Pontifícia Universidade Católica
do Rio de Janeiro

Alexandre Malheiros Meslin

MUSANet: A Multitier Platform for Developing Smart-City Applications

Tese de Doutorado

Thesis presented to the Programa de Pós–Graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências – Informática.

Advisor : Prof. Noemi de La Rocque Rodriguez
Co-advisor: Prof. Markus Endler

Rio de Janeiro
April 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Alexandre Malheiros Meslin

MUSANet: A Multitier Platform for Developing Smart-City Applications

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Informática. Approved by the Examination Committee:

Prof. Noemi de La Rocque Rodriguez
Advisor
Departamento de Informática – PUC-Rio

Prof. Markus Endler
Co-advisor
Departamento de Informática – PUC-Rio

Prof. Francisco José da Silva e Silva
UFMA

Prof. Alfredo Goldman vel Lejbman
USP

Prof. Flávia Coimbra Delicato
UFF

Prof. Sérgio Colcher
Departamento de Informática – PUC-Rio

Rio de Janeiro, April 16th, 2021

Alexandre Malheiros Meslin

Alexandre Meslin holds a degree in Electronic Engineering from UFRJ, and an M.Sc. in Systems and Computer Engineering from the same University. He is currently working for a Ph.D. degree in Computer Science at PUC-Rio. He is Systems Analyst at the UFRJ, lecturer at the PUC-Rio, and Instructor at CEDERJ. Alexandre's main interests and experience are in Computer Systems, with focus on IoT, WSN, and Smart Cities.

Bibliographic data

Meslin, Alexandre Malheiros

MUSANet: A Multitier Platform for Developing Smart-City Applications / Alexandre Malheiros Meslin; advisor: Noemi de La Rocque Rodriguez; co-advisor: Markus Endler. – 2021.

116 f: il. color. ; 30 cm

Tese (doutorado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2021.

Inclui bibliografia

1. Redes de Computadores – Teses. 2. Sistemas Distribuídos – Teses. 3. Cidades Inteligentes. 4. Internet das Coisas Móveis. 5. IoT. 6. Sistemas Distribuídos. 7. Rede de Computadores. I. Rodriguez, Noemi de La Rocque. II. Endler, Markus. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

To my wife, Bianca, and my daughter, Amanda, who have always been close to me at best and also at worst times, always supporting me during this journey. To my friends for their psychological support and the rare pleasant moments of dinners and games.

I would also like to dedicate to two special people who have always accompanied me and continue to accompany me, even though they are no longer with us, to my mother, Sônia, who always believed in me, and to Denise Cavalcante Meslin.

Acknowledgments

First, I would like to thank my advisors, Dr. Noemi Rodriguez and Dr. Markus Endler, professors in the Departamento de Informática at the Pontifícia Universidade Católica do Rio de Janeiro. This dissertation would not be possible without your guidance and support and for the patience to review the text of this work (and our articles) countless times.

Then, I would also like to thank everybody at the Laboratory for Advanced Collaboration at PUC-Rio and the entire InterSCity project team at the Instituto de Matemática e Estatística at USP. Also the professors and students at the Laboratório de Sistemas Distribuídos Inteligentes at UFMA for being "volunteers" in the first MUSANet course.

Next, thanks to the secretary of the Departamento de Informática at PUC-Rio for simplifying all the bureaucratic work that is part of the doctoral journey, especially to Regina Zanon and Cosme Pereira Leal, because without them, I would have been retired several times!

Finally, I want to thank all the professors of the Departamento de Informática, especially professors Dr. Roberto Ierusalimschy, Dr. Eduardo Sany Laber, Dr. Marco Serpa Molinaro, and Dr. Simone Diniz Junqueira Barbosa, for the classroom teachings that contributed to the basis of that work. I also thank the support of the Department of Informatics that provided the necessary equipment to develop the studies presented in this thesis.

The InterSCity project is sponsored by the INCT of the Future Internet for Smart Cities funded by CNPq, proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) — Finance Code 001, and FAPESP, proc. 2014/50937-1 and 2015/24485-9.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Meslin, Alexandre Malheiros; Rodriguez, Noemi de La Rocque (Advisor); Endler, Markus (Co-Advisor). **MUSANet: A Multitier Platform for Developing Smart-City Applications**. Rio de Janeiro, 2021. 116p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

The number of Smart Cities in the world is growing very fast, and there is no indication that there will be a decrease in this growth soon, because a Smart City helps its managers and inhabitants to enjoy its resources and manage several of its aspects. City administrators are installing sensors and actuators in different parts of the city to collect data and react in real time to expected changes. It is up to computer programmers to design computer systems capable of processing all this data and making it available to inhabitants and administrators as information in an organized manner. To allow programmers to investigate the behavior of applications before they are deployed in the city, we have developed a three-layer testbed that helps programmers analyze performance in a controlled environment that can also be used for implementation. The testbed allows developers to distribute processing, including complex event/data streams, in the cloud, fog or edge. Although the testbed architecture is platform independent, we have implemented a reference version using free software. All the components used in the reference version were evaluated individually by their developers, but to verify the scalability of the integration, we developed several applications to evaluate the behavior of the architecture. Four applications for collecting and processing IoT data have been developed to illustrate how the testbed can guide programmers in choosing the best way to implement their applications. Based on the behavior of the developed applications, we created a taxonomy to classify applications for smart cities according to their characteristics and distribution possibilities to help the developer to implement their application.

Keywords

Smart City; Internet of Mobile Things; IoT; Distributed System; Computer Network.

Resumo

Meslin, Alexandre Malheiros; Rodriguez, Noemi de La Rocque; Endler, Markus. **MUSANet: Uma Plataforma Multi-Camada para Desenvolvimento de Aplicações para Cidades Inteligentes**. Rio de Janeiro, 2021. 116p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O número de Cidades Inteligentes no mundo está crescendo muito rapidamente, e não há indicação de que em breve haverá uma diminuição nesse crescimento, isso porque uma Cidade Inteligente ajuda seus gestores e habitantes a desfrutarem de seus recursos e gerenciarem vários de seus aspectos. Os administradores da cidade estão instalando sensores e atuadores em diferentes partes da cidade para coletar dados e reagir em tempo real às mudanças esperadas. Cabe aos programadores de computadores projetarem sistemas computacionais capazes de processar todos esses dados e disponibilizá-los como informações aos habitantes e administradores de forma organizada. Para permitir que os programadores investiguem o comportamento das aplicações antes de sua implementação na cidade, desenvolvemos um testbed de três camadas que ajuda os programadores a analisar o desempenho em um ambiente controlado que também pode ser usado para implementação. O testbed permite que os desenvolvedores distribuam o processamento, incluindo fluxos de eventos/dados complexos, na nuvem, névoa ou borda. Embora a arquitetura do testbed seja independente de plataformas, nós implementamos uma versão de referência utilizando softwares gratuitos. Todos os componentes utilizados na versão de referência foram avaliados individualmente por seus desenvolvedores, mas para verificar a escalabilidade da integração, nós desenvolvemos várias aplicações para avaliar o comportamento da arquitetura. Nós também desenvolvemos quatro aplicativos para coleta e processamento de dados IoT para ilustrar como o testbed pode guiar os programadores na escolha da melhor forma de implementar suas aplicações. Baseado no comportamento das aplicações desenvolvidas, nós criamos uma taxonomia para classificar as aplicações para cidades inteligentes de acordo com as suas características e possibilidades de distribuição para auxiliar ao desenvolvedor a implementar a sua aplicação.

Palavras-chave

Cidades Inteligentes; Internet das Coisas Móveis; IoT; Sistemas Distribuídos; Rede de Computadores.

Table of contents

List of figures

List of tables

List of Abreviations

3G – Third Generation

4G – Fourth Generation

5G – Fifth Generation

6LoPAN – IPv6 over Low -Power Wireless Personal Area Networks

AJAX – Asynchronous Javascript And XML

ALPR – Automatic License Plate Recognition

AMQP – Advanced Message Queuing Protocol

AWS – Amazon Web Service

BCI – Barcelona Ciudad Inteligente

BLE – Bluetooth Low Energy

CDDL – Context Data Distribution Layer

CEP – Complex Event Processing

DDS – Data Distribution Service

DI – Departamento de Informática

EC2 – Elastic Cloud Computing

Full-HD – Full High Definition (1920x1080 pixels)

HD – High Definition

HTTP – Hypertext Transfer Protocol

HTTPS – Hypertext Transfer Protocol Secure

IEEE – Institute of Electrical and Electronics Engineers

IME – Instituto de Matemática e Estatística da USP

IoMT – Internet of Mobile Things

IoT – Internet of Things

IP – Internet Protocol

IPv4 – Internet Protocol version 4

IPv6 – Internet Protocol version 6

LAC – Laboratory for Advanced Collaboration

LTE – Long-Term Evolution

M2M – Machine-to-Machine

MCS – Mobile Crowd Sensing

M-EPA – Mobile Event Processing Agent

MEC – Mobile Edge Computing

MQTT – Message Queuing Telemetry Transport

MR-UDP – Mobile Reliable UDP

MRTG – Multi Router Traffic Grapher

MUSANet – Mobile Urban Sensor and Actuator Network

OGC – Open Geospatial Consortium

PoA – Point of Attachment

PUC – Pontifícia Universidade Católica

QoE – Quality of Experience

RAM – Random Access Memory

RPL – Routing Protocol for Low-Power and Lossy Networks

RTT – Round Trip Time

RUNES – Reconfigurable, Ubiquitous, Networked, Embedded System

SBRC – Simpósio Brasileiro de Redes de Computadores

SDDL – Scalable Data Distribution Layer

TCP – Transport Control Protocol

USP – Universidade de São Paulo

VM – Virtual Machine

WAN – Wide Area Network

WCGA – Workshop on Clouds and Applications

WSN – Wireless Sensor Network

XML – Extensible Markup Language

Loucura? Sonho?
Tudo é loucura ou sonho no começo. Nada do que o homem fez no mundo teve início de outra maneira — mas tantos sonhos se realizaram que não temos o direito de duvidar de nenhum.

Monteiro Lobato,
Obras Completas, São Paulo, Brasiliense, 1946, 7ª edição, p. 178.

1
Introduction

Large modern cities have become increasingly complex and challenging to handle, as several city services have to be managed concurrently and using the least possible material and human resources. Examples of these services are public transportation [1], traffic [2], ambulant healthcare, garbage collection [3], energy [4], water distribution [5], public security, schools, fire control [6], and many others. Also, more often than desired, occasional harmful events or accidents happen in the city. These events may potentially affect the operation of one or several city services locally. Therefore, they require the city authorities to promptly act to mitigate the impact on the quality of other services.

Due to the increasing need to better understand the citizen's demands for services, plan, optimize and coordinate providers of the city services, and also promptly identify harmful events or accidents, more and more municipal administrators [7] are instrumenting city resources with sensors and cameras. City resources may include streets, bus stations, post lamps, buildings, energy meters, service vehicles, etc. These sensors continuously collect data about all relevant city resources and convey them to municipal operation centers. A team of experts monitors the city information on dashboards after several data analytics and AI algorithms, typically relying on cloud computing, have analyzed them. Moreover, many of these information services (e.g., public transportation schedules, security, and health) are also made available through the location-based web or mobile apps to the citizens, who can use it for organizing their daily tasks.

However, as cities get more and more instrumented with such sensors and cameras, soon the communication infrastructure for uploading sensory data to the cloud and downloading actuation commands to IoT devices (traffic lights, lamp posts, etc.) will become a bottleneck for smart cities. Only very few smart city deployments nowadays are scalable in terms of network usage. Scalability can be achieved through the distribution of processing, which contributes to increased communication between the elements involved. It is up to the programmer to make complex decisions about the trade-off between processing distribution and communication. Aiming to tackle this problem, in this work we propose a three-tier middleware that allows for the developer to experiment

with data processing in different tiers during the design phase and use the same middleware for final deployment.

The remainder of this work is divided as follows. In this chapter, we present the motivation for this work and its goals, and we discuss some problems for developing and deploying smart-city applications we tackle. In Chapter 2, we present some work related to smart cities and testbeds, as well as, we present some work that evaluates multi-layer systems. In Chapter 3, we present the architecture of MUSANet and the systems that make up its three tiers and middleware for IoT in smart cities. In Chapter 4, we present the experiments we performed to evaluate the integration of Mobile Hub, ContextNet, and InterSCity, and the results obtained. In Chapter 5, we present the applications developed to investigate the effects of distributing processing among MUSANet tiers. We end this chapter by discussing the results and what we learned from them. In Chapter 6, we present the conclusions and describe future work.

1.1 Motivation

For the implementation of Smart Cities, it is necessary to have a system capable of managing information from data acquisition using the most diverse kinds of sensors or human interactions, such as their location or demands, throughout its storage for later offline statistical analysis, passing through its agile processing [8, 9]. An example of offline historical analysis would be analyzing seasonal changes in temperature variations, using temperature measurement data recorded at regular intervals. Traffic flow-dependent is an example of a reaction that must be carried out in near real-time.

The architecture of the first smart cities, such as Santander in Spain [10] or EBBITS [11], was based solely on cloud computing. In these early systems, different sensor devices send data to, or are queried by, processing servers in the cloud. In this architecture, all processing is performed in a data center located in the cloud, which becomes a centralizing element of all data and control flow as shown in Figure 1.1. As many sensor devices cannot implement communication protocols over the network and others implement a simplified TCP stack, there is a need to convert their protocols to those used on the Internet through dual-stack gateways. The processing of data carried out by gateways is limited to converting the protocol used by IoT (Internet of things stationary) and IoMT (Internet of things mobile) devices into TCP/IP used by stationary servers on the Internet. If there is an increase on computing demand, administrators can increase processing capacity using more powerful computers or even creating processing server clusters. On the other hand, because all data generated by

the sensors must reach a central point, we can have a bottleneck in the network. Another major problem of a cloud-centric architecture is the lack of control and predictability of the delay and data loss in the communication between sensor and actuation IoT peripheral devices and the processing servers [12]. This lack of control is mostly due to the path that the data must take from source to destination, which is normally under the control or administration of third-party entities, such as mobile network or cable network operators, 4G, LTE, 5G operators.

