workload distribution between the sensor nodes (skewness) by applying the following equation:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{4.5}$$

Every time the statistics are updated, the standard deviation over the count of samples per sensor node is computed. The calculated standard deviation will always be a positive real value. If equals to zero, it means that the distribution is precisely balanced. The larger the calculated standard deviation value, the more unbalanced the distribution [80]. By computing the standard deviation the Selector can identify whether the distribution is skewed or not. To make this decision, a parameter of LE-Stream so called **skewness rate** must be set. This parameter allows the definition of the acceptable level of workload imbalance between sensor nodes. If set to 0 (zero), it means that if all sensor nodes do not have the same number of samples sent, the

distribution is considered skewed. It is important to emphasize that some imbalance always exists and, when small, it will not cause the premature death of overloaded sensor nodes. Assuming that in a normal distribution the standard deviation is equal to 1 [80], the default value for the skewness rate in LE-Stream is set to 1. Nevertheless it can be adjusted if an application requirement demands it. When the standard deviation is less than or equal to the skewness rate, the distribution is considered even. As a consequence, no sensor node is selected to be deactivated. When the standard deviation is greater than the skewness rate, the distribution is considered uneven and the Selector component proceeds with selecting sensor nodes to deactivate.

4.3.3 Selecting Sensor Nodes To Deactivate

Once the statistics are updated and the distribution is defined as uneven or skewed, it is time to select which sensor nodes must be deactivated. The first step of this task is simply sorting the statistics entries in descending order by the count of samples. Another important parameter of LE-Stream is the **sleep rate**, which stands for the size of the subset of sensor nodes to be selected for deactivation. For example, if the sleep rate is set to 0.3 in a group of 100 sensor nodes, it means that the top 30 sensor nodes in the ordered statistics list will be selected for deactivation. Algorithm 2 describes how the selection step works. Once the list of most overloaded sensor nodes is set, the Selector component sends it to the Gatherer component over a CoAP PUT request. Algorithm 3 summarizes the operation of the Selector component.

The Gatherer, in turn, replaces its local list of overloaded sensor nodes with this new list. This list will be checked before any W_i computations. Each sensor on this list will arbitrarily receive a predefined longer time window, which can be adjusted on LE-Stream according to the application requirements.

Algorithm 2: SelectNodes() Function

Input: *statistics*: Statistics list
Input: *sleepRate*: Sleep rate parameter
Output: *selectedNodes*: list of sensor nodes to be deactivated
`/* Sort statistics by value to a new list */`
$sortedStats[] \leftarrow$ `SortByValue`(*statistics*,DESC)
`/* Compute the size of the selection */`
$selectionSize \leftarrow$ `Round`(`Length`(*statistics*) $*$ *sleepRate*)
`/* Set a new list with the top overloaded sensor nodes */`
$selectedNodes[] \leftarrow \emptyset$
for $i = 0;\ i < selectionSize;\ i = i + 1$ **do**
  $selectedNodes$.append($sortedStats$[i].$sensorId$)
end

Algorithm 3: Selector component algorithm

Input: $buffer$: Data available at the Buffer
Input: $sleepTime$: Sleep time parameter
Input: $skewnessRate$: Skewness rate parameter
Input: $sleepRate$: Sleep rate parameter

```
/* Initialize variables                                          */
```
$statistics[] \leftarrow \emptyset$
$lastKeys[] \leftarrow \emptyset$
$selectedNodes \leftarrow \emptyset$
```
/* Runs continuously while buffer is not empty                   */
```
while $buffer \neq \emptyset$ **do**
```
    /* Refresh statistics                                        */
```
    UpdateStatistics ($Buffer$, $lastKeys$)
```
    /* Check skewness and update selectedNodes                   */
```
    if StandardDeviation $(statistics) \geq skewnessRate$ **then**
        $selectedNodes \leftarrow$ SelectNodes($statistics$, $sleepRate$)
    else
        $selectedNodes \leftarrow \emptyset$
    end
```
    /* Send selectedNodes to the Gatherer via CoAP PUT request   */
```
    PutToGatherer ($selectedNodes$)
end

4.4 Two-Step Data Prediction Model

A prediction model aims at computing future sensor readings. Simple Exponential Smoothing (SES) [79] and Double Exponential Smoothing (DES), also known as Holt Method [81, 82] are good examples of data prediction models. They are computationally economical and thus interesting choices for WSNs and IoT [64]. However, both were designed to predict readings of an individual sensor based on its own past readings.

As already mentioned in section 4.2, LE-Stream assumes a multi-sensed entity scenario. Thus, our data prediction model must consider readings from different sensors, physically closer to each other and under the control of the same edge node, to calculate its output value. IRESE [8] and VSF [67], presented in Chapter 3, also assume a similar multi-sensed scenario and inspired the approach we adopted in our proposed framework. From a high level perspective, our two-step data prediction model consists of (i) cleaning noisy / incorrect readings and (ii) computing an output value based on an aggregation function.

4.4.1 Identifying and Cleaning Incorrect Samples

Sensor generated data need to be accurate. Physical sensors fail. Incorrect readings need to be identified and discarded as best as possible to improve the quality of the information and decisions based on the acquired data. A common practice in sensed environments is to use a group of sensors to monitor the same entity, providing the necessary redundancy that makes it easier to identify incorrect or noisy readings [3]. Data streams are continuous flows of isolated data points. In an IoT use case with sensor generated data, these data points are represented by sensor readings. No data point can be considered an outlier on an individual basis analysis. Thus, defining a way to group and analyze these data points is a major concern when designing an outlier detection task. Buffering incoming data on a predefined length or time interval to create frames is a common approach to perform operations on data streams [53].

Clustering is a problem widely studied in the data mining and AI literature. However, it is more difficult to adapt arbitrary clustering algorithms to the context of data stream processing. Its potentially unbounded in size feature makes this adaptation especially complex [52]. K-means is certainly one of the best-known clustering and also the starting point for a number of variations tailored for stream processing [53]. However, since each outlier can represent a different cluster and the number of cluster is an expected input for K-means and its variations, such algorithms are not suitable for the purpose of outlier detection.

Density-based techniques are more effective and versatile than K-means for the purpose outlier detection data streams and IoT [52]. Such techniques, in turn, are able to determine the number of clusters as an output. Moreover, to ensure that outliers are not confused with concept drifts, LE-Stream uses short processing windows (on the order of seconds or, at most, one minute). In this way, whatever the algorithm adopted in the implementation of step 1 (clustering), the outlier detection will always be limited to the data points contained in this short window. If a longer window is defined, there is a risk of a concept drift to be confused with an outlier and data accuracy is compromised.

Therefore, even not being tied to a specific algorithm, LE-Stream requires that a density-based approach with short processing windows to be chosen. The Figure 4.8 provides a graphical representation which helps to understand why density-based clustering are a reasonable choice to tackle an outlier detection task. A pool of good candidate techniques is presented in [83]. For the sake of simplicity, we chose Density-based spatial clustering of applications with noise (DBSCAN) from scikit-learn [84] in our implementation of the predictor component on LE-Stream (Chapter 5). DBSCAN has a specific parameter where a radius (eps) is defined to be consid-

ered when evaluating the data set. Only points that are too far apart are considered outliers. Assuming the sensors redundancy and short processing windows, when there is a concept drift, most active sensors follow the variation, while outliers stand out. Algorithm 4 describes the original sequential DBSCAN algorithm. It assumes that any distance function can be used and also that RangeQuery() can be implemented using a database index for better performance or a slow linear scan [85].