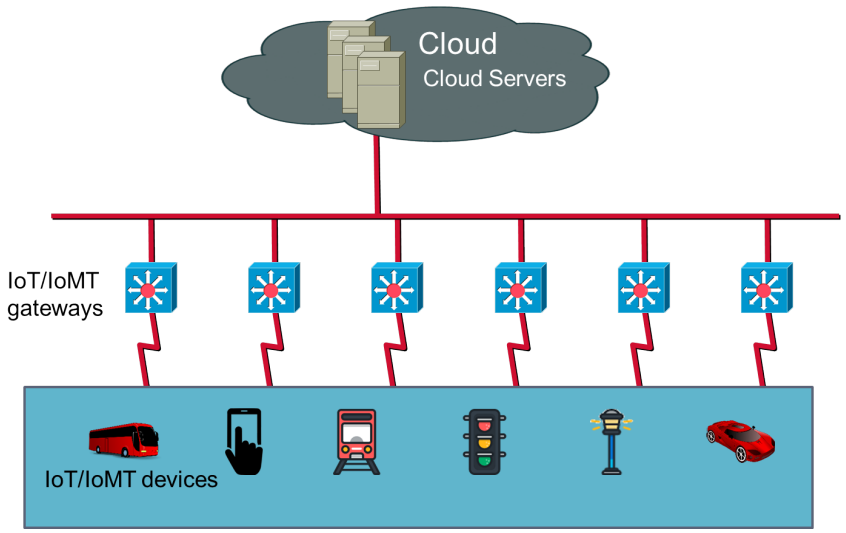


Figure 1.1: Generic single-layer architecture. At the top, there is a computer cluster located at a data center in the cloud. The dual-stack gateway set converts from sensor device (at the bottom) protocols to conventional Internet protocols and vice versa.

The communication delay caused by the data transfer to the hot-spot data center located in the cloud can seriously compromise some types of applications, especially those related to data flow streamed for users (video-conferencing [13], augmented reality [14]), geo-distributed applications (smart traffic light control [15]), or near real-time (smart grid, smart healthcare [16] or emergency response [17]). To try to solve the problem of centralization and bottleneck of a hot spot in the cloud and, at the same time, obtain better control over the communication delay and increase the predictability of the paths between sensor devices and processing servers, many system and software architects [18, 19, 20, 21, 22, 23, 24, 25, 26, 27] advocate the move of part of the processing from the cloud to the fog, bringing it nearer to the source of data, creating a two-tier architecture – cloud and fog.

Fog computing [12] describes a decentralized computing infrastructure located between the cloud and IoT devices that produce data. It is either composed of a cluster of machines, or a single server, that provide storage,

computing and networking resources for "localized IoT data processing" and take off computational load the cloud for some portion of the Smart City application. Its main purpose is to improve (communication and processing performance) for large scale deployments.

Considering this approach, where computing can be performed first in the fog and then at the cloud (Figure 1.2), data generated by the sensor devices are sent to one of the processing servers distributed throughout the fog. The path used to send this data can be implemented using a customized network or even the existing infrastructure, that is, the Internet, 3G/4G structures, etc. The distribution of gateways to access the stationary network infrastructure and processing servers allows for a shorter path in router hops and, consequently, smaller delays between sensor devices and processing servers. Decentralization also eliminates the path concentration, allowing local problems in the computer network not to affect the entire system. On the other hand, near-real-time applications can still suffer interference from communication delay between sensor devices and fog servers, making the user experience unpleasant or even go so far as to make the application unfeasible.

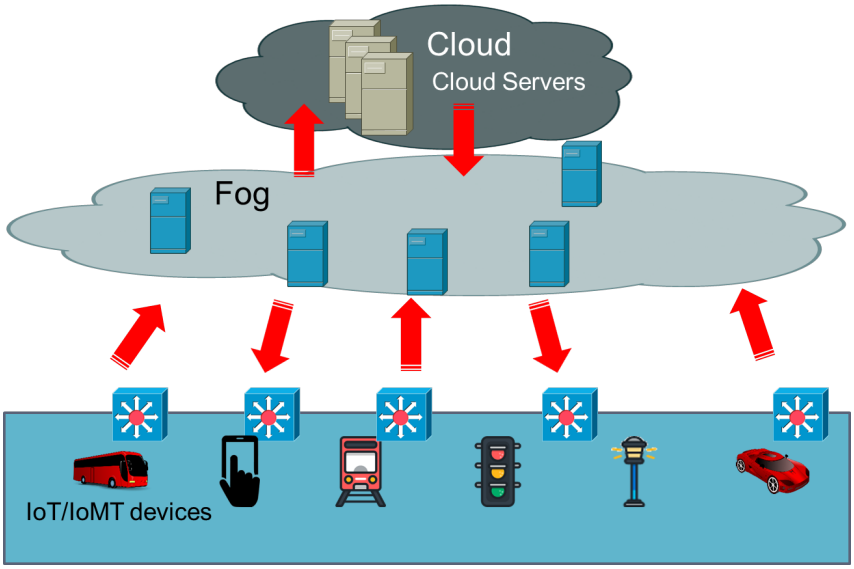


Figure 1.2: Generic two-layer architecture. In this model, data servers are moved closer to the gateways distributed in the city. Data is processed in the fog or cloud.

While designing a two-layer architecture, the developer must be concerned with the decision where processing should be done, that is, whether each processing task will be performed at a (regular) data center in the cloud or in a fog of smaller data-centers distributed across the city's stationary network, or even distributed in both layers. The developer of the smart-city application should be aware that each processing server located in the fog will receive data

generated only from a fraction of the Smart City sensor devices, i.e. the ones that are in the same administrative domain. Although this data distribution allows for distributed computing, it limits the decision-making capacity of each processing server. For a global decision or even a non-local decision, processing servers must exchange information among them.

Moving more processing capabilities towards the sensors, we take the network edge closer to them and create a new layer able to process: the Edge. In such a three-tier architecture, as Figure 1.3 shows, the data generated by the sensor devices can be processed on the edge, in the fog or at the cloud, depending on a multitude of factors, such as, for example, relevance, urgency, quantity, computing complexity, data aggregation, etc.

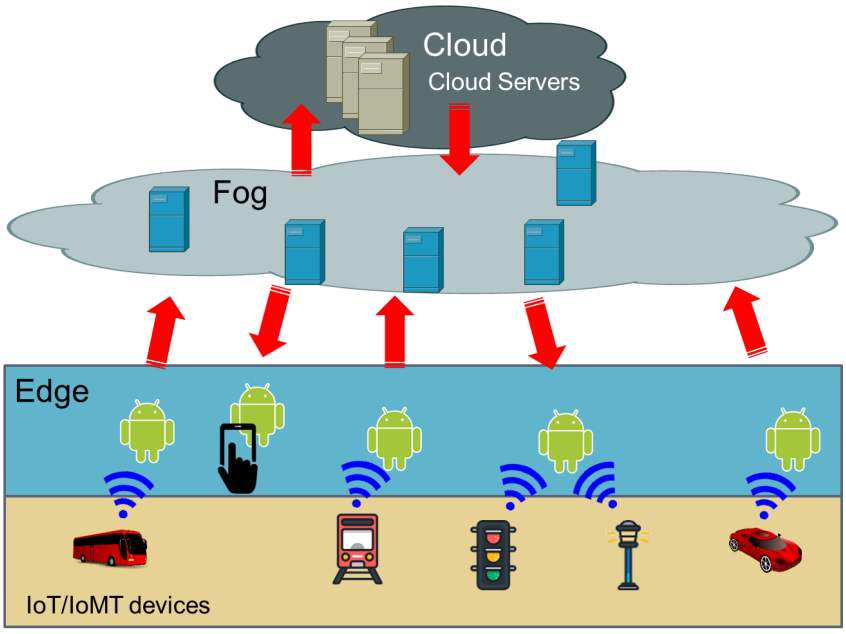


Figure 1.3: Generic three-layer architecture. In this model, smart devices are used as sensor gateway to stationary infrastructure. The smart device can have build-in sensors or connect to Bluetooth-enabled sensor devices.

To implement their applications, system developers and architects will need a highly scalable, layered environment that can be deployed at critical points in the city. So, the first problem addressed in this work can be described as:

Problem Statement I:

The middleware for a large smart city must be scalable, layered- and geo-distributed with support to mobile devices.

If, on the one hand, the three-tier architecture allows greater flexibility in processing. On the other hand, it still requires the developer to select where the processing will be carried out, which can make the application design task

even more laborious with the introduction of new processing tiers because it leaves even more options to the developer.

Using three-tier middleware to develop or deploy their application, the developers need to break up their system into tasks and associate these tasks with the middleware tiers, which leads us to the second problem addressed in this work:

Problem Statement II:

Programmers need a test environment that emulates several real environments, varying characteristics such as bandwidth, number of nodes in each layer, data loss rate, delays, disconnections, etc., so that they can know the behavior of their applications during the development process.

Vehicle traffic control in a city is an exciting example of tiered processing distribution. We would have single board computers installed on the streets and avenues at the edge tier, collecting information from magnetic sensors and calculating the average speed and the number of vehicles. Each single board computer would be responsible for part of the route and would not be aware of what is happening in other domains. Still, they would send their information to servers in the fog who could decide traffic light timing in their locations. Traffic information can be sent to the cloud for storage and later statistical analysis or sent as relevant information to end-users interested in traffic.

Still considering traffic control, in the fog layer, we can consider two forms of control. In first, we want to control the flow of vehicles on the main roads only. In this case, each main road can be associated with a server process, in the fog, which would have access to information about the vehicle flow on the main road and on the secondary roads that cross it. Through this information, the server can modify the timing of the traffic lights of the roads involved. Another way to control the flow of vehicles is to consider that the city's traffic is fully connected and that changes in the time of a traffic light can influence the entire city. We must consider the distribution of controlling traffic-lights tasks on several servers that will need to change information between them to make a global decision, which could be done through some decision algorithms like Paxos [28, 29] or Raft [30], for example.

1.2 Goals

Our goal in this research is to investigate how to best provide an environment that allows software developers and architectures to experiment

with different implementations of their applications with different patterns of communication and computing. Developers should be able to analyze the effects of distributing their application tasks to different layers. And considering each layer individually, processing can also be done on a single server, on a LAN server cluster, or even on geo-distributed servers connected via WAN. The development environment must provide mechanisms that allow for:

- Distributing the application processing across the tiers. The data-center-only programming model is very centralizing, prone to creating bottlenecks, and with a single point of failure. On the other hand, multilayer environments allow the distribution of processing according to each layer's characteristics.
- Distribution of the processing within each tier. Whenever an application processing function is determined for some tier, that function can be divided and executed in a data center or distributed geographically on several servers, increasing the degree of distribution and parallelism of the application, as exemplified in Figure 1.4. For example, in a vehicle traffic control application, if the function of controlling the timing of traffic lights is assigned to the fog, each fog server can control the traffic lights on one or more main routes.
- Checking the performance of the application. Software developers can run their applications on various computing models: cloud-only, cloud-fog, cloud-fog-edge, etc. The development environment needs to provide tools that allow a quantitative evaluation of each implementation's performance concerning computer networks, response time, data loss, processing time, etc.
- Simulating or emulating a realistic deployment environment. Deployed smart-city applications are subject to packet delays and losses on computer networks. The real components where the applications run have finite processing power and memory capacity. The development environment must support the emulation of resource limitation so that the developers can know their applications' expected performance when they are deployed. They can use this information to choose the most appropriate model of processing distribution.
- Implementing the system without significant reengineering efforts. At the end of the development and testing process, the application is ready to be deployed. A development environment that can also be used as a deployment middleware eliminates the need for major product reengineering.

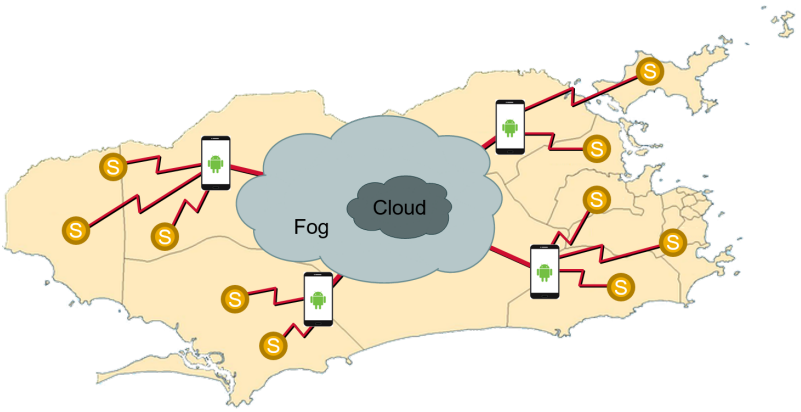


Figure 1.4: Data flow in a Smart City. The circles represent nearby sensors connected to smart devices. The Fog and the Cloud are connecting them. Computing can be performed at the data center located in the cloud, distributed in small data centers located in the fog, or through pre-processing on smart devices on the edge.

Based on these premises and the problems presented in Section 1.1, we have the following research questions:

Research Question I:

How to provide an architecture that helps investigate the best tier/layer for processing Smart-City information?

Research Question II:

At which level should IoT data be processed in a Smart City?

Research Question III:

How to provide a distributed scalable smart city infrastructure with support for mobile sensing, with a convenient roaming model?

To exemplify how Smart-City application developers are supported in their endeavor with our environment, suppose an application where image processing consumes most computational resources (CPU, memory, network). In a cloud-fog-edge environment, images are acquired at the edge and processed somewhere between the edge and the cloud. The programmer could choose to carry out the processing at the capture site, sending only consolidated data to the fog-cloud tiers as a slice of a captured image containing just the subject. A second option would be to capture the images on the edge tier, process the information in the fog, and then send the consolidated data to the cloud. Finally, a third option would be to capture the data on the edge and send it

to the cloud routed through the fog. Using our middleware, developers can experiment with the different ways of implementing the application, analyze each one's pros and cons, and implement the best option, which may even include the division of image processing between the different layers. That is, the device at the edge would do a pre-processing and send images with some degree of data consolidation, as just the part of the image that contains the subject, to the fog. At this point, a server would analyze a smaller image and could process more quickly. It is important to note that when talking about an application for Smart Cities, we are talking about hundreds or thousands of sensors sending a large amount of data and not just a single camera connected to a server.

Figure 1.5 shows an application that can be broken into six tasks or layers named **A**, **B**, **C**, **D**, **E**, and **F**. In this example, the Task or Layer **D** can be performed on both the edge tier and the fog tier. Through experiments on our middleware, developers can choose the best way to implement the application by distributing the application's tasks or layers in the middleware tiers.

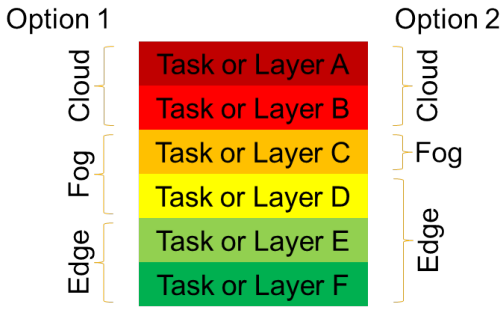


Figure 1.5: Example of two possible task or layer distribution across tiers. Task or layer **D** can be executed either in the fog or on the edge tier.

Considering that most entities in a city, like living agents or things, are always on the move (e.g.their population, cargo transportation, mobile sensors, etc.), it is imperative to provide support also for communication and processing of Smart-City data produced/consumed by mobile entities, and where the movement-specific aspects of these entities is a relevant aspect of the entire Smart-City application logic, extending the concept of IoT to IoMT (Internet of Mobile Things) [31]. Mobility support should cover data and context information concerning movement-specific aspects, such as current location, direction of movement, speed, acceleration, time spent at each place, etc. of mobile elements and ways of sending messages based on custom logic and mobility-specific groupings of mobile IoT devices.

1.3 Discussion

Over the last few years, we have been working on a three-tier middleware architecture for smart cities with support to mobile nodes called MUSANet [32, 33, 34]. MUSANet moves the traditional network edge even closer to the sensor devices, allowing for part of the processing, carried out in the fog in the two-tier architecture, to be transferred to the new network edge. In addition to minimizing latency, processing on the edge allows for data to be filtered, processed, and aggregated before being transmitted to the fog, saving network bandwidth and energy [35]. The implementation of edge computing can be done through smart sensor devices attached to low-cost small and portable Single-Board-Computers, such as Raspberry Pi, or through smartphones with low-range, low-power wireless radios (WPAN), such as Bluetooth Low Energy or ZigBee to connect to sensors devices or even using their built-in sensors.

Also, as we consider it essential to support mobile devices on the edge for IoMT [31] or MEC (Mobile Edge Computing), we decided to build MUSANet on the top of the ContextNet and InterSCity middleware, that natively provide support for mobile peripheral IoT devices/sensors/actuators. When moving around the city, the devices located on the edge of the network are capable of sending, in addition to the data from the sensors connected to them, their location, date and time, energy levels, and other relevant contextual and operational information. Services running in the Fog or Cloud may then use this information to be aware of context aforementioned, manage connections to edge devices optimizing response time or the number of connections at stationary gateways, and send unicast or multicast (groupcast) messages.

Commonly, some of the several sensors installed in the Smart City generate extensive data flows that need to be treated considering an event window or the relationship of information from other sensors. For example, in a weather station, the variation in temperature and pressure over a short time may indicate that rainfall will soon occur. Although this analysis can be done locally, through edge computing, if several stations report the same event to the servers in the fog, this may indicate that the possibility of rain is even more possible or that the phenomenon will cover a large part of the city. This example illustrates the use of Complex Event Processing (CEP) [36] on the edge (for analysis of each weather station individually) and in the fog (to handle events generated on the edge).

The three-tier MUSANet architecture can be used as a testbed that allows developers to test their application in a controlled environment equal to, or at least very similar to, the deployment and operation environment of the

Smart City. The same infra-structure architecture used for development and testing can be used in production, which allows the developer to have a single environment at all phases of the project. Our testbed allows the developer to deploy the application under development, configure delays, data loss, and bandwidth to simulate a production network, and observe the influence of these variations on their application. One can obtain these parameters through publicly available network monitoring tools such as ping, tracert, Wireshark, and telnet. In addition, MUSANet allows for developers to deploy their services on MUSANet without the need to stop or reconfigure services already implemented during the implementation of the application in the production environment. The MUSANet also provides tools for analyzing network performance and resource allocation, such as memory and CPU.

2
Related Work

To better organize the work related to this research, we have divided this chapter into three parts to contemplate the points covered in this research work. First, we present in Section 2.1 the solutions used in some Smart Cities. Then, in Section 2.2, we analyze some testbeds for Smart Cities. And finally, in Section 2.3, we selected some research papers that developed an analytical way to help developers distribute computing among several layers of multi-layer middleware for Smart Cities.

2.1
Smart Cities

There are currently several smart city initiatives, where sensors and actuators provide data and respond to stimuli to facilitate the management of day-to-day activities in cities. The Forbes website [37] cites Barcelona, New York, London, Nice, and Singapore as the five most "smart" cities. A smart city efficiently manages energy consumption (Smart Grid), vehicle traffic and parking areas, public lighting, etc., through integration between the various government agencies, companies, and the population, providing open data to all. In this Section, we present the Smart Cities of Santander and Barcelona and the University of Padova project for a Smart-City middleware. We end the Section by listing some of the most important Smart Cities in the world.

The SmartSantander [38] project consists of a testbed and an infrastructure for data collection developed in Santander city, in Spain. Several types of sensors [10] were installed around the city, creating the largest testbed in the world. As Figure 2.1 shows, the architecture of the project is divided into three parts: (1) IoT nodes, composed of several IoT device, responsible for capturing the sensor information; (2) gateways, which perform the connection between the IoT nodes and the central infrastructure, and also serve as a bridge between the IEEE 802.15.4 networks and Internet; and (3) the layer formed by the servers, which store, process, and deliver data to external applications. The gateway layer distributed its gateway throughout the city, connecting WSNs that do not implement TCP/IP to the infrastructure. The project allows collecting information from the smart city, both providing services to the

population and the government, and also it can be used as a testbed with real data. However, its gateway layer does not implement a load balance policy to distribute data traffic, application developers cannot choose the layer where they will process the data, and there are no tools for gathering information about the performance of their applications.

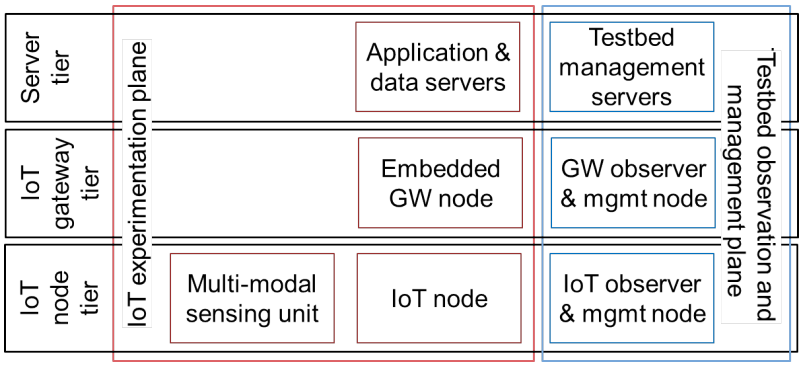


Figure 2.1: SmartSantander 3-layer architecture.

Barcelona City [39, 40], in Spain, is an example of success in the implementation of a smart system, known as BCI – Barcelona Ciudad Inteligente (Barcelona, Smart City). They distributed different types of sensors and actuators throughout the city to collect a variety of data [41]. Data is sent over the WAN to the platform via Sentilo [42] module, an open-source application using the municipality's infrastructure. Although there are several access points available to connect the sensor elements to the central infrastructure, there is no control over the number of elements at each gateway, nor a policy to optimize connections. The data collected by Sentilo is filtered, processed, and stored in an intermediate layer that makes it available for applications that provide services to the city. The Barcelona system performs information processing in the central and application layers. BCI's infrastructure has an API so that trusted third-party companies can use the information available in the city council's database. Other companies can access the database through the standards proposed by the OGC [43]. Part of the BCI infrastructure was used as a testbed of real data for the development of new applications, but, unlike the case of Santander, only trusted third-party companies could participate in the tests.

University of Padova, Italy, together with the city hall of Padova city and Patavina Technologies, developed a smart city project [44] able to monitor public lighting and atmospheric conditions such as temperature and humidity. In their architecture, the sensor nodes installed on the poles create a WSN based on the IEEE 802.15.4, 6LoPAN and IPv6 RPL protocols. Gateways, installed around the city, work with a dual-stack protocol to convert the sensor

network protocols to Ethernet, IPv4, allowing the endpoints near sensors to send data to the database through the standard network infrastructure. Load balancing at gateways is done by implementing different sets of protocols in each gateway, so a sensor can only connect to a gateway that implements its protocol set.

Other examples of Smart Cities in the world are Sydney, Melbourne, Brisbane, Canberra, and Adelaide in Australia [45], New York, Cedar Rapids, Austin, Columbus, and LaGrange in the USA [46], London, Amsterdam [47], and Stockholm in Europe [48], Jakarta, Hanoi, Singapore [49], and Kuala Lumpur [50] in Asia, and Santiago, Buenos Aires, and Montevideo in South America.

2.2 Testbeds

Several authors have developed testbeds for experimentation in conditions close to real ones in a single layer. Testbeds developed for the Edge layer typically offer access to physical components (Edge Nodes), support for multiple protocols for IoT, and tools for quantitative measurements, in addition to hiding low-level implementation details [51, 52, 53, 54, 55, 55, 56]. However, few predict interaction with humans. Focused on the Fog layer, there are also a lot of testbeds [57, 58, 59]. Most of these testbeds offer access to different types of networks, a controllable environment to emulate an actual environment, simulation, and access to actual components. Some testbeds also support mobility. In addition to physical testbeds, there are also several mathematical simulators for Edge [60, 61, 62] and Fog [63, 64, 65] layers.

There are several testbed implementations and projects for Smart Cities. One of the most notable cases is SmartSantander [38], briefly described in Section 2.1. It was implemented as a testbed in a Smart City and made available for free public access allowing developers to collect real data instead of relying on simulated data as most testbeds do. Through this testbed, the developer can perform experiments using real data from sensors distributed across a city. The time slot reservation system guarantees the developer exclusivity in capturing and processing data during the experiment period.

Corradi et al. [66] and Cardone et al. [67] describe in their work the middleware ParticipAct, a crowdsensing platform based on smartphones. The work was developed at the University of Bologna, Italy, and had the collaboration of 300 students for one year. This two-tier middleware uses the participants' smartphones as the edge of the network to process, store, and manage data collected from tasks passed on to volunteer users. When the

users end their tasks, their smartphones send the results to the stationary core layer. The core layer is responsible for sending tasks to users, preparing the received data for storage, and aggregating the information in time windows to be consulted. Unlike the testbed developed in Santander, this testbed was used in experiments within the university to analyze the behavior of the population in collaborating with a participatory environment. Students were encouraged to participate in challenges and received virtual prizes in return.

The main objective of the testbed described by Svítek et al. [68] at Prague is to test Smart-Cities applications over the existing GOLEMIO [69] service infrastructure. Golemio is a storage platform for large amounts of data implemented within the scope of the SmartPrague project. The aim is to allow access to open data from the city of Prague so that smart-city system architects can develop and test their applications using real data. However, as in SmartSantander, information about application performance such as the bandwidth consumption, the number of packets lost, or the effect of different bandwidths on the application is not directly supported.

Other testbed examples are OrganiCity, Select4Cities-based middleware, and Embers. OrganiCity [70] is a testbed that includes data set from some Smart Cities, as Aarhus in Denmark, London in England, and Santander in Spain. This data is available for immediate use by software developers. Select4Cities [71] is a challenge for developers to create middleware for Smart Cities based on competition to solve a specific proposed problem. Embers [72] is a testbed for Smart Cities aiming to provide a platform-independent test environment, so the software developers do not become dependent on a specific vendor.

All presented testbeds fulfill their role in a specific area. Some have been implemented in a real environment, can be accessed by the community, but do not support a new deployment in a large city. Others are test tools, not being prepared for deployment in a real environment. To the best of our knowledge, no testbed has been developed to be used from design to deployment in a real environment, offering debugging tools and application performance analysis.

2.3 Multi-tier system evaluation

Our approach in this work is to create a tool that, in the design and development phases of an application for Smart Cities, can help developers to better understand the behavior of their application. In this section, we reference some work that proposes a theoretical analysis of the behavior of applications.

Several authors [73, 74, 75, 76] have already proposed algorithms based on the mathematical analysis of two-layer middleware architectures (cloud

and fog) for Smart Cities. Articles that are not focused on Smart Cities, such as Mahmoud et al. [77] for health-care, were not considered here, although many had good proposals for algorithms and quantitative analysis. That said, considering only Smart-Cities initiatives, Naas et al. [78] proposed a resource allocation algorithm called iFogStor whose premise is to reduce latency by distributing nodes in the fog in order to allow network connections with fewer hops. In their work, Deng et al. [79] and Cheng et al. [80] used respectively the delay in the network and energy expenditure as parameters to allocate resources. Most of the works that present mathematical models for allocating data processing study the distribution between the cloud and the fog, which is considered the edge of the network.

In this work, we consider the edge of the network to be the mobile devices that collect data from the sensors and have processing capacity, even if reduced, and we investigate the allocation between the fog and the new network edge.