Algorithm 4: Density-based Spatial Clustering of Applications With Noise

```
Input: db: Database
Input: eps: Radius
Input: minPts: Density threshold
Input: dist: Distance function
Output: db labeled with clusters or noisy points
cluster ← 0                                              // Cluster counter
foreach point ∈ db do
    if point.label ≠ null then continue
    neighbors ← RangeQuery(db, dist, point, eps)         // Find initial neighbors
    if minPts > |neighbors| then
        point.label ← −1                                 // Non-core points are noise
        continue
    end
    cluster ← cluster + 1                                // Next cluster label
    point.label ← cluster                                // Label initial point
    seedSet ← neighbors\{point}                          // Expand neighborhood
    foreach q ∈ seedSet do
        if q.label = −1 then q.label ← cluster
        if q.label ≠ null then continue
        neighbors ← RangeQuery(db, dist, q, eps)
        q.label ← cluster
        if minPts > |neighbors| then continue
        seedSet ← seedSet ∪ neighbors
    end
end
```

4.4.2 Aggregating Data

Choosing an aggregation function can become a complex task depending on the application's use case. Similar to the clustering algorithm for the outlier detection task, this is a feature where the proposed framework gives some autonomy to whom implement it. In a context of strict response time data stream processing for IoT, it can be assumed that the input data are readings sampled by sensors in a short time window. We must also take into account that these data points have already gone trough a data cleaning task and are trustworthy. Thus, central tendency statistical measures, such as mean and median, are reasonable choices to represent the value

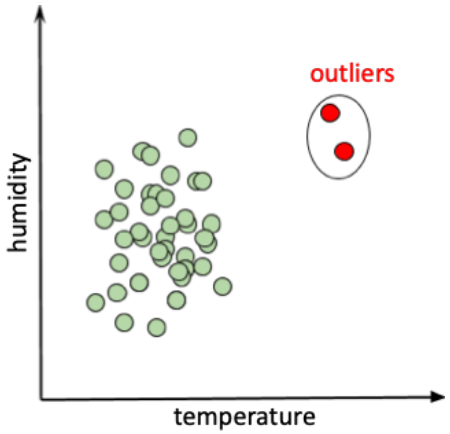

Figure 4.8: Outlier detection with density-based clustering

measured in that time interval. Also for the sake of simplicity, we chose to use the statistical metric of arithmetic mean in our implementation:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{4.6}$$

Regardless of the choices made when implementing the tasks of clustering (outlier removal) and aggregation, the Predictor component must interact with Buffer and Gatherer components as illustrated in Figure 4.9.

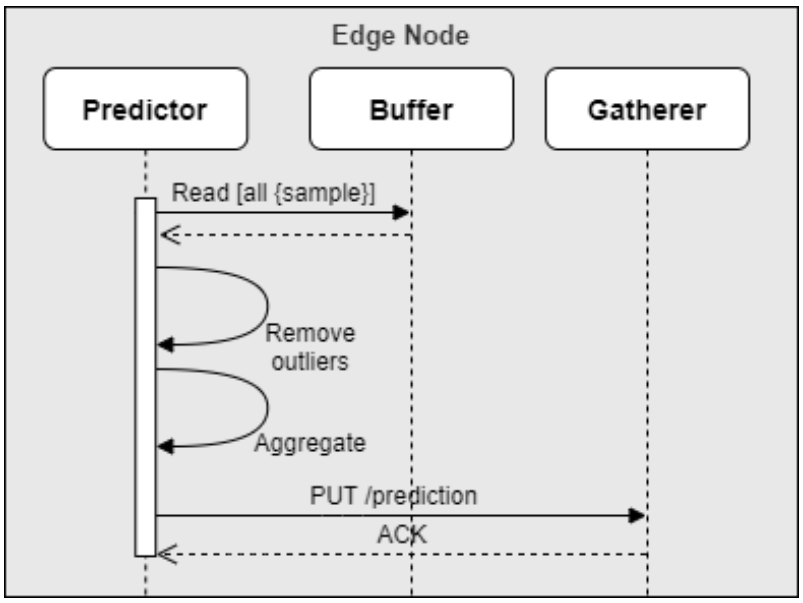

Figure 4.9: Sequence of events and interactions between software components in the data prediction activity.

Chapter 5

Experimental Evaluation

In this Chapter, we describe the experiments performed to evaluate the proposed framework in terms of (i) accuracy of the output data when reducing the number of sensor samples; (ii) how efficient is the proposed adaptive sampling strategy in terms of energy consumption; (iii) how effective is the active node selection feature on improving the workload distribution between sensor nodes and (iv) how fast is the proposed data prediction model to process buffered data (thus contributing to decrease the total response time for applications).

5.1 Use Case and Test Environment

A preliminary data set of real sensor readings was generated from six DHT11 sensors continuously collecting temperature and humidity data during a period of one hour. Then we used this data set to generate a synthetic data set of 480 sensors by adding to these measured quantities a random value ranging from -1 to 1. Finally the synthetic data set was used as an input to simulate two different but comparable scenarios:

1. **Fixed Sampling**: All the collected data is sent to the edge node according to its timestamp and used to predict the output. This scenario simulates a use case where all the sensors are active all the time.

2. **LE-Stream**: A number of independent processes runs an instance of the sampler component on a sensor node. Each process communicates with the edge node to send data when in the active mode, and sleeps according to the time windows received from the edge. Data points from the generated data set in which the timestamp corresponds to the sensor's sleep time are discarded.

All components were developed in Python programming language version 3.0. The CoAP features were implemented with CoAPthon [86]. Redis [87] is an in-memory data structure store which was used to implement the Buffer component.

The Gateway's REST API was implemented with Flask [88], which is a software framework designed to support the development of web applications including web APIs. In terms of infrastructure, we created an isolated virtual network on a public cloud environment to make it possible to simulate the scenarios on 5 different scales. Table 5.1 describes the number of sampler components and sensor nodes ingesting data into a single edge node. Also, the amount of computing resources (vCPUs and RAM) of the edge node for each round of simulation is indicated. Both the sensor nodes and the edge node are powered by 64-bit Arm-based processors, with 10 Gbps network bandwidth and Ubuntu linux 20.04 operating system.

Table 5.1

Round	Samplers	Sensor Nodes	Edge Node	
			vCPUs	RAM
1	30	1	1	4 GiB
2	60	2	2	8 GiB
3	120	4	4	16 GiB
4	240	8	8	32 GiB
5	480	16	16	64 GiB

5.2 Data Accuracy Analysis

The reduced number of samples sent to the edge node when using an adaptive sampling strategy implies that less data is available for the data prediction model. Thus, there is a concern that the data accuracy might be affected in LE-Stream. In fact, as it is shown in Fig. 5.1, while the fixed sampling mode keeps a high number of samples at the buffer all the time, in LE-Stream this number varies according to the entity being measured. When the temperature and humidity values are stable, the number of buffered samples is low. When the temperature and humidity values start to change, quickly the number of buffered samples increases. This behavior is also observed in all simulation rounds with different scales. As the number of sensors and thus the amount of data being ingested increase, the difference in the number of buffered samples over time is even greater. Fig. 5.1 also presents this difference between rounds of simulation 1 to 5.

To evaluate how this reduced amount of data available affects the data prediction output, we use Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) metrics [63], [89]. We assume the outputs from the executions using fixed sampling as the real / observed values to calculate the differences from the outputs obtained with adaptive sampling mode on each simulation round. Table 5.2 presents the calculated RMSE and MAE and data range per round of simulation for humidity and