Mobile Edge Computing (MEC) is still in its infancy, and few studies analyze how computing in three-tier architecture should be distributed. HetMEC [81], proposed by Wang et al., is a three-tier architecture similar to MUSANet, where sensor elements connected to smart devices connect via wireless networks to stationary infrastructure. Wang et al., in their article, analyze the architecture mathematically and prove the results through simulations. Vasconcelos et al. [82] mathematically analyze the allocation of resources in a distributed network in Cloud, Fog, and Mist considering cost, bandwidth, and latency. Their results were validated through simulations of the architecture.

Canali et al. [83] analyzed how to distribute the computation among several Fog nodes to minimize communication latency and processing time. The article presents a mathematical model considering that servers in the Fog receive different data types from the city's sensors. This model is used to associate servers and sensors. In Canali's work, the sensors connect to the servers, and these connections generate the assignment of services. Yousefpour et al. [84] proposed a heuristic model for allocating servers according to their workload. In their model, if a server is overloaded, it can send the job to another server or the cloud. An adaptation of these models would allow us to analyze the assignment from the point of view of services, using the result to suggest the best connection point for the sensor elements.

2.4 Discussion

In this Chapter, we presented some Smart Cities implementations. We also presented some platforms developed for smart cities without trying to exhaust the subject. There are a number of surveys on smart city and IoT platforms [85, 86]. Here we focus on multi-tiered approaches.

Several mathematical models, in the line of Vasconcelos' work, have already been proposed to help the system architects for Smart Cities allocate tasks and other resources in middleware layers. Many of these models have been validated through simulations. However precise they may be, mathematical models and simulations fail to reflect a smart city's real environment fully. At this point, emulators and real environments allow system architects to analyze the behaviors of their applications in a real environment since a testbed is a small instance of the real world.

Corradi and Cardone's middleware ia an example of how the population can collaborate with the data collection of a Smart City being encouraged by rewards. Administrators and third-party companies can use this system to obtain the collaboration of volunteers. They can spontaneously use their smartphones as hubs to collect data in regions where sensors do not have a continuous connection to the Internet, that is, using opportunistic connections [87, 88, 89].

All the studied cases present middleware that fulfills the role for which they were designed. But no platform provides a set of tools for performance analysis and environment emulation of an actual Smart City. In short, they do not offer tools for the system architect to compare different implementations of the same application, varying the distribution of tasks in the middleware layers.

To the best of our knowledge, no middleware allows designers, developers, quality assurance teams, and administrators to investigate the more appropriate tier or layer to implement data processing.

3
MUSANet

We start this chapter providing an overview of the MUSANet architecture and the main technologies used in its implementation. Next, we discuss how we integrated all of these technologies int the edge-fog-cloud model of MUSANet. Then we present some middleware that can be used to build the MUSANet tiers. Finally, we depict how we implemented the MUSANet testbed.

The MUSANet architecture is divided into three independent tiers. The top tier provides distributed scalable storage capacity in the cloud. Data can be stored in a structured way and retrieved through high-level queries. The middle tier provides Complex Event Processing[1] (CEP) [36], responds in near real-time, and sends data to be stored in the top tier. This tier, implemented in the fog, includes distributed gateways between the data-collecting devices and the stationary infrastructure. The bottom tier connects the stationary infrastructure to smart objects, forming a Bluetooth network. This tier uses a dual-stack network protocol to retransmit collected information to the middle tier via IP over 3G/4G/5G or Wi-Fi. This tier, located close to sensor devices, represents the new network edge Many studies [90, 91, 92] report the importance of moving cloud computing to the edges of the network, aiming mainly to decrease traffic on the net and minimize latency. Gupta et al. report the advantages of moving computing to the mobile edge of the network, beyond the fog, as implemented in some commercial platforms [93, 94] and academic work [95].

All three tiers offer APIs so that external applications can interface with MUSANet. An example is an application to control the internal temperature of a bus. This application only requires the bottom tier to receive the inside and outside temperatures and control the air conditioner's servomechanisms. An application to inform bus drivers of their locations related to the next and previous bus from the same line and advise them to slow down or to speed up uses the middle tier to access the next and previous bus locations. A bus data collection system for historical analysis uses the top tier to store and retrieve information.

[1]The current InterSCity version now supports CEP.

3.1 Cloud – InterSCity

To implement MUSANet's top tier, we chose to use InterSCity [96], an open-source platform based on microservices, to model the resources of a Smart City. The platform is being developed collaboratively by ten Brazilian educational and research institutions (USP, FGV, IFMA, PUC-Rio, UFABC, UFMA, UFMS, UFRJ, UNICAMP, UNIFESP). In this Section, we present an overview of this platform as originally proposed.

Figure 3.1 presents the InterSCity architecture. Its microservice-based architecture allows for replicating only those overloaded services, leading to greater efficiency in terms of elasticity [97]. IoT gateways connect this infrastructure to resources and can register new features, send sensor data or sign up to receive actuator commands. A resource is an element of the Smart City that can produce data or be an actuator. A resource can be, for example, a bus equipped with several sensors, where each sensor can produce one or more data that will be collected and sent to InterSCity. The Resource Catalog module stores information about the city's static features, while the Data Collector stores sensor data, allowing external programs to access either real-time or historical data. External applications send actuation commands to the Actuator Controller module which sends the command to actuators subscribed to the platform. The Resource Discovery module facilitates searching for information. It allows for data to be selected by location, values ranges, etc. The Resource Viewer module acts as a front-end service, allowing for the creation of external applications that visually explore the city's resources.

InterSCity was initially designed to receive information directly from resources connected to sensors through an IoT gateway, which is not part of its implementation. If the sensor device/resource cannot send data via HTTP, it is up to the IoT gateway to convert the sensor-device protocol to HTTP. InterSCity expects that the resources sent sensor data with some context information, delegating to the lower tiers the task of including, for example, the geographic location of the sensor or resource and converting the original sensor protocol, usually a lightweight protocol, to HTTP. Based on the sensor-provided geographic coordinates, InterSCity adds other context information, such as neighborhood, city, state, country, etc. We present our solution for the IoT Gateway in Section 3.2.

For example, if we create a bus as an InterSCity resource at latitude -23.559616 and longitude -46.731386, as shown in Listing 3.1, the response will be similar to that shown in Listing 3.2.

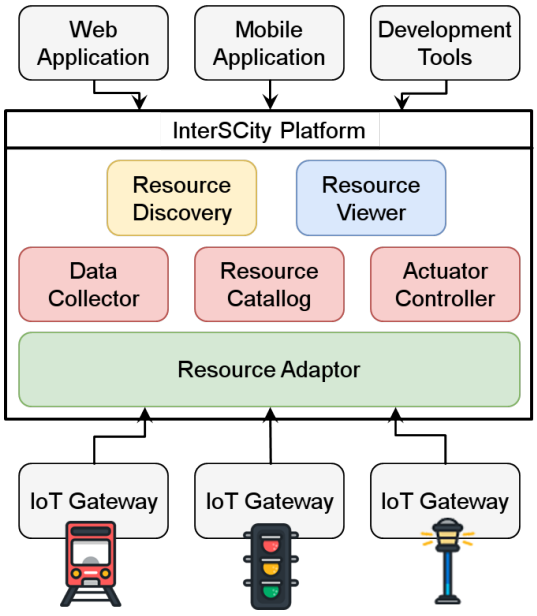


Figure 3.1: InterSCity Platform Architecture [96]. The Figure shows the platform with one instance of each one of its services, as well as how external applications (at top) and IoT resources (at bottom) communicate with it.

Listing 3.1: Creating a bus as an InterSCity resource.

```
{
  "data": {
    "description": "A public bus",
    "capabilities": [
      "bus_monitoring"
    ],
    "status": "active",
    "lat": -23.559616,
    "lon": -46.731386
  }
}
```

The platform stores the data hierarchically allowing for high-level queries. Smart objects (smart sensors and smart actuators) in the city environment are modeled as a multilevel hierarchical data structure, allowing for an smart object to have several different types of data associated with it. For example, public bus objects can be described as having various “capabilities”, such as ambient conditions (for instance, temperature and humidity), number of passengers, location and characteristics). The “positioning capability” may contain speed and location information, which in turn has latitude and longitude information. The capability features may be line number, serial number, and so on.

Listing 3.2: Positive response from InterSCity when creating a bus as a resource.

```
{
  "data": {
    "uuid": "45b7d363-86fd-4f81-8681-663140b318d4",
    "description": "A public bus",
    "capabilities": [
      "ordem",
      "linha",
      "velocidade"
    ],
    "status": "active",
    "lat": -23.559616,
    "lon": -46.731386,
    "country": "Brazil",
    "state": "Sao Paulo",
    "city": "Sao Paulo",
    "neighborhood": "Butantan",
    "postal_code": null,
    "created_at": "2017-12-27T13:25:07.176Z",
    "updated_at": "2017-12-27T13:25:07.176Z",
    "id": 10
  }
}
```

Although other IoT platforms for Smart Cities [98, 99] could be used to store data in the MUSANet cloud, we chose to use the InterSCity platform because it can model complex physical objects like buses with GPS and sensors, maintenance holes, service vehicles, and also virtual objects like events, notifications, commands and so on. It also supports mobile objects, allowing for the location of smart objects to be reconfigured when they move. InterSCity can be easily integrated with the ContextNet (see Section 3.2) via HTTP. Finally, MUSANet and InterSCity are multi-institutional research project[2].

3.2 Fog – ContextNet

For MUSANet's middletier, we chose ContextNet [100], a public-domain middleware developed to allow for mobile devices to communicate with stationary servers efficiently. The ContextNet is being developed at the Laboratory for Advanced Collaboration (LAC) [101] at PUC-Rio.

Figure 3.2 depicts the ContextNet architecture. The main ContextNet modules are the *Controller*, implemented as *Processing Nodes*; the *GroupDefiner*, responsible for grouping the mobile nodes; the *PoA-Manager*, which controls

[2]https://interscity.org/

the access to gateways; and the *Gateway*, a dual-stack protocol converter and group/mobile-nodes mapper. In the following, we describe these modules that constitute the core of ContextNet.

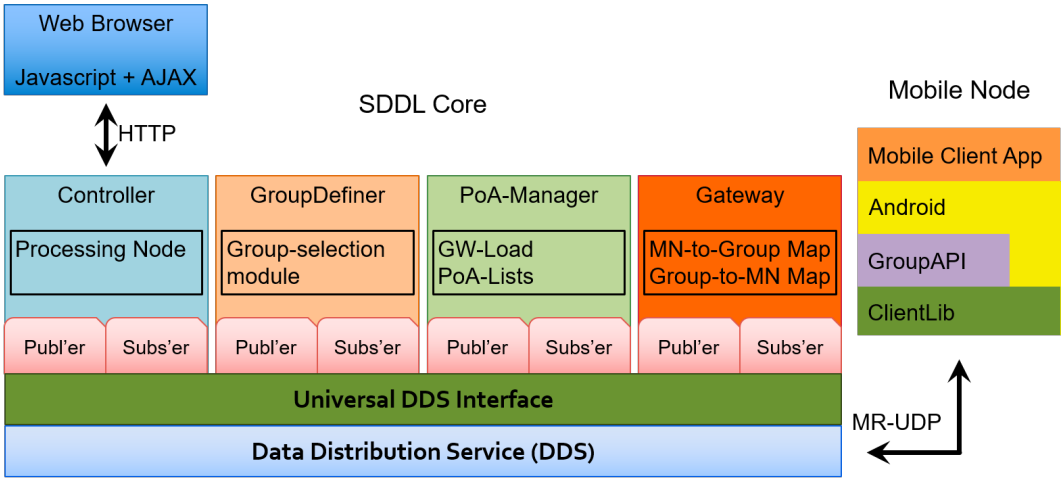


Figure 3.2: ContextNet Architecture [102]. The core components (Processing Node, Group Definer, PoA-manager and Gateway) are connected by the DDS.

ContextNet allows for its gateways to be deployed throughout the city for load-balancing the connections. The PoA-Manager is a module that uses a custom policy to suggest the best gateways for mobile nodes to connect to ContextNet. Whenever a mobile device moves to a more favorable gateway, the PoA-Manager warns the device to disconnect from its previous gateway and connect to a gateway with a more favorable network connection.

The *GroupDefiner* module can group the different sensor devices connected to ContextNet according to a policy defined by the application. This grouping allows for unicast (individual) or groupcast (for a group) messages to be sent to the sensor devices. Examples of grouping policies can be the location of the sensor device, its characteristics, its energy level, and so on. If we consider, for instance, rescuers in a disaster environment, we could group them by their location, where messages to rescuers could alert them to the need to go to a specific area. Another grouping of the same rescuers could consider their hierarchical or functional levels, where messages could be sent only to the coordinators. GroupDefiner allows intersections between the different groups defined by the application, making the sending of messages accurate and straightforward.

As the Processing Node is a Java application, its implementation can be done through virtual machines, one for each application, through containers, or by a host shared by several applications.

The ContextNet interconnects its components through SDDL [103], a distribution messaging system implemented using the OMG DDS standard [104].

For bidirectional communication with mobile devices, ContextNet uses MR-UDP [102], a reliable UDP-based protocol with mobility support.

ContextNet's focus on mobile devices was the main reason for our choice of it to implement the fog tier. We were also interested in ContextNet's support for distributed processing, load balance of connections, and groupcast message.

3.3 Edge – Mobile-Hub

To implement the MUSANet's bottom layer, we chose the Mobile Hub [31] Android app. Mobile-Hub is a public-domain[3] application framework for Android devices that provides Internet connectivity to mobile devices using Bluetooth technology. The Mobile Hub was developed to be used as a context-aware hub for mobile or stationary Bluetooth sensors. The Mobile Hub natively implements the ContextNet ClientLib library, making it a convenient interface for collecting and sending data to ContextNet. The ClientLib library, in addition to containing the objects necessary to communicate with ContextNet, also implements the MR-UDP protocol. This protocol is based on UDP with reliability and aimed at mobile devices.

The Mobile Hub allows Bluetooth-enabled sensor devices to connect to Android devices, typically smartphones, acting as a bridge between IoT/IoMT devices and the stationary infrastructure. Once connected, the sensor devices send data to the smartphone, which can process it in a conventional way or through CEP [36]. On the Android device, the data can be analyzed, summarized, grouped, and enriched with context. Resulting data can then be sent to ContextNet for further processing or storage in the cloud or generate responses for the actuator devices.

Figure 3.3 presents a block diagram of Mobile-Hub. The Mobile-Hub S2PA Service discovers and connects to mobile sensors and actuators using Bluetooth. Once connected, sensors can send their data to the Mobile-Hub. Data generated by connected sensors feed the M-EPA Service, which runs CEP rules to generate high-level events (determining, for instance, whether the temperature of a bus is too cold or too hot). The original data or high-level CEP events, enriched with energy level, location and timestamp, can be sent to ContextNet, to be handled by a Processing Node or stored in InterSCity. It is important to notice that location information is related to the Mobile-Hub device, which may differ from the sensor location itself because not all sensors provide location information. This difference is limited because the sensor and smart device are connected via Bluetooth, which has a maximum range in the

[3]Available at http://www.lac.inf.puc-rio.br/dokuwiki/doku.php?id=m_hub

order of a few tens of meters. These same data and events can be sent, as messages, to local mobile Android applications running on the same device as the Mobile Hub service.

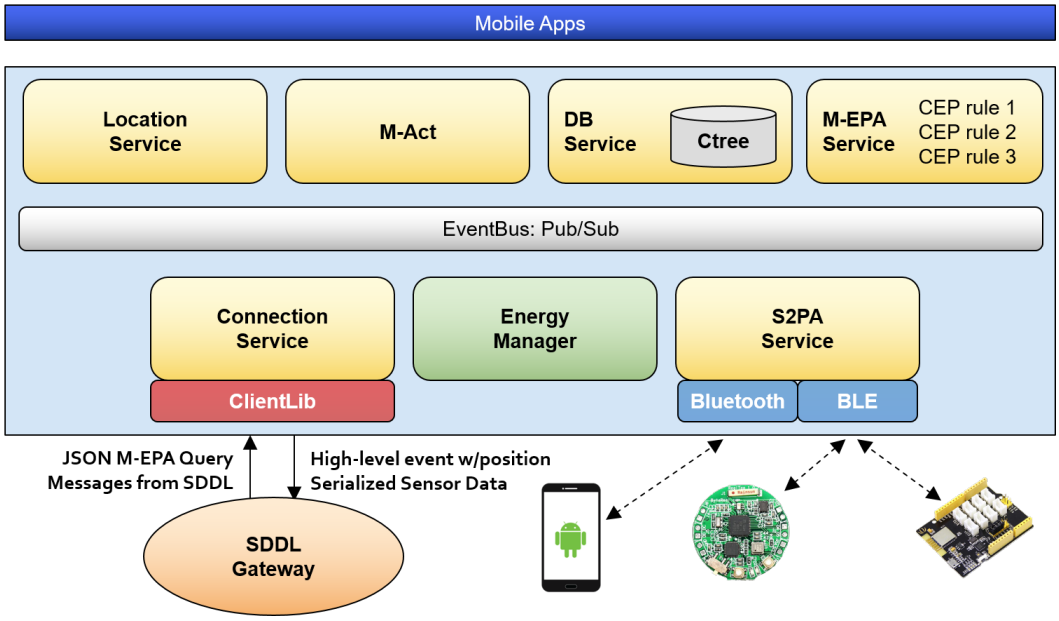


Figure 3.3: Mobile-Hub Architecture [31].

Applications located at the core of ContextNet can send messages to the Mobile Hub, either to deploy CEP Rules in the M-EPA Service or to trigger Bluetooth actuators connected to the Mobile-Hub. An example of an actuation command sent from the middle tier to a Mobile-Hub at the edge could be to increase the ambient temperature of the bus if rain has been detected in the way ahead of the bus.

Although Mobile-Hub has the capability to process information from multiple sensors connected to the Android device, including Complex Event Processing, it has limited storage capacity and, due to the Bluetooth technology used for connection to the sensors, it only has access to data from nearby sensors. The Android device running Mobile-Hub needs to be connected to an upper layer with higher processing capability and able to store a considerable amount of information. Other devices running Mobile-Hub can also be connected to this layer for data processing in a more global way. Because this upper tier is typically part of the stationary infrastructure, processing capability and storage space are not issues.

3.4 MUSANet Architecture

In Smart City, sensors produce a myriad of information that must be stored for historical analysis, but some applications need to consume this

information in almost real-time. Traffic light synchronization and accident detection and route deviation due to the accident are examples of this type of applications. Other applications need to analyze historical data to generate reports or for city planners. Typically climatic or seasonal events need to be stored and compared by these applications. Traffic information falls into these two categories: it needs to be processed in real time so that traffic lights timing is properly configured, but it also needs to be stored so that city hall can make long-term decisions such as those regarding the development of the city's road network.

Although architectures such as InterSCity are intended to be extensible and scalable, elements inherent to communication, such as the network between the sensor and the platform, create bottlenecks and delays in the system. Besides, the vast majority of sensor elements have little communication capacity, using short-range protocols, such as Bluetooth or NFC. To deal with geographic scale, we propose a scalable distributed multi-tier architecture that reduces response time by implementing a system with edge processing capability where BLE-enabled Android devices running a Mobile-Hub application collect data from nearby sensors and process, filter, and consolidate information before sending it to an upper layer. Mobile-Hubs send data to one of many ContextNet Point of Attachments (PoA); the choice of a suitable Point of Attachment minimizes the delay between capture and delivery of information. The PoA works as a gateway to the ContextNet stationary infrastructure in the fog. There will be a Processing Node in the local network of this gateway, allowing information to be processed without more communication delays. The ContextNet Processing Node is responsible for contextualizing the data and forwarding it to the InterSCity platform. Mobile-Hub actuators registered in ContextNet receive commands relative to specific regions, avoiding floods of unnecessary commands to unrelated actuators.

MUSANet has three tiers reflecting different levels of abstraction, as shown in Figure 3.4. InterSCity implements the top tier, forming a structured storage system, supporting high-level queries, and modeling the Smart-City elements. The middle tier, implemented by ContextNet middleware, allows for the allocation of sensors and actuators into groups or regions, and provides distributed attachment points to the infrastructure. The bottom tier consists of smart mobile devices running Mobile-Hub to connect Bluetooth-enabled sensors and actuators to the MUSANet infrastructure through Wi-Fi or 3G/4G/5G networks.

The top tier of MUSANet is an system that provides structured information processing and the storage module. This module also provides a REST

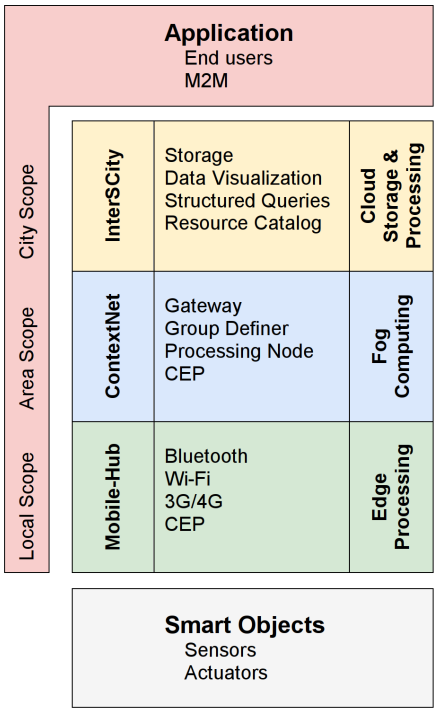


Figure 3.4: MUSANet three-tier Architecture. The main API for exchanging information between applications and MUSANet is through Smart-City resources mapped in InterSCity, at the Cloud. Still, the applications also have interfaces in the Fog and Edge tiers.

API for developing external applications for end users or other systems that rely on the smart city data managed by the InterSCity framework, which can be accessed using HTTP PUT, DELETE, POST and GET requests.

As an example, the Unix command in Listing 3.3 retrieves information from all buses within a radius of 50 meters form a geographical location determined by latitude and longitude.

Listing 3.3: Unix command to retrieve buses located within a radius of 50 meters from latitude 22.938625 and longitude -43.192611.

```
1 $ curl --globoff  -L "interscity.example.com:8000/
    discovery/resources?lat=-22.938625&lon=-43.192611&
    radius=50".
```

The middle tier receives data from mobile or stationary edge nodes connected to sensors. This tier is also responsible for defining the groups (regions) to which the mobile nodes belong based on their geographic location. Also at this level, patterns of events generated by the edge tier can be analyzed and emergent complex events may be assembled, such as a drop of luminosity on a street section for more than 60 seconds. Those events are analyzed by a CEP engine allowing for rapid response in almost real-time. This tier is also

responsible for filtering and consolidating data that must be sent for storage in InterSCity and for sending commands to actuators. Because this tier receives data from several mobile devices, it has the ability to make decisions regarding a large geographical area, such as increasing by 10% the green light period at all traffic lights for vehicles traveling on Main street of the financial district. Warnings about bus delays, traffic jams, weather conditions, and detection of possible catastrophes such as floods are examples of decisions that could be mapped to this tier. Another example of decision making that could be processed at this tier would be determining the best route for a garbage-truck based on weight or volume sensors located in culverts and trash cans. In hazardous locations in violent neighborhoods, sound sensors coupled to mobile smart devices running the Mobile-Hub may use CEP rules to detect gunfire and send messages to the middle tier. The ContextNet Processing Node can send an alert with the approximate location and the intensity of gunfire to the police station or even to police officers in the area who are using a smartphone with Mobile-Hub. A quick response to this type of event can use the sound, location, and intensity information to notify the closest policemen or police vehicles about the gunfire. The policemen or the police vehicles can be selected and reached by their location sent by a Mobile-Hub application running on their smartphones.

At the bottom tier are mobile and stationary devices that may use their embedded sensors and actuators (e.g. in smartphones) for direct access by smart city applications, or else, function as bridges between nearby sensor or actuator smart devices with a Bluetooth interface and the middle tier infrastructure. This tier can also generate complex events based on data from local sensors, that is, sensors connected to the device. Resulting events can be sent to the fog tier.

A MUSANet configuration defines a number of distributed ContextNet sites called *slices* that are connected through the Internet to create a single ContextNet infrastructure. A slice is a set of micro-services consisting of a Gateway – and possibly Processing Nodes – a GroupDefiner, and a PoA-Manager, all of them connected by a local network. These elements communicate internally within the slice through the SDDL protocol using multicast messages. Communication between elements of different slices occurs through the Internet, with unicast messages. The implementation of these elements can be done on a single or on multiple machines. Figure 3.5 shows the block diagram of our architecture with four ContextNet slices with Gateways, InterSCity services, Group Definers, and Processing Nodes. The MUSANet infrastructure needs at

least one PoA-Manager (not shown in Figure) to send the nearest MUSANet Gateway address to the mobile nodes.

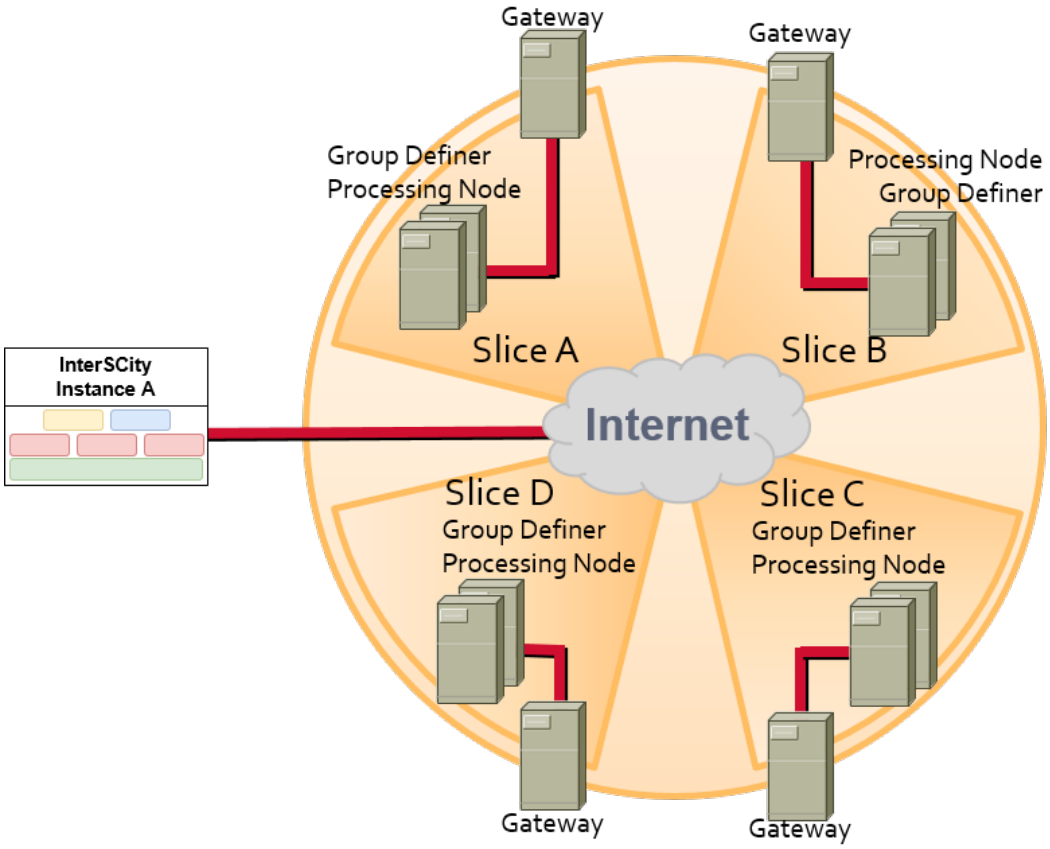


Figure 3.5: This Figure shows one InterSCity and four ContextNet slices connected over the Internet.