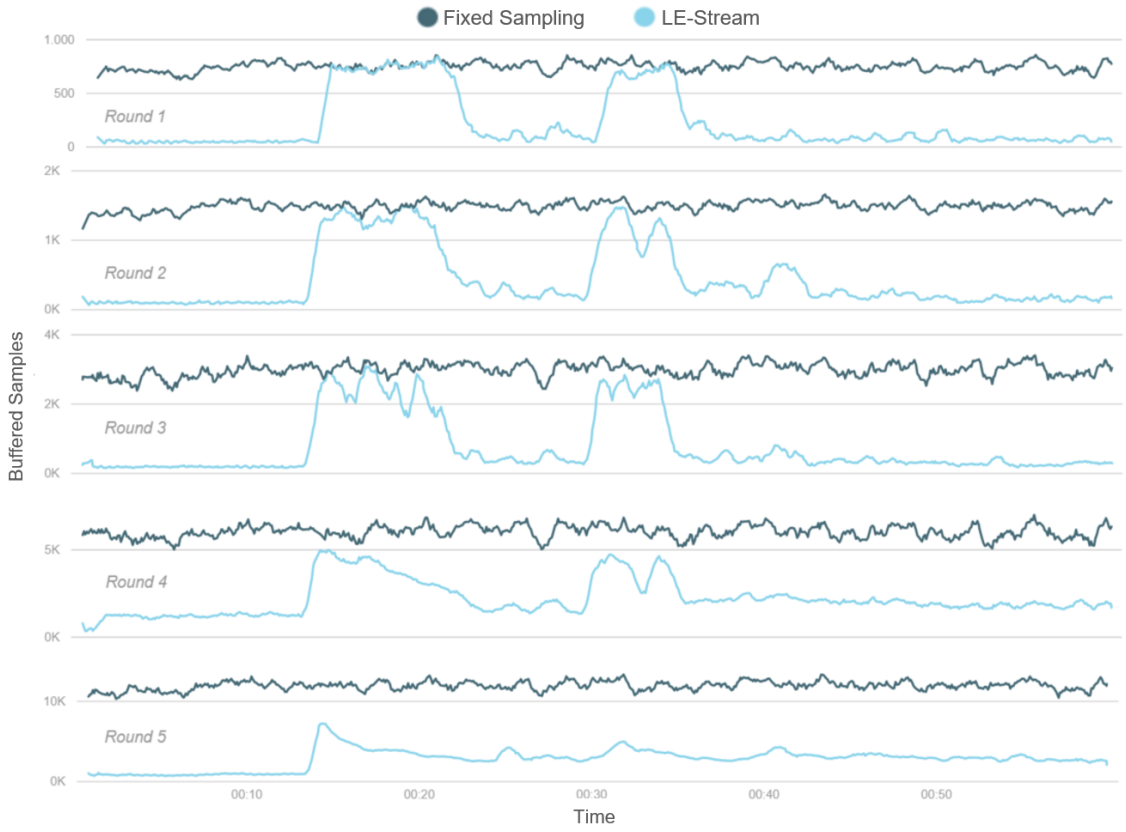

Figure 5.1: Number of samples at the buffer over time for each sampling strategy.

temperature. It is possible to say that scaling up sensors and computing resources does not affect RMSE and MAE.

Despite the considerable difference in the number of samples available in the buffer, both fixed and adaptive sampling modes have a very similar data prediction output. Fig. 5.2 makes it possible to visualize how approximate are the outputs for both sampling modes. The red lines represent temperature readings while the blue lines represent humidity readings. The darker lines correspond to the fixed sampling mode while the lighter ones correspond to LE-Stream with adaptive sampling mode. By comparing rounds 1 to 5 once more, it is also possible to note that the greater the amount of data, the closer are the outputs for the different sampling approaches.

It is important to mention that the sensors redundancy (multi-sensed scenario)

Table 5.2: RMSE and MAE for humidity and temperature per round of simulation.

	Humidity			Temperature		
Round	RMSE	MAE	Range (%)	RMSE	MAE	Range (°C)
1	1.4024	0.5365	(19.91 - 51.01)	0.6954	0.1894	(25.44 - 51.01)
2	1.4519	0.5720	(17.15 - 50.44)	0.8583	0.2517	(25.46 - 57.53)
3	0.9148	0.4110	(16.84 - 52.23)	0.8082	0.3369	(25.59 - 50.65)
4	1.1486	0.3656	(19.06 - 51.23)	0.6642	0.2609	(25.31 - 58.59)
5	1.4887	0.4738	(16.45 - 50.46)	0.9738	0.2384	(25.40 - 58.09)

Figure 5.2: Temperature and humidity readings processed with different sampling strategies.

assumed by LE-Stream contributes directly in terms of data accuracy, as it is already mentioned in Chapter 4. The experiments presented so far support this hypothesis. Furthermore, this sensors redundancy assumption is also an important enabler for the novel adaptive sampling and active node selection strategies proposed in this work and evaluated in the sections to follow.

5.3 Efficiency in Terms of Energy Consumption Reduction

Once it is proved that the reduced number of samples does not significantly affect the accuracy of the output data, we need to verify the benefit of the adaptive sampling strategy in terms of energy consumption. To verify how efficient the proposed framework is on reducing the energy consumption of the sensor nodes when compared to a traditional fixed sampling approach, we use PowerPi [90]. It is a power consumption model to calculate energy consumed by an application running on a Raspberry Pi device (RPi).

To calculate the energy consumption of an application with PowerPi, it is re-

quired that only the application to be measured is running on the device, along with essential operating system tasks. Since our experiments rely on concurrent processes simulating sensor nodes running on shared hosts, an assumption was made to allow this calculation: the energy consumption was measured individually on a physical Raspberry Pi 3 model B+ for each of the four main actions performed by a sensor node. The four main actions performed by a sensor node which were individually measured are:

1. *Sampling*: call the physical sensor to obtain the temperature and humidity values - **0.2104W**.
2. *Sending*: send sampled data to the edge node - **2.5368W**.
3. *Getting window*: get time interval from edge node - **1.0092W**.
4. *Sleeping*: sensor inactive for 1s - **0.1745W**.

The total energy consumption EC_i of a sensor node i is computed as a weighted sum of each of these individual measures plus a constant value of **2.5198W** for idle time [90]. The weights are based on the application logs, where the number of requests R_i and the sleep time S_i of the sensor node is registered:

$$EC_i = (0.2104 + 2.5368 + 1.0092)R_i + 0.1745S_i + 2.5198 \tag{5.1}$$

The assumption made can lead to calculated values that might not precisely represent the actual energy consumption of the devices. However, for a strictly comparative analysis between the scenarios, the calculations performed are valid. Figures 5.3 and 5.4 represent, respectively, the average energy consumption per sensor node and total energy consumed by sensor nodes for each round of simulation. Table 5.3 indicates the energy savings achieved for sensor nodes by LE-Stream (RTE-S) in comparison with Fixed Sampling (FS) in both . LE-Stream leads to an average energy consumption savings per sensor node ranging from 42.95% to 60.59% and a total energy consumed by sensor nodes savings ranging from 42.96% to 75.30%, which is a very significant result. It is also possible to state, from the same Fig. 5.3, that actions which involve communication between the sensor and edge nodes (specially “Sending") are the most expensive in terms of energy consumption. The LE-Stream essentially replaces the “Sending" activity by less expensive actions: “Sleeping", which is the least expensive activity and “Getting" which is also a communication activity but with much smaller payloads.

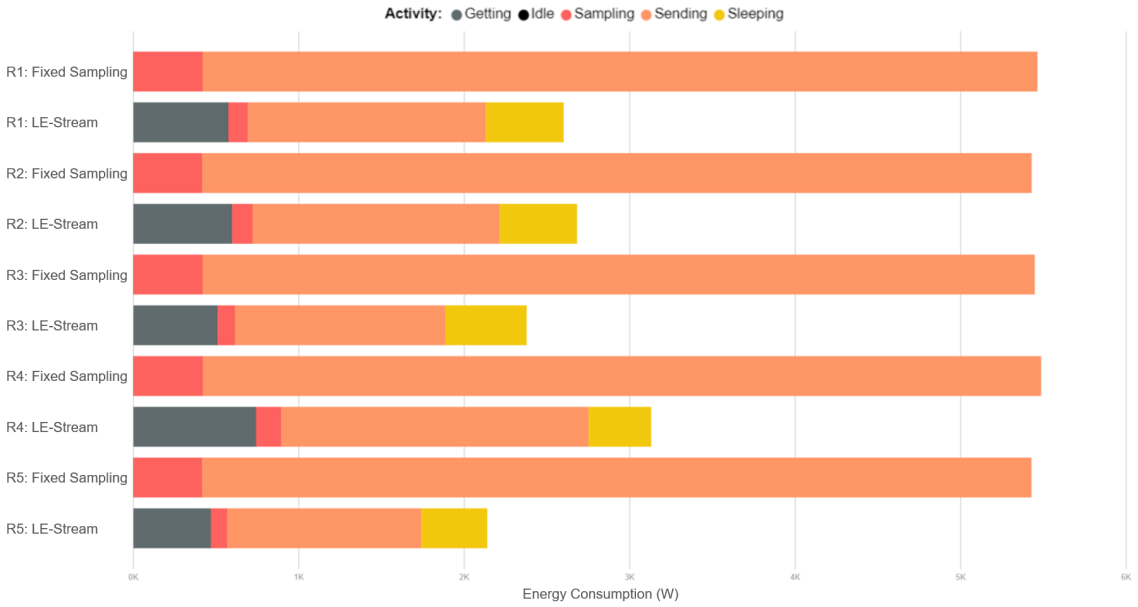

Figure 5.3: Average energy consumption per sensor node for each round.