The mobile devices that are part of Mobile Edge Computing (MEC) connect via the radio antennas of the available wireless network infrastructure, composed of 3G/4G/5G, Wi-Fi, and WiMax, among others, to the stationary Processing Nodes of ContextNet slices. The Processing Nodes communicate with the other Processing Nodes and with InterSCity through the Smart-City wired network infrastructure (the Internet), as shown in Figure 3.6.

In our architecture, the smart city is divided into groups or regions based on sensor distribution and not just neighborhood or zones. These regions can have intersections, and there is also the possibility that regions encompass entirely more than one region, as exemplified by Figure 3.7, where region **B**, **C** and **D** have intersections with two and three regions while region **E** is completely disjoint.

For example, to implement the regions in MUSANet for public transportation in a city such as Rio de Janeiro, an administrator could use neighborhoods as a basis, where clusters of adjacent small neighborhoods interconnected by a large set of bus lines could compose a single region, and larger neighborhoods

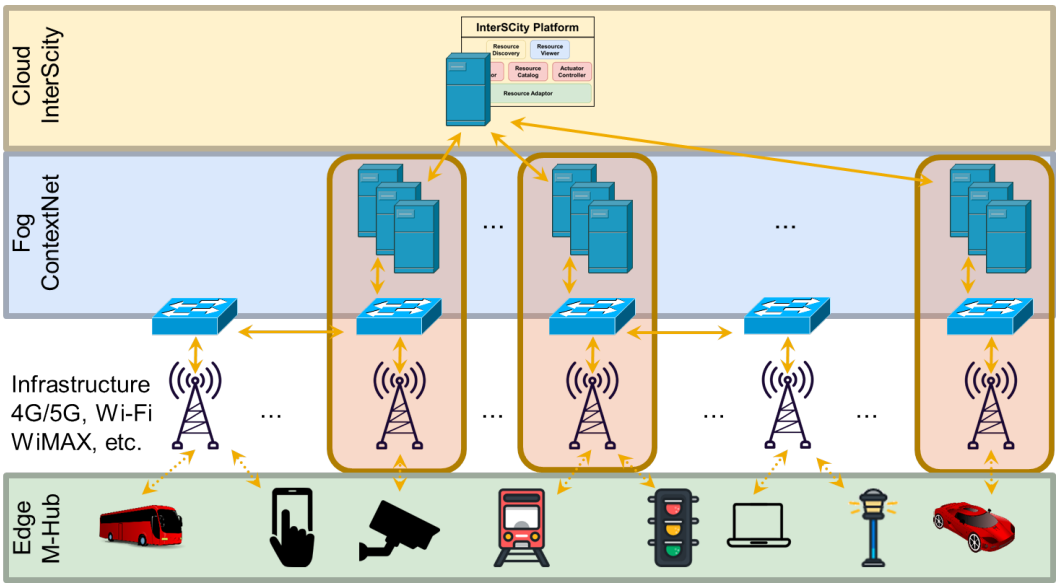


Figure 3.6: MUSANet connection diagram.

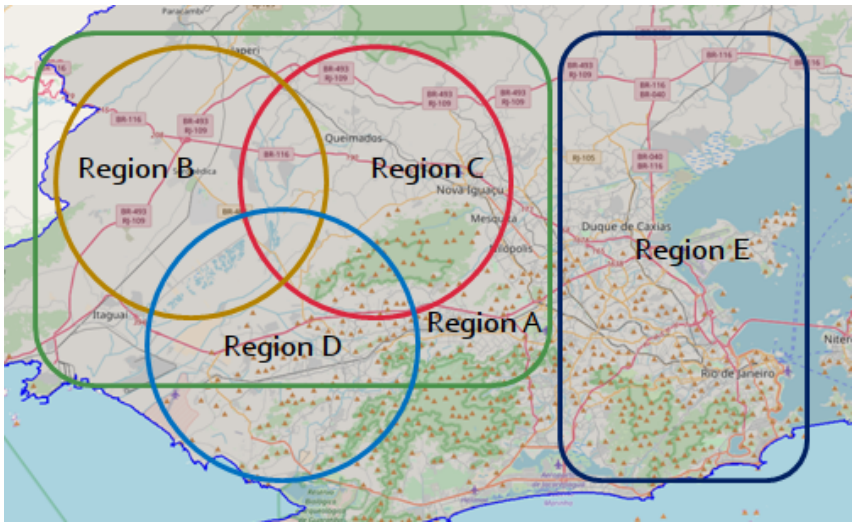


Figure 3.7: Example of regions in a city. Map from OpenStreetMap[4].

could be divided into more than one region. Each bus stop could be assigned to a micro-region completely within the region defined by the neighborhood, as previously described. Bus stops close to each other along the same route could belong to the same region.

Using ContextNet to add the region id to the data context allows us to distribute better the edge device connections to fog's gateways to load balance the data flow across multiple gateways, increasing network throughput while decreasing latency. Mobile objects should send their location to a GroupDefiner to find out which groups they belong to according to the application's policy. From the group information, the PoA-Manager can inform the mobile object of its nearest ContextNet Gateway. Future messages will be sent to the Processing

Node of this group through this gateway until the object moves to the region of a new group with a new gateway. If the object is at the intersection of two or more groups, the object must send the information to the Processing Nodes of all groups where it is located, using their respective gateways.

The nearest ContextNet Gateway is determined by the time it takes the packets to go from Mobile-Hub to the ContextNet Gateways, ie, by RTT (Round Trip Time). This approach allows for the implementation of MUSANet to be done independently of the technology used in the network physical layer between the edge at bottom tier and the fog at middle tier (between Mobile-Hub devices and ContextNet Gateways). But if we consider that there is Wi-Fi network coverage and/or 5G mobile network in the entire city, we can have, at each Wi-Fi, 4G or 5G Base Station, a small virtualized cloud [105] or a cloudlet containing a slice of ContextNet with, at least, a Gateway. The implementation of this cloudlet can be done by the city hall using the infrastructure of Wi-Fi networks or through services offered by telecommunications companies in their 5G infrastructure, as described in section 2.1 of [105]. In this case, the Gateway closest to the mobile device would always be the one that is physically located on the antenna, either Wi-Fi or 5G, where the mobile device is connected.

Each Processing Node is programmed to process only certain types of data. Data sent by sensors that are not to be processed by the node can be discarded because another node responsible for processing these data will also receive them. When data arrives at a Processing Node, the CEP engine scans the data in real time and triggers actions, when applicable. Depending on the data type, a Processing Node can send it to be processed and stored in the InterSCity service. The data are sent from Processing Nodes to InterSCity via HTTP. By default, elements within the ContextNet network use DDS protocol, but to avoid unnecessary modifications to the platform, InterSCity does not need to register within ContextNet and is accessed through its native protocol. Sensors and actuators that do not register with ContextNet, regardless of the reason, can directly contact any instance of InterSCity via HTTP.

3.5 Alternative Middleware

In this Section, we present some other technologies that could be used to implement MUSANet. The MUSANet architecture is not dependent on specific technologies, and we could use a variety of technologies at each level of our system.

An example is the Dimmer [97] middleware, which is a microservice-based platform that enables device and capability discovery. It provides an API for

access to near real-time and historical data that can be used to manage the database. Its Service Platform can process data from sensors at the cloud like the ContextNet Processing Node does in the fog. The Dimmer gateways can gather data from a heterogeneous set of sensors, but its gateways can neither preprocess data nor be installed close to sensors. Therefore, if we used Dimmer as the MUSANet top tier, ContextNet, and Mobile-Hub as the middle and bottom tiers, we will would processing capability to the top tier.

Another example of middleware for Smarty Cities is the FIWARE [106, 98] platform, an OpenStack and Docker-based environment that stores data hierarchically in a MongoDB database, supporting both sensors and actuators. The platform also provides support for authenticating users and setting user roles with configurable permissions. Like InterSCity, the platform natively supports complex event analysis through CEP for near real-time event handling. Unlike ContextNet, FIWARE does not provide direct support for load balancing on its gateways, and the smart objects must insert all context information before sending data to it. FIWARE was designed to be integrated into a myriad of platforms "Powered by FIWARE"[5] from seaport [107] to agriculture [108] management applications, so, if we use it as the MUSANet upper layer, we will obtain a scalable and elastic system able to process data streams at the cloud thanks to the FIWARE CEP engine.

At the edge level, like the Mobile-Hub in functionality, MOSDEN [109] can collect data from sensor devices that have low power radios or do not use protocols compatible with Internet routing. MOSDEN is compatible with smartphones and Raspberry Pi, which guarantees a wide range of performance. However, MOSDEN was developed to connect directly to stationary servers located in the cloud, not supporting a dynamic or automatic change of gateway for load sharing or minimizing communication delays.

The Smart Streets IoT Hub described by Blackstock et al. [110] and the gateway presented by Aloi et al. [111] could be alternatives to Mobile-Hub. The first one was developed to interface with the Hyper/Cat platform [112]. The latter is a smartphone gateway with support for Wi-Fi, Bluetooth, LTE (H+, 3G/4G/5G, or else) networks and even ZigBee via a Micro-SD card from Spectec[6]. All of their features are implemented by Mobile-Hub, which was built to work with PoA-Manager to select the best gateway, even while roaming.

Amazon[7] offers a set of ready-to-use solutions for IoT. Programmers can customize their application using AWS Lambda, Amazon EC2, and others, for their individual needs. AWS Lambda, a serverless event-driven platform at the

[5]As stated on the FIWARE website.
[6]http://www.spectec.com.tw/sdz-530.html
[7]https://aws.amazon.com/

cloud, can process complex events. Although we can easily integrate different solutions to create a full IoT custom application scalable and elastic, there may be a network delay at about 130 ms between the data source and the platform at the Cloud because not all locations have an AWS datacenter nearby (in terms of network hops). Even if we implement a solution using ContextNet and Mobile-Hub in the fog and edge levels, respectively, these delays will still be present.

Distributed systems based on cloud computing like Amazon Web Service (AWS) [113] are highly scalable, able to support billions of IoT devices and trillions of messages per seconds. This kind of system can be scaled to perform thousands of concurrent queries over petabytes of data [114]. While very powerful, AWS does not provide stream-processing capability natively at the edge using CEP rules.

IBM Watson IoT Platform [115], Microsoft Azure [93], Amazon AWS, and Google Cloud IoT Platform provide cloud-based IoT platforms capable of meeting smart city data processing and storage needs. Several components like storage and database, analytics and artificial intelligence, Complex Event Processors, and IoT hub can be linked together for near real-time analysis of data streams.

IBM and Microsoft solutions require a third-party IoT gateway for sensors that cannot interact directly with the Internet and also they do not provide edge data processing, i.e., computing capability close to sensor data acquisition. AWS Greengrass and Google IoT Android Things add processing capability to the edge, but none of them look for the best gateway to connect to the cloud. Their IoT components have multiple modules that support several sensors, but the vast majority of modules require machines with many computational resources. The support for the HTTPS, AMQP, AMQP over WebSockets, MQTT, and MQTT over WebSockets protocols allows us to connect a range of IoT devices (sensors and actuators) over the Internet.

To the best of our knowledge, none of them provides roaming facilities for mobile devices.

Table 3.1 compares the main characteristics of some smart city platforms.

3.6 The MUSANet Testbed Implementation

In this Section, we describe how we implemented the testbed used to carry out the experiments reported in this work. The entire procedure is also publicly available on the Internet through MUSANet website [117] to allow its reproduction or general use of the MUSANet architecture and testbed.

Table 3.1: Middleware feature comparition.
✓: feature presented;
F: Free software available;
O: Open source software available;
C: Commercial (payed) system;
X: Feature not available;
?: Feature not documented.

	MUSANet [32]	Dimmer [97]	FIWARE [98]	AWS [113]	ClouT [116]	Watson [115]	Azure [93]	Google Cloud IoT [94]	Stack4Things [95]
Mobile sensor support	✓	X	X	X	X	X	X	✓	X
Distributed network access	✓	X	X	X	X	X	✓	X	X
Processing in the edge	✓	X	X	X	X	X	✓	✓	✓
Processing in the fog	✓	X	X	X	X	X	✓	X	X
Processing in the cloud	X[8]	✓	✓	✓	✓	✓	✓	✓	✓
Group communication	✓	X	X	X	X	X	X	X	X
License	F	?	O	C	?	C	C	C	?
CEP support	✓	X	✓	✓	✓	✓	✓	✓	✓

The testbed was initially deployed on a Dell server with an Intel i7 processor, 16 GB of DDR3 RAM, and a 1 TB secondary HDD storage unit. We use VMWare ESXi 4 as a hypervisor to virtualize the machines where we install mobile client emulators, four slices of ContextNet, and an instance of InterSCity. Despite the low processing, memory, and storage capacity, it was possible to migrate ContextNet, from its Data Center version to the distributed version. More details on how to deploy ContextNet are available on the LAC Wiki [101] and the InterSCity-onibus[9] repository on GitHub.

After completing the prototype test of the geographical-distributed version of ContextNet, we migrated the entire system to virtual machines located in the cloud of the Informatics Department of PUC-Rio (Cloud-DI[10]). Unlike other hypervisor systems, the Cloud-DI hypervisor pre-allocates all the virtual machine's resources so that experiments carried out simultaneously by one team do not interfere with those of another. The platform consists of 16 hosts with 336 vProcs, 1120 GB of main memory, and 7 TB of storage. Communication

[9]https://github.com/meslin8752/InterSCity-onibus
[10]http://suporte.inf.puc-rio.br/molde/corpo_menu.php?link=servicos/cloud.htm (Text in Portuguese)

between hosts is done via a 48-port Gigabyte Ethernet switch. KVM [118] from Qumranet, now Red Hat, is used as a hypervisor.

The initial experiments were carried out with four virtual machines to virtualize four slices of ContextNet, one VM to emulate mobile clients and one VM to operate as a router connecting the other virtual machines via the network. The router, in addition to controlling the flow of data between the virtual machines, allows for selective access to the Internet allowing access via SSH to the virtual machines while creating a controlled environment free of extra traffic inherent to systems connected to the Internet. Figure 3.8 shows the configuration used in the first experiments. Subsequently, we could add six more virtual machines to the initial set, allowing for an expansion in the scalability test of the MUSANet architecture.