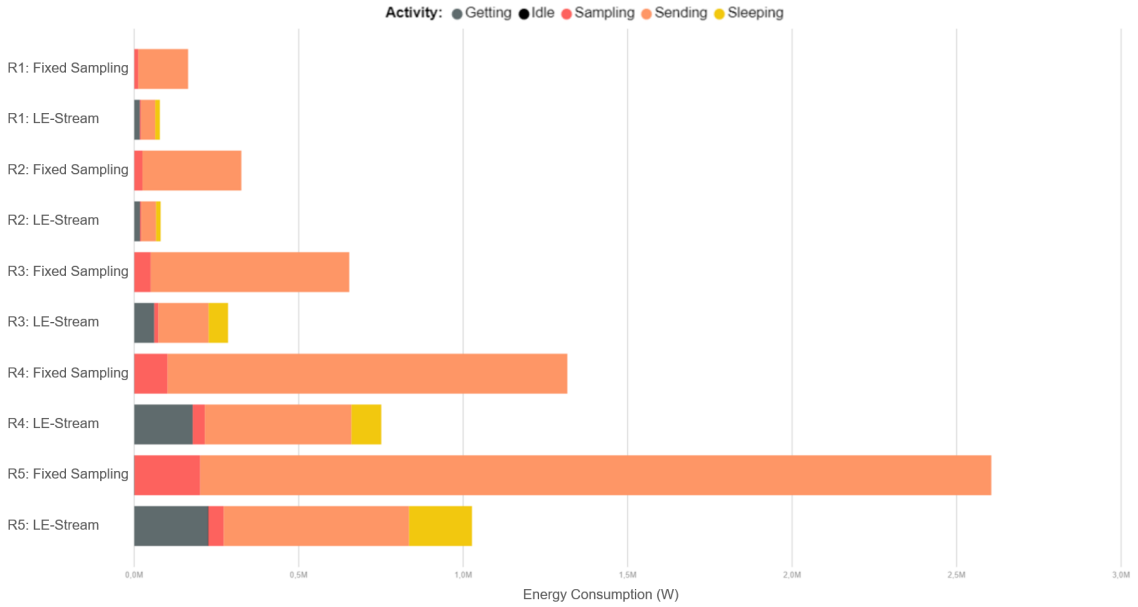

Figure 5.4: Total energy consumed by sensor nodes for each round.

Table 5.3: Energy savings for sensor nodes.

	Average per Sensor Node (W)			Total Consumption (W)		
Round	FS	LE-S	Savings	FS	LE-S	Savings
1	1,092.82	520.15	52.40%	163,922.39	78,021.86	52.40%
2	1,085.82	536.34	50.61%	325,745.93	80,451.68	75.30%
3	1,089.57	475.46	56.36%	653,744.54	285,277.32	56.36%
4	1,097.22	625.91	42.95%	1,316,664.64	751,087.01	42.96%
5	1,085.59	427.84	60.59%	2,605,418.02	1,026,822.90	60.59%

5.4 Effectiveness of Active Node Selection on Improving the Workload Distribution

Once it was demonstrated how efficient LE-Stream is in terms of energy consumption, it is also important to understand how effective the active node selection feature is on solving the poor workload distribution problem described on Chapter 4.3. To make this evaluation possible, the application logs include the computation of the standard deviation metric over the statistics maintained by the Selector component. The standard deviation is calculated each 30s for each simulation round. Also, the simulation rounds were executed again with the Selector component disabled to allow understanding the standard deviation behaviour in LE-Stream with and without the active node selection feature enabled. Fig 5.5 shows the standard deviation metric calculated over time for each round of simulation for both cases. The red lines represent the calculated standard deviation for LE-Stream with the active node selection feature disabled while the blue lines represent the calculated standard deviation for LE-Stream with the active node selection feature enabled. Table 5.4 shows the standard deviation metric calculated at the end of each simulation round either with active node selection (ANS) enabled and disabled and the relation ANS Enabled / ANS Disables to measure the effectiveness of the ANS feature.

Table 5.4: Standard deviation calculated at the end of each simulation round.

Round	ANS Disabled	ANS Enabled	Relation
1	251.53	157.59	62.65%
2	222.77	72.61	32.59%
3	258.76	68.76	26.57%
4	215.75	12.67	5.87%
5	334.98	72.62	21.68%

Regardless of whether the ANS feature is enabled or disabled, the tendency for the standard deviation to increase over time is always true. This means that there is always a trend for imbalance in the workload distribution over time. However, the active node selection feature proved to be very effective in contributing to a better workload distribution by slowing down the growth of the standard deviation in all rounds of simulation. With a better distribution of the workload between the sensor nodes, not only the occurrence of samples with incorrect values but also the death of sensors due to lack of energy take longer to happen. In this way, it can be said that the ANS feature contributes to increasing the overall operating life of the system, increasing the time during which there is coverage of the monitored phenomenon.

It is possible to claim that the effectiveness of the active node selection feature is greater in scenarios with larger numbers of sensor nodes, as shown in Figure 5.5.

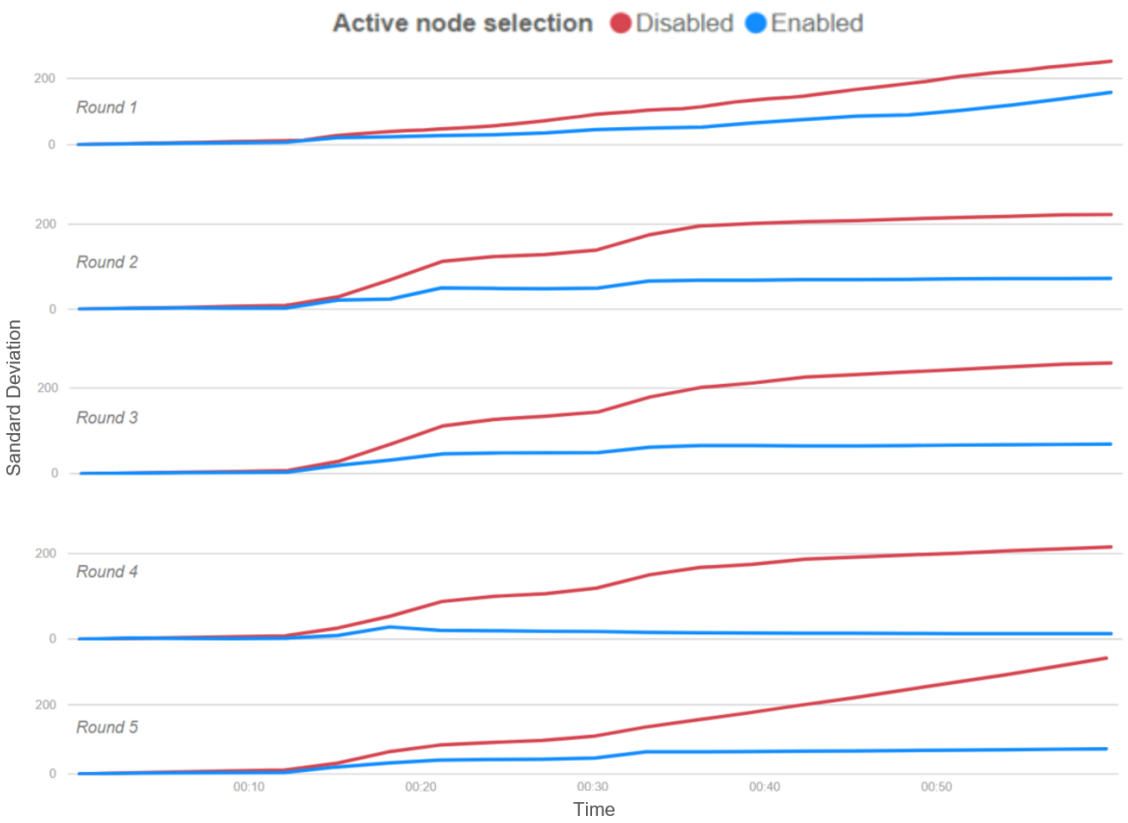

Figure 5.5: Progress of the standard deviation metric on LE-Stream with and without the active node selection feature.

However, it is not possible to establish a direct relationship between the effectiveness of the feature and the increase in the number of sensor nodes. As it can be seen in Table 5.4, on the simulations performed the ANS feature was more effective in round 4 than in round 5.

5.5 Data Stream Processing Speed

Earlier research on computer response times suggests that:

- **0.1 s** is the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result [91];
- **1.0 s** is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1s but less than 1.0s [91];
- **2.0 s** is the limit where response to simple commands becomes unacceptable to users [92].

According to the definitions above, to meet the strict response time requirement we consider 1.0 s as a threshold for the data prediction task's runtime. To achieve such goal on this multi-sensed environment, it is required to choose fast and non compute intensive algorithms when implementing the Predictor component to run on an edge device. DBSCAN was chosen for data cleaning / pre-processing due to its good capability of finding arbitrarily shaped clusters, what makes it robust to outlier detection [83]. The experiments carried involve continuous variables, which are real values over a non-empty range. Thus, We believe that a simple average is able to fairly represent the temperature and humidity values given a set of samples over a given short period of time. Thus, the statistical Mean function was chosen as the aggregation function for this implementation.

In all rounds of simulations the data prediction process is called once every five seconds to generate the output data which were used to plot the graphs of Fig. 5.1 and Fig. 5.2. The time elapsed in each run was also registered as an attribute on the output data. This information allowed us to evaluate how fast the data is being processed, considering the data cleaning and aggregation tasks. Table 5.5 shows how many times the data prediction task ran (Count), the average speed time in seconds and its respective standard deviation (Std. Deviation) for each simulation round. We highlight that the average processing speed in all rounds of simulation is far below the established threshold of 1 s. For round 5, which involves the largest volume of data processed, the average represents only 25% of this threshold. Fig 5.6 shows how the data prediction processing speed increases as the volume of data processed (number of buffered samples) grows. The graph presents a linear progression, which suggests a good scalability of the proposed solution using the chosen algorithms. There were actually a very small number of data points (9) above the threshold line (1.0 s), which represent less than 0.3% of the total and therefore can be considered outliers. These outliers are outside the plotting area to provide a better view of the relevant part of the data. It is important to highlight that even with a number of samples in the order of 7 thousand units, the processing time remains below 0.6 s.

Table 5.5: Average data prediction processing speed per simulation round.

Round	Count	Average (s)	Std. Deviation
1	613	0.03	0.03
2	687	0.05	0.04
3	962	0.08	0.08
4	961	0.18	0.10
5	684	0.25	0.28

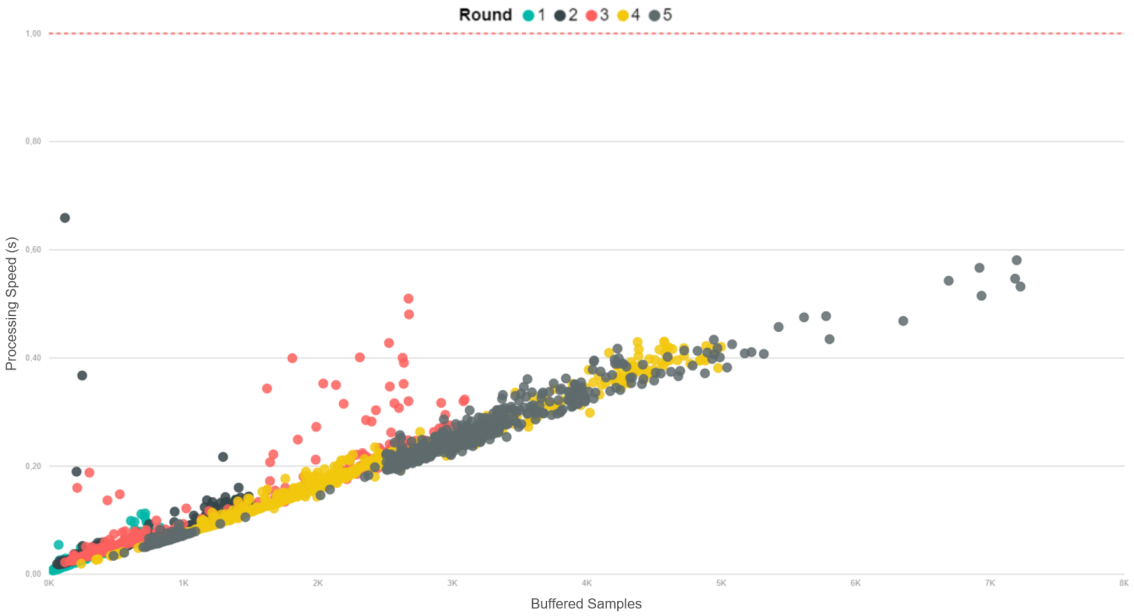

Figure 5.6: Data prediction processing speed per amount of buffered samples

Chapter 6

Conclusion and Future Work

This work presented LE-Stream, a latency and energy-aware DSP framework for edge based IoT systems. LE-Stream contributes to advance the state of the art on IoT data processing by addressing the latency and energy dimensions of the data stream processing problem without compromising data accuracy. Experiments show that a combination of an adaptive sampling strategy with a data prediction model was very effective in reducing the number of samples and network traffic. Being energy efficient, LE-Stream has reduced the average energy consumption of sensor nodes up to 60.58%. An active node selection feature improves the workload distribution among the sensor nodes which also contributes to increasing the overall operating life of the system, thus increasing the time during which there is coverage of the monitored phenomenon. The experimental results described in Chapter 5.5 indicate that the proposed data prediction model deployed at the edge of the network successfully addresses strict response time requirements by meeting a strict execution time threshold of 1s for the data prediction activity. The main contribution of LE-Stream is its capability of tackling strict response time processing, energy consumption and data accuracy requirements all together. Thus, it might be used to enable the development of long running latency-aware DSP IoT systems in remote outdoor environments, where energy sources are scarce and it undesirable or unfeasible replacing batteries frequently.

In the current version of LE-Stream, collaboration between edge nodes is not considered, but we believe that this is an important direction for future research. Such collaboration can allow building a broader view of the phenomena monitored in the environment. It can also lead to more efficient decisions regarding adapting data sampling rates and shutting down redundant sensors. For example, if collaboration allows the identification of several edge nodes monitoring the same phenomenon, adaptation decisions can be made involving a larger geographic area, without the need to involve the cloud. Moreover, such collaboration between edge nodes could leverage the usage of LE-Stream to the federated learning field.

Also, LE-Stream does not consider the mobility factor of sensor nodes, where a certain sensor node that communicates with a gateway A starts to communicate with a gateway B. The active node selection scheme needs to know the activity history of the sensor nodes involved in the sensing activity in question to perform their calculations correctly. Since the mobility of sensor nodes can be not only a characteristic but a critical factor for certain IoT applications, adding this sensor node mobility capability to LE-Stream would also be a very interesting direction for feature research.