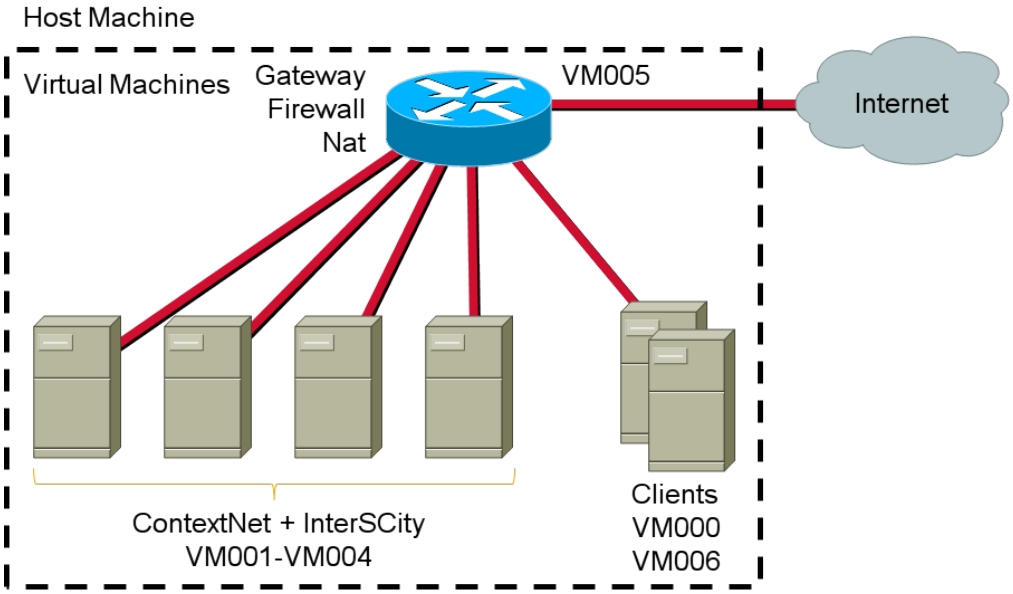


Figure 3.8: Testbed Architecture.

We chose Ubuntu[11] as the operating system to be used on all virtual machines, including the gateway. It is a free and open-source operating system with no license limitations for commercial use. On the other hand, the operating system chosen should not impact the deployment of the testbed or the implementation platform because everything that was implemented in the tests is compatible with most Linux distributions available on the market. Detailed information for the implementation of MUSANet is available on our website [117].

As a tool to emulate the behavior of real networks, we choose the *TC*[12] application that allows for controlling traffic on the network, configuring the

[11]https://ubuntu.com/
[12]http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html

bandwidth per interface, the delay in packet delivery, and even the percentage of packet loss. The developer has a series of tools to implement this task, like Wondershaper[13].

MRTG [119] is another important tool that the developer can use to monitor the network's use by the application over time. MRTG uses SNMP [120] to obtain information on processor usage, network traffic, amount of available memory, etc. The description of the OpenALPRSample application on the MUSANet website [117] exemplifies the installation of MRTG and SNMP using the testbed Gateway (VM005) as an SNMP management station. The data can be viewed through web pages over the Internet using an Apache server, as shown in Figure 3.9.

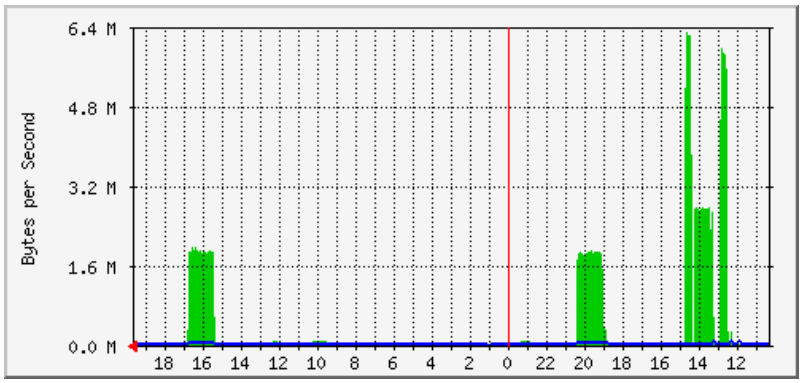


Figure 3.9: Example of data visualization using MRTG and SNMP.

3.7 Discussion

The proposed architecture presented in this chapter allows for data from sensors to be processed at different hierarchical levels in a distributed way, aiming to minimize data traffic and its delay. The information can be analyzed in near real-time or stored for historical analysis. Data processing can be used to trigger alarms, which, in turn, can be stored and/or used as a data source for other alarms. Alarms and applications external to the proposed architecture can send commands to actuators in specific regions.

Its architecture was conceived to allow it to be used both as a design and development tool and as a production tool. The initial tests of its architecture were carried out in an environment composed only of public-domain software and, for the most part, by open-source software. The InterSCity, ContextNet, and Mobile-Hub suite was chosen to present the MUSANet architecture due to some of its unique features, such as, for example, extensive native support

[13]http://manpages.ubuntu.com/manpages/trusty/man8/wondershaper.8.html

for mobile devices, ease of integration between modules, and simplicity in communication protocols, since scalability and elasticity are characteristics present in most middleware for Smart Cities.

The MUSANet modular tiered architecture allows for the programmer to implement data processing in different tiers according to the application's requirements or the need to distribute the processing, bring computing closer to the sensors, or use servers with greater capacity, or, even, all of them. In Chapter 5, we explore distributed processing in several scenarios.

In Section 3.5, we presented some software solutions that could also be used in some tiers of MUSANet to replace InterSCity, ContextNet, and Mobile-Hub.

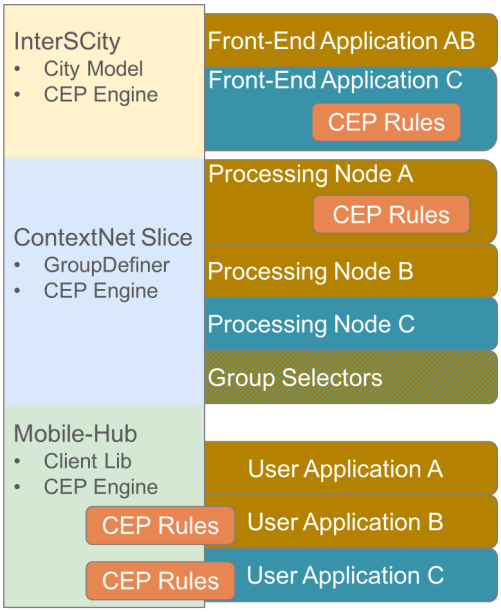


Figure 3.10: MUSANet logic block diagram showing application AB using front-end application AB, Processing Nodes A and B, and User Application A and B. The diagram also shows application C using Front-End Application C, Processing Node C, and User Application C. All application are sharing the Group Selectors.

Figure 3.10 shows a block diagram of the logical part of MUSANet. This example shows two deployed applications, named **Application AB** and **Application C**, and the logical elements involved in them. **Application AB** uses the **Front-end AB** as an interface with users or as a B2B on one hand and with the API provided by the MUSANet cloud layer on the other hand. In the fog, the **Application AB** distributes the computation between **Processing Node A** and **Procesing Node B**. **Processing Node A** uses the CEP engine of its ContextNet slice to process data streams from the edge or cloud, or even from events from other CEP rules. On the edge of the network, we have two mobile applications that are part of the **Application AB**: **User Application**

A and **User Application B**. One of the applications could be an actuator, for example, **User Application A**, and the other an application running on an end user's smartphone, **User Application B**. In this example, **User Application B** receives the CEP rules through Processing Node and executes them in the Mobile-Hub CEP engine. The second application, **Applicaton C**, distributes processing across the three tiers of MUSANet through the **Front-end C**, **Processing Node C**, and **User Application C**. **Front-end C** uses InterSCity's CEP engine to examine the flow of data in the cloud. **Processing Node C** processes data in the fog while a mobile **User Application C** installed on the user's smartphone processes data flow in the Mobile-Hub engine using CEP rules injected by **Processing Node C** before sending data and events to the fog.

4
Evaluating the Architecture

Each one of MUSANet's components had its scalability tested individually [103, 121]. However, this work uses all of them together, and mainly, we use an all-new-brand geographical-distributed version of ContextNet. So, as a proof-of-concept, we evaluate the architecture with all modules together through experiments that verify the MUSANet scalability. The main objective of this evaluation is to verify whether the testbed behaves like an actual Smart City environment, if all its layers, when interconnected, remain responsive, or if it has any limitations that can influence the results of application experiments.

In this Chapter, we describe experiments that examine the MUSANet throughput with a different number of gateways and the number of nodes (sensors and actuators) that can connect to a single gateway, and the time to send a message to an actuator depending on the distance they are from the source. This Chapter is organized as follows. First, we present in Section 4.1 the infrastructure that we used to build the MUSANet testbed. Next, we describe two experiments that we implemented to evaluate the architecture of MUSANet. The first, in Section 4.2, we used to evaluate the performance of MUSANet in terms of the number of connections, slices, and gateways. The second experiment, in Section 4.3, demonstrate the importance of policy and managing connections to the gateway. We end this Chapter, in section 4.4, analyzing the experiment results.

4.1
Testbed Infrastructure

We conducted these experiments by running our infrastructure on a testbed composed of virtual machines hosted in the Cloud-DI, the PUC-Rio virtual-machine cloud. The environment consists of four to ten virtual machines set up as ContextNet slices. Each slice contains a Gateway, a Processing Node, a Group Definer, and a PoA-Manager, as well as a complete instance of InterSCity. We arranged these machines in separate networks connected by a virtual machine with 13 network interfaces. We configured the ContextNet network interfaces with delays compatible with what we would find on the actual Internet for more accurate experiments. InterSCity and ContextNet

could share VM001 in the experiments we present in this paper because all results are related to the edge and fog performance. For InterSCity performance, please refer to Del Esposte [122].

ContextNet [103] was initially developed to be deployed on a local network, within a data center. For us, this implementation would not be useful because it does not allow the distribution of its elements throughout the city, keeping the system geographically centered at a data center in the cloud. It is important to notice that ContextNet, in its original implementation, supports distributed processing using multiple Processing Nodes. However, this processing is restricted to a single data-center since all Processing Nodes must use the same subnet, which is usually implemented on a local network. Through some configuration modifications of its core, which uses OpenSplice's DDS implementation, we were able to configure ContextNet to work in a geographically distributed network domain, dividing it into slices that were deployed in several remote networks connected by a (emulated) WAN. The first experiment carried out with MUSANet, Bus Monitoring, presented in the Section 4.2, was used to validate this modification, verifying the characteristics of scalability, delay and data loss. Although it is not part of the scope of this work to discuss how to do the distributed implementation of ContextNet, details of its configuration are available online at MUSANet web site [117].

All information used to emulate the sensors is composed of real data obtained from the database of the city of Rio de Janeiro. We use data on buses from the city of Rio de Janeiro instead of traffic simulators to make the experiments more realistic. To emulate sensors or buses, we use two virtual machines (VM000 and VM006) connected to the same network. The use of a single network to emulate all buses does not interfere in the experiment because one client does not communicate directly with another client, only with the ContextNet slices. We created a separate thread to represent each bus that appears in the database. These threads connect to one of the ContextNet Gateways in the same way that an Android device running Mobile-Hub would do. All threads act independently, without sharing data, to emulate the actual system as closely as possible. Once connected, the thread begins to send data to stationary infrastructures, just as real buses would. For testing purposes, we control how often the threads emulating buses send data. All virtual machines have Internet access through VM005, which is in charge of isolating the actual Internet network traffic from the network traffic generated by our experiments.

Instructions on how to reproduce this experiment can be found on the Bus Monitoring page[1] on the MUSANet website.

[1]https://musanet.meslin.com.br/bus-monitoring

4.2 Maximum Number of Connections per Gateway

We first performed an experiment to verify the maximum number of connected clients that each ContextNet Gateway supports, as well as to investigate possible scalability issues when increasing the number of Gateways. We performed two different types of tests. In the first set of tests, we emulated up to 8,000 buses in a single region, all connected to the same Gateway. In the second set, we connected up to 16,000 buses to up to four gateways in different networks to verify scalability.

There are many articles reporting application simulations in the fog, such as [123] using iFogSim [124]. Even so, we prefer to use real data of bus movements in the City of Rio de Janeiro provided by the municipality. Although the data can be obtained in real-time, we prefer to use a historical set to allow repeatability of the experiments. Each emulated bus sends data containing its geographical position (latitude and longitude), its speed, its serial number, and its bus-line ID every 5 seconds, 1,000 times. The data received by the Gateway are then forwarded to the Processing Node of the slice to be processed or to be stored in InterSCity. These data are also checked by the GroupDefiner to map mobile nodes to the correct groups. We repeated this experiment 10 times and Figure 4.1 shows the average throughput achieved when connecting 1,000 to 8,000 emulated buses with a single ContextNet Gateway. For comparison, it also shows the theoretical maximum throughput.

We can determine the maximum number of packets per second mathematically by dividing the number of buses used in the experiment by the time in seconds between each packet transmission. The results obtained in the experiment were very close to those previously calculated with a maximum difference of 0.34%.

We next emulated the connection of up to 16,000 buses in up to 4 ContextNet slices. In this experiment, only one of the slices has a Processing Node, so the other slices had to send the data to the slice with the Processing Node. For two slices, the observed results were again very close to the estimated values. Figure 4.2 presents these results, which demonstrate that even with 16,000 clients, connections between the Mobile-Hubs running on emulated buses and the ContextNet slices occurred without problems.