Even though LE-Stream had already achieved very significant results in terms of energy consumption reduction, it can be possible to achieve an even greater reduction by improving the communication model between sensor nodes and the gateway. Currently the sensor nodes (Sampler) make 2 requests (*PUT /sample* and *GET /time_window*) when sending samples to the gateway (Gatherer), as of depicted on Fig. 4.4. By following the principle of exchanging communication operations for computing operations as described on Chapter 2.5. Finding a way to adapt the CoAP protocol so that the Sampler could send back the *time_window* value to the Sampler when responding to the *PUT /sample* request would also be a promising research direction.

Although the framework presents promising results when identifying sudden variations in monitored phenomena, studying other algorithms for computing time intervals can also be an opportunity for increasing the framework's reactivity to such variations. TCP congestion control is proved as a good solution, but there might a better solution for the specific purpose of reacting to sudden variations.

References

[1] ATZORI, L., IERA, A., MORABITO, G. "The Internet of Things: A survey", *Computer Networks*, v. 54, pp. 2787–2805, 2010. ISSN: 1389-1286. doi: 10.1016/j.comnet.2010.05.010.

[2] AKYILDIZ, I., SU, W., SANKARASUBRAMANIAM, Y., et al. "Wireless sensor networks: a survey", *Computer Networks*, v. 38, pp. 393–422, 2002. ISSN: 1389-1286. doi: https://doi.org/10.1016/S1389-1286(01)00302-4.

[3] DIAS DE ASSUNÇÃO, M., DA SILVA VEITH, A., BUYYA, R. "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions", *Journal of Network and Computer Applications*, v. 103, pp. 1–17, 2018. ISSN: 1084-8045. doi: 10.1016/j.jnca.2017.12.001.

[4] KARKOUCH, A., MOUSANNIF, H., AL MOATASSIME, H., et al. "Data quality in internet of things: A state-of-the-art survey", *Journal of Network and Computer Applications*, v. 73, pp. 57–81, 2016. ISSN: 1084-8045. doi: 10.1016/j.jnca.2016.08.002.

[5] QIN, Y., SHENG, Q. Z., FALKNER, N. J., et al. "When Things Matter", *J. Netw. Comput. Appl.*, v. 64, pp. 137–153, abr. 2016. ISSN: 1084-8045. doi: 10.1016/j.jnca.2015.12.016.

[6] KLEIN, A., LEHNER, W. "How to Optimize the Quality of Sensor Data Streams", *4th International Multi-Conference on Computing in the Global Information Technology, ICCGI 2009*, 01 2009. doi: 10.1109/ICCGI.2009.10.

[7] DAUTOV, R., DISTEFANO, S., BRUNEO, D., et al. "Pushing Intelligence to the Edge with a Stream Processing Architecture". In: *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 792–799, 2017. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.121.

[8] JANJUA, Z. H., VECCHIO, M., ANTONINI, M., et al. "IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge", *Engineering Applications of Artificial Intelligence*, v. 84, pp. 41–50, 2019. ISSN: 0952-1976. doi: 10.1016/j.engappai.2019.05.011.

[9] ANASTASI, G., CONTI, M., DI FRANCESCO, M., et al. "Energy conservation in wireless sensor networks: A survey", *Ad Hoc Networks*, v. 7, pp. 537–568, 2009. ISSN: 1570-8705. doi: 10.1016/j.adhoc.2008.06.003.

[10] LE BORGNE, Y.-A., SANTINI, S., BONTEMPI, G. "Adaptive Model Selection for Time Series Prediction in Wireless Sensor Networks", *Signal Process.*, v. 87, pp. 3010–3020, dez. 2007. ISSN: 0165-1684. doi: 10.1016/j.sigpro.2007.05.015.

[11] GIOUROUKIS, D., DADIANI, A., TRAUB, J., et al. "A Survey of Adaptive Sampling and Filtering Algorithms for the Internet of Things". In: *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*, DEBS '20, p. 27–38, New York, NY, USA, 2020. Association for Computing Machinery. ISBN: 9781450380287. doi: 10.1145/3401025.3403777.

[12] ABABNEH, N. "Evaluation of On/Off scheduling protocols for ad hoc and sensor networks". In: *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, pp. 419–423, 2010. doi: 10.1109/WCINS.2010.5544122.

[13] RAHMAN, A., JADOON, W., KHAN, F. "Energy Efficiency techniques in cloud computing", *International Journal of Computer Science and Information Security (IJCSIS)*, v. 14, pp. 317–323, 07 2016.

[14] DELICATO, F. C. *Service-Oriented Middleware for Wireless Sensor Networks*. Tese de Doutorado, Universidade Federal do Rio de Janeiro - Electrical Engineering Department, Rio de Janeiro, Brazil, 6 2005.

[15] AL-FUQAHA, A., GUIZANI, M., MOHAMMADI, M., et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", *IEEE Communications Surveys Tutorials*, v. 17, pp. 2347–2376, 2015. doi: 10.1109/COMST.2015.2444095.

[16] KRČO, S., POKRIĆ, B., CARREZ, F. "Designing IoT architecture(s): A European perspective". In: *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pp. 79–84, 2014. doi: 10.1109/WF-IoT.2014.6803124.

[17] LI, W., SANTOS, I., DELICATO, F. C., et al. "System modelling and performance evaluation of a three-tier Cloud of Things", *Future Generation Computer Systems*, v. 70, pp. 104–125, 2017. ISSN: 0167-739X. doi: 10.1016/j.future.2016.06.019.

[18] TAHERKORDI, A., ELIASSEN, F., MCDONALD, M., et al. "Context-Driven and Real-Time Provisioning of Data-Centric IoT Services in the Cloud", *ACM Trans. Internet Technol.*, v. 19, nov. 2018. ISSN: 1533-5399. doi: 10.1145/3151006.

[19] PAETZ, C. *Z-Wave Essentials*. North Charleston, SC, USA, CreateSpace Independent Publishing Platform, 2018. ISBN: 171870822X.

[20] PORCU, G., BURON, J., BRANDT, A. "Home Automation Routing Requirements in Low-Power and Lossy Networks". RFC 5826, abr. 2010.

[21] NIEMINEN, J., SAVOLAINEN, T., ISOMAKI, M., et al. "IPv6 over BLUETOOTH(R) Low Energy". RFC 7668, out. 2015.

[22] HONKANEN, M., LAPPETELAINEN, A., KIVEKAS, K. "Low end extension for Bluetooth". In: *Proceedings. 2004 IEEE Radio and Wireless Conference (IEEE Cat. No.04TH8746)*, pp. 199–202, 2004. doi: 10.1109/RAWCON.2004.1389107.

[23] MONTENEGRO, G., SCHUMACHER, C., KUSHALNAGAR, N. "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals". RFC 4919, ago. 2007.

[24] CHESHIRE, S., KROCHMAL, M. "Multicast DNS". RFC 6762, fev. 2013.

[25] CHESHIRE, S., KROCHMAL, M. "DNS-Based Service Discovery". RFC 6763, fev. 2013.

[26] SHELBY, Z., HARTKE, K., BORMANN, C. "The Constrained Application Protocol (CoAP)". RFC 7252, jun. 2014.

[27] HUNKELER, U., TRUONG, H. L., STANFORD-CLARK, A. "MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks". In: *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pp. 791–798, 2008. doi: 10.1109/COMSWA.2008.4554519.

[28] OASIS. "Advanced Message Queuing Protocol (AMQP) Version 1.0". 2012.

[29] FIELDING, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[30] RICHARDS, R. "Representational State Transfer (REST)". In: *Pro PHP XML and Web Services*, Apress, pp. 633–672, Berkeley, CA, 2006. ISBN: 978-1-4302-0139-7. doi: 10.1007/978-1-4302-0139-7_17.

[31] RAWAT, P., SINGH, K. D., CHAOUCHI, H., et al. "Wireless sensor networks: a survey on recent developments and potential synergies", *The Journal of Supercomputing*, v. 68, pp. 1–48, Apr 2014. ISSN: 1573-0484. doi: 10.1007/s11227-013-1021-9.

[32] FIELDING, R. T., RESCHKE, J. "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content". RFC 7231, jun. 2014.

[33] ZHU, Q., WANG, R., CHEN, Q., et al. "IOT Gateway: BridgingWireless Sensor Networks into Internet of Things". In: *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 347–352, 2010. doi: 10.1109/EUC.2010.58.

[34] WANT, R., FARKAS, K., NARAYANASWAMI, C. "Guest Editors' Introduction: Energy Harvesting and Conservation", *IEEE Pervasive Computing*, v. 4, pp. 14–17, 2005. doi: 10.1109/MPRV.2005.12.

[35] RAGHUNATHAN, V., SCHURGERS, C., PARK, S., et al. "Energy-aware wireless microsensor networks", *IEEE Signal Processing Magazine*, v. 19, pp. 40–50, 2002. doi: 10.1109/79.985679.

[36] POTTIE, G. J., KAISER, W. J. "Wireless Integrated Network Sensors", *Commun. ACM*, v. 43, pp. 51–58, maio 2000. ISSN: 0001-0782. doi: 10.1145/332833.332838.

[37] KARL, H., WILLIG, A. *Protocols and Architectures for Wireless Sensor Networks*. Hoboken, NJ, USA, John Wiley & Sons, Inc., 2005. ISBN: 0470095105.

[38] SANTI, P. "Topology Control in Wireless Ad Hoc and Sensor Networks", *ACM Comput. Surv.*, v. 37, pp. 164–194, jun. 2005. ISSN: 0360-0300. doi: 10.1145/1089733.1089736.

[39] KESHAVARZIAN, A., LEE, H., VENKATRAMAN, L. "Wakeup Scheduling in Wireless Sensor Networks". In: *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '06,

p. 322–333, New York, NY, USA, 2006. Association for Computing Machinery. ISBN: 1595933689. doi: 10.1145/1132905.1132941.

[40] DEMIRKOL, I., ERSOY, C., ALAGOZ, F. "MAC protocols for wireless sensor networks: a survey", *IEEE Communications Magazine*, v. 44, pp. 115–121, 2006. doi: 10.1109/MCOM.2006.1632658.

[41] LANGENDOEN, K. "Medium access control in wireless sensor networks", *Medium Access Control Wirel. Netw.*, v. 2, 11 2007.

[42] RAGHUNATHAN, V., GANERIWAL, S., SRIVASTAVA, M. "Emerging techniques for long lived wireless sensor networks", *IEEE Communications Magazine*, v. 44, pp. 108–114, 2006. doi: 10.1109/MCOM.2006.1632657.

[43] TANG, C., RAGHAVENDRA, C. S. "Compression Techniques for Wireless Sensor Networks". In: Raghavendra, C. S., Sivalingam, K. M., Znati, T. (Eds.), *Wireless Sensor Networks*, Springer US, pp. 207–231, Boston, MA, 2004. ISBN: 978-1-4020-7884-2. doi: 10.1007/978-1-4020-7884-2_10.

[44] FASOLO, E., ROSSI, M., WIDMER, J., et al. "In-network aggregation techniques for wireless sensor networks: a survey", *IEEE Wireless Communications*, v. 14, pp. 70–87, 2007. doi: 10.1109/MWC.2007.358967.

[45] RISTESKA STOJKOSKA, B., MAHOSKI, K. "Comparison of Different Data Prediction Methods for Wireless Sensor Networks". In: *Proceedings of the 10th Conference for Informatics and Information Technology*, 04 2013.

[46] AKYILDIZ, I. F., KASIMOGLU, I. H. "Wireless sensor and actor networks: research challenges", *Ad Hoc Networks*, v. 2, pp. 351–367, 2004. ISSN: 1570-8705. doi: 10.1016/j.adhoc.2004.04.003.

[47] MORABITO, R., COZZOLINO, V., DING, A. Y., et al. "Consolidate IoT Edge Computing with Lightweight Virtualization", *IEEE Network*, v. 32, pp. 102–111, 2018. doi: 10.1109/MNET.2018.1700175.

[48] FEI, X., SHAH, N., VERBA, N., et al. "CPS data streams analytics based on machine learning for Cloud and Fog Computing: A survey", *Future Generation Computer Systems*, v. 90, pp. 435–450, 2019. ISSN: 0167-739X. doi: https://doi.org/10.1016/j.future.2018.06.042.

[49] BABCOCK, B., BABU, S., DATAR, M., et al. "Models and Issues in Data Stream Systems". In: *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS

'02, p. 1–16, New York, NY, USA, 2002. Association for Computing Machinery. ISBN: 1581135076. doi: 10.1145/543613.543615.

[50] ISAH, H., ABUGHOFA, T., MAHFUZ, S., et al. "A Survey of Distributed Data Stream Processing Frameworks", *IEEE Access*, v. 7, pp. 154300–154316, 2019. doi: 10.1109/ACCESS.2019.2946884.

[51] GAMA, J. A., ŽLIOBAITUNDEFINED, I., BIFET, A., et al. "A Survey on Concept Drift Adaptation", *ACM Comput. Surv.*, v. 46, n. 4, mar 2014. ISSN: 0360-0300. doi: 10.1145/2523813.

[52] AGGARWAL, C. C. "Mining Sensor Data Streams". In: *Managing and Mining Sensor Data*, Springer US, pp. 143–171, Boston, MA, 2013. ISBN: 978-1-4614-6309-2.

[53] TSAI, C.-W., LAI, C.-F., CHIANG, M.-C., et al. "Data Mining for Internet of Things: A Survey", *IEEE Communications Surveys Tutorials*, v. 16, pp. 77–97, 2014. doi: 10.1109/SURV.2013.103013.00206.

[54] MOHAMMADI, M., AL-FUQAHA, A., SOROUR, S., et al. "Deep Learning for IoT Big Data and Streaming Analytics: A Survey", *IEEE Communications Surveys Tutorials*, v. 20, pp. 2923–2960, 2018. doi: 10.1109/COMST.2018.2844341.

[55] SHOBANADEVI, A., MARAGATHAM, G. "Data mining techniques for IoT and big data — A survey". In: *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 607–610, 2017. doi: 10.1109/ISS1.2017.8389260.

[56] LI, S., XU, L. D., ZHAO, S. "The internet of things: a survey", *Information Systems Frontiers*, v. 17, pp. 243–259, Apr 2015. ISSN: 1572-9419. doi: 10.1007/s10796-014-9492-7.

[57] MEHMOOD, E., ANEES, T. "Challenges and Solutions for Processing Real-Time Big Data Stream: A Systematic Literature Review", *IEEE Access*, v. 8, pp. 119123–119143, 2020. doi: 10.1109/ACCESS.2020.3005268.

[58] SAMIZADEH NIKOUI, T., RAHMANI, A. M., BALADOR, A., et al. "Internet of Things architecture challenges: A systematic review", *International Journal of Communication Systems*, v. 34, pp. e4678, 2021. doi: 10.1002/dac.4678.

[59] XU, Y., HELAL, A. "Scalable Cloud–Sensor Architecture for the Internet of Things", *IEEE Internet of Things Journal*, v. 3, pp. 285–298, 2016. doi: 10.1109/JIOT.2015.2455555.

[60] CATARINUCCI, L., DE DONNO, D., MAINETTI, L., et al. "An IoT-Aware Architecture for Smart Healthcare Systems", *IEEE Internet of Things Journal*, v. 2, pp. 515–526, 2015. doi: 10.1109/JIOT.2015.2417684.

[61] LORIA, M. P., TOJA, M., CARCHIOLO, V., et al. "An efficient real-time architecture for collecting IoT data". In: *2017 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 1157–1166, 2017. doi: 10.15439/2017F381.

[62] VUČINIĆ, M., TOURANCHEAU, B., ROUSSEAU, F., et al. "OSCAR: Object security architecture for the Internet of Things". In: *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pp. 1–10, 2014. doi: 10.1109/WoWMoM.2014.6918975.

[63] WANG, T., KE, H., ZHENG, X., et al. "Big Data Cleaning Based on Mobile Edge Computing in Industrial Sensor-Cloud", *IEEE Transactions on Industrial Informatics*, v. 16, pp. 1321–1329, 2020. doi: 10.1109/TII.2019.2938861.

[64] MONTEIRO, L. C., DELICATO, F. C., PIRMEZ, L., et al. "DPCAS: Data Prediction with Cubic Adaptive Sampling for Wireless Sensor Networks". In: Au, M. H. A., Castiglione, A., Choo, K.-K. R., et al. (Eds.), *Green, Pervasive, and Cloud Computing*, pp. 353–368, Cham, 2017. Springer International Publishing. ISBN: 978-3-319-57186-7.

[65] AL-HOQANI, N., YANG, S.-H. "Adaptive Sampling for Wireless Household Water Consumption Monitoring", *Procedia Engineering*, v. 119, pp. 1356–1365, 2015. ISSN: 1877-7058. doi: 10.1016/j.proeng.2015.08.980.

[66] GUPTA, M., SHUM, L. V., BODANESE, E., et al. "Design and evaluation of an adaptive sampling strategy for a wireless air pollution sensor network". In: *2011 IEEE 36th Conference on Local Computer Networks*, pp. 1003–1010, 2011. doi: 10.1109/LCN.2011.6115154.

[67] SARKAR, C., RAO, V. S., VENKATESHA PRASAD, R., et al. "VSF: An Energy-Efficient Sensing Framework Using Virtual Sensors", *IEEE Sensors Journal*, v. 16, pp. 5046–5059, 2016. doi: 10.1109/JSEN.2016.2546839.

[68] VIKASH, MISHRA, L., VARMA, S. "Performance evaluation of real-time stream processing systems for Internet of Things applications", *Future Generation Computer Systems*, v. 113, pp. 207–217, 2020. ISSN: 0167-739X. doi: 10.1016/j.future.2020.07.012.

[69] XHAFA, F., KILIC, B., KRAUSE, P. "Evaluation of IoT stream processing at edge computing layer for semantic data enrichment", *Future generation computer systems*, v. 105, pp. 730–736, Apr 2020. doi: 10.1016/j.future.2019.12.031.

[70] OUNACER, S., TALHAOUI, M. A., ARDCHIR, S., et al. "A New Architecture for Real Time Data Stream Processing", *International Journal of Advanced Computer Science and Applications*, v. 8, 2017. doi: 10.14569/IJACSA.2017.081106.

[71] DAUTOV, R., DISTEFANO, S. "Stream Processing on Clustered Edge Devices", *IEEE Transactions on Cloud Computing*, pp. 1–1, 2020. doi: 10.1109/TCC.2020.2983402.

[72] LIU, D., YAN, Z., DING, W., et al. "A Survey on Secure Data Analytics in Edge Computing", *IEEE Internet of Things Journal*, v. 6, pp. 4946–4967, 2019. doi: 10.1109/JIOT.2019.2897619.

[73] GARG, S., SINGH, A., KAUR, K., et al. "Edge Computing-Based Security Framework for Big Data Analytics in VANETs", *IEEE Network*, v. 33, pp. 72–81, 2019. doi: 10.1109/MNET.2019.1800239.

[74] GUAN, Z., ZHANG, Y., WU, L., et al. "APPA: An anonymous and privacy preserving data aggregation scheme for fog-enhanced IoT", *Journal of Network and Computer Applications*, v. 125, pp. 82–92, 2019. ISSN: 1084-8045. doi: 10.1016/j.jnca.2018.09.019.

[75] CHAUHAN, R., KAUR, H., CHANG, V. "An Optimized Integrated Framework of Big Data Analytics Managing Security and Privacy in Healthcare Data", *Wireless Personal Communications*, v. 117, pp. 87–108, Mar 2021. ISSN: 1572-834X. doi: 10.1007/s11277-020-07040-8.

[76] EUGSTER, P. T., FELBER, P. A., GUERRAOUI, R., et al. "The Many Faces of Publish/Subscribe", *ACM Comput. Surv.*, v. 35, pp. 114–131, jun. 2003. ISSN: 0360-0300. doi: 10.1145/857076.857078.

[77] ZHANG, H., CHEN, G., OOI, B. C., et al. "In-Memory Big Data Management and Processing: A Survey", *IEEE Transactions on Knowledge and*

Data Engineering, v. 27, pp. 1920–1948, 2015. doi: 10.1109/TKDE.2015.2427795.

[78] ELMAZI, D., CUKA, M., IKEDA, M., et al. "A Fuzzy-Based System for Actor Node Selection in WSANs Considering Load Balancing of Actors". In: Barolli, L., Leu, F.-Y., Enokido, T., et al. (Eds.), *Advances on Broadband and Wireless Computing, Communication and Applications*, pp. 97–109, Cham, 2019. Springer International Publishing. ISBN: 978-3-030-02613-4.

[79] HA, S., RHEE, I., XU, L. "CUBIC: A New TCP-Friendly High-Speed TCP Variant", *SIGOPS Oper. Syst. Rev.*, v. 42, pp. 64–74, jul. 2008. ISSN: 0163-5980. doi: 10.1145/1400097.1400105.

[80] SALKIND, N. *Encyclopaedia of Research Design, Vol. 1.* Oaks, CA, Sage Publications, 2010.

[81] HYNDMAN, R., ATHANASOPOULOS, G. *Forecasting: Principles and Practice.* Australia, OTexts, 2014.

[82] WRIGHT, D. J. "Forecasting Data Published at Irregular Time Intervals Using an Extension of Holt's Method", *Management Science*, v. 32, pp. 499–510, 1986.

[83] CAMPELLO, R. J. G. B., MOULAVI, D., SANDER, J. "Density-Based Clustering Based on Hierarchical Density Estimates". In: Pei, J., Tseng, V. S., Cao, L., et al. (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 160–172, Berlin, Heidelberg, 2013. Springer. ISBN: 978-3-642-37456-2.

[84] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al. "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, v. 12, pp. 2825–2830, 2011.

[85] SCHUBERT, E., SANDER, J., ESTER, M., et al. "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN", *ACM Trans. Database Syst.*, v. 42, jul. 2017. ISSN: 0362-5915. doi: 10.1145/3068335.

[86] TANGANELLI, G., VALLATI, C., MINGOZZI, E. "CoAPthon: Easy development of CoAP-based IoT applications with Python". In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 63–68, 2015. doi: 10.1109/WF-IoT.2015.7389028.

[87] CARLSON, J. L. *Redis in Action.* USA, Manning Publications Co., 2013. ISBN: 1617290858.

[88] DWYER, G., AGGARWAL, S., STOUFFER, J. *Flask: Building Python Web Services*. UK, Packt Publishing, 2017. ISBN: 1787288226.

[89] CHAI, T., DRAXLER, R. R. "Root mean square error (RMSE) or mean absolute error (MAE)?" *Geoscientific Model Development Discussions*, v. 7, pp. 1525–1534, fev. 2014. doi: 10.5194/gmdd-7-1525-2014.

[90] KAUP, F., GOTTSCHLING, P., HAUSHEER, D. "PowerPi: Measuring and modeling the power consumption of the Raspberry Pi". In: *39th Annual IEEE Conference on Local Computer Networks*, pp. 236–243, 2014. doi: 10.1109/LCN.2014.6925777.

[91] NIELSEN, J. "Chapter 5 - Usability Heuristics". In: NIELSEN, J. (Ed.), *Usability Engineering*, Morgan Kaufmann, pp. 115–163, San Diego, 1993. ISBN: 978-0-12-518406-9. doi: 10.1016/B978-0-08-052029-2.50008-5.

[92] SHNEIDERMAN, B. "Response Time and Display Rate in Human Performance with Computers", *ACM Comput. Surv.*, v. 16, pp. 265–285, set. 1984. ISSN: 0360-0300. doi: 10.1145/2514.2517.