In this experiment, there was no significant loss of packets when using up to 10,000 connections as Figure 4.2 shows. Because we extrapolated somewhat beyond the target number of connections per Gateway, we can notice that with up to 10,000 buses connected, there was less than 0.003% of packets were lost, showing that a small amount of gateways is required considering only

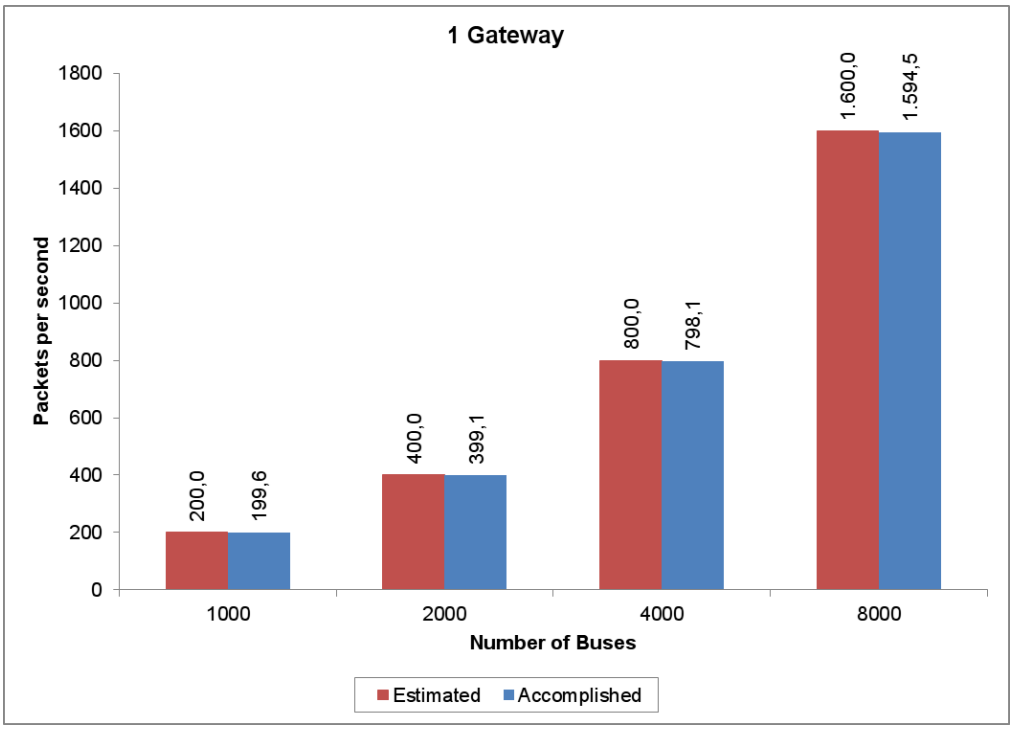


Figure 4.1: Load benchmark with 1 ContextNet Gateway for 1000, 2000, 4000 and 8000 buses. We repeated this experiment 10 times, with a margin of error of plus or minus 0.04% at a 95% level of confidence.

this amount of buses. When we try to emulate above 10,000 buses connected simultaneously to one gateway, connections start to become unstable, and many buses are disconnected. Moreover, when they try to reconnect, they cause an extensive sequence of connections/disconnections, preventing the experiment from continuing. For comparison purposes, we repeated the test using the same infrastructure with FogFlow [125] gateway. The results we obtained were very similar to those observed using ContextNet. We did not run the test with more than one FogFlow gateway because, to the best of our knowledge FogFlow has no load balancing or gateway switching mechanism for connected IoT devices as is the case with ContextNet PoA-Manager.

We next examined data rates and data loss in the communication between buses and the Processing Node. Table 4.1 shows the results for 1 to 10 ContextNet slices. In each case, only one of the slices contained a Processing Node and an InterSCity installation. In experiments with more than one slice, we distributed the buses among ContextNet Gateways. Slices that did not have a Processing Node sent the received data to the slice with a Processing Node. Up to 16,000 buses were emulated, with each bus sending data every 5 seconds. Total losses include data lost during transmission between the bus and Gateway and between the Gateway of one slice and the Processing Node of another slice. Despite the increase in data losses with an increasing number of slices, we can

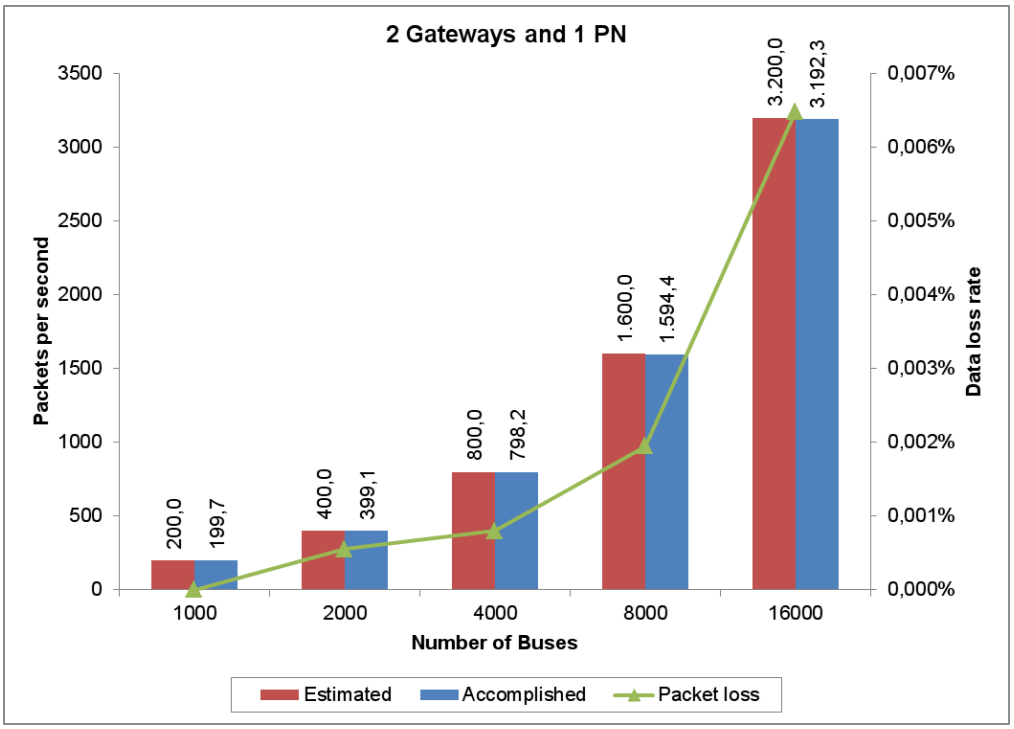


Figure 4.2: Load benchmark with 2 ContextNet Slices. The bars and the left Y-axis represent the throughput, and the line and the right Y-axis, the data loss rate. We repeated this experiment 10 times, with a margin of error of plus or minus 0.04% at a 95% level of confidence.

observe that the percentage of data lost concerning the amount of transmitted data is always low, not exceeding 0.007%, considering the number of connected devices equivalent to the bus fleet for each gateway.

Table 4.1: Data rate for 1000, 2000, 4000 and 8000 buses connected to ContextNet. The maximum throughput for 1000, 2000, 4000, and 8000 buses are 200, 400, 800, and 1600 packets per second respectively. We repeated this experiment 10 times, with a margin of error of plus or minus 0.04% at a 95% level of confidence.

Buses	1 GW	2 GW	4 GW	6GW	8GW	10 GW
1000	199.6	199.5	199.4	199.6	199.3	199.3
2000	399.0	398.7	398.8	398.9	398.6	398.4
4000	797.8	796.6	797.5	797.7	797.0	797.1
8000	1593.2	1594.4	1594.4	1594.8	1591.4	1593.2

Referring to the city of Barcelona [39] in Spain, where 1,800 sensors generate 1,300,000 pieces of data per day [41], i.e., 1.5 pieces of data per second and scaling to a city the size of Rio de Janeiro, which has an area 16 times larger, we can assume that 28800 sensors would generate approximately 24 pieces of data per second. In our experiments, we generate data from 8,000 buses every 5 seconds, i.e., 1,600 pieces of data per second. These experiments

show that even for a large city such as Rio de Janeiro, São Paulo, or Mexico, MUSANet supports the workload without significant loss of performance or packets. The number of gateways used in the deployment of the system is more dependent on the need to create regions or slices of ContextNet to always provide a point of attachment near the sensors than on the number of buses or sensors in the city.

We next conducted tests to determine the relationship between the number of supported connections and memory usage. The graph in Figure 4.3 shows the number of nodes that could connect to a single gateway by varying the amount of available memory. On the X-axis, we have the amount of available memory in bytes, ranging from 2MB to 8GB. The Y-axis displays the number of nodes that successfully connected to the gateway. Through this graph, we can see that the number of possible connections in a single Gateway remains proportional to the amount of available memory up to approximately 8,000 connections. From this point, the curve tends asymptotically to 10,000 connections.

We believe that this behavior is due to TCP limitations because the protocol provides less than 64k ephemeral ports for all applications running in the same machine. Once allocated, the port remains unavailable for a while, even when released.

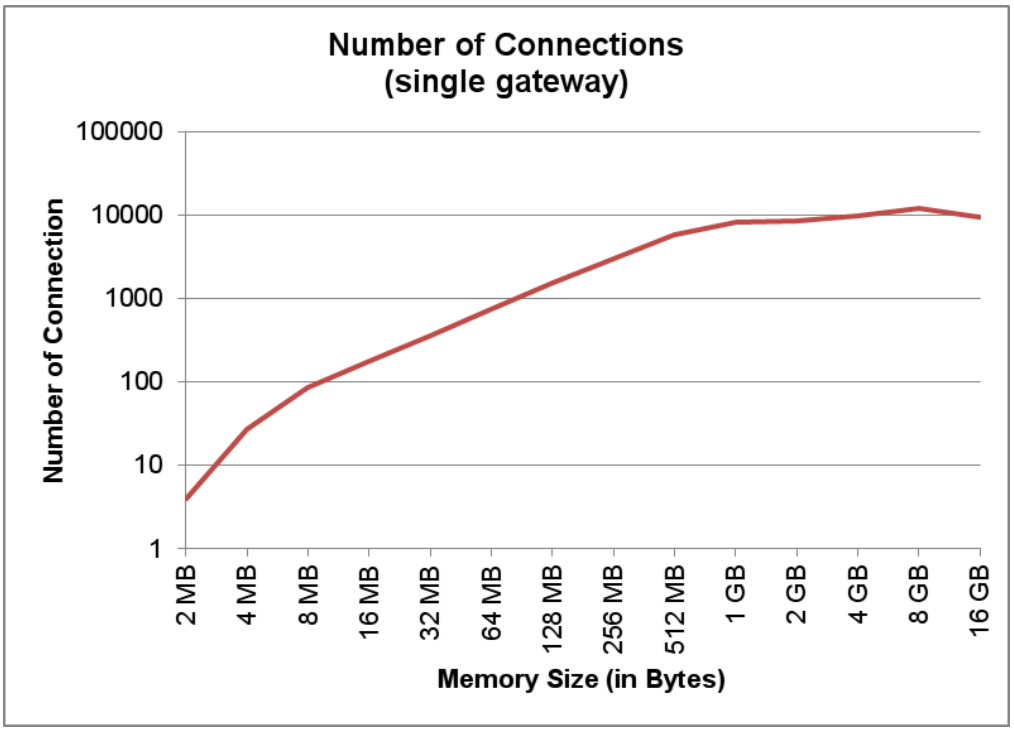


Figure 4.3: The number of connections accepted by a single ContextNet Gateway with a different amount of available memory. We repeated this experiment 10 times, with a margin of error of plus or minus 0.04% at a 95% level of confidence.

Instructions on how to reproduce this experiment can be found on the Bus Monitoring page[2] on the MUSANet website.

4.3 Communication Delays

In this experiment, we try to identify the time elapsed between the occurrence of an event and the notification of mobile nodes within the area of interest for this event. We connected Edge Nodes on ContextNet's gateways using different numbers of network hops to measure the impact of a failed connection policy. We then compare the results with those obtained by an optimal connection policy. We are using buses approaching one of their stops as the event to notify users located in the region of the bus stop. We compare the time it takes to notify mobile nodes (users) connected to the ContextNet slice where the event occurred to the time it takes to notify mobile nodes connected to other slices but with interest in the same event. We use this experiment to demonstrate the importance of selecting the most suitable Gateway for each mobile node.

To make the experiment scenario realistic, we added delays of the order of 100 ms in the network interfaces of the virtual machine that has routing function (VM005) as Figure 4.4 shows. These delays are compatible with those that may be found on the Internet, as measured, for instance, in the path between the sensors and the City Hall website in Rio de Janeiro.

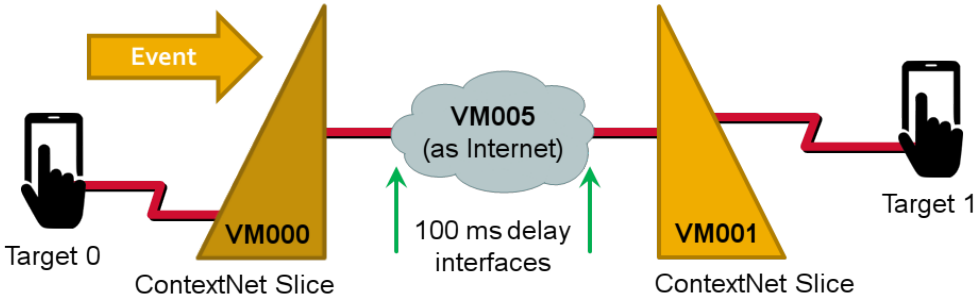


Figure 4.4: Influence of different gateways on application performance. Event inserted through slice VM000 and user in different slices.

For this experiment, we created groups in GroupDefiner representing areas of 100 m^2 around the bus stops, similar to those illustrated in Figure 4.5. We also created groups representing areas along the way that the bus must cross to get to the bus stop. The distance between these areas should be calculated based on the time the bus takes to leave one area and reach another. This time must be sufficient to warn passengers at the bus stop that their bus is arriving.

[2]https://musanet.meslin.com.br/bus-monitoring

Listing 4.1 displays the CEP rule that we used in this experiment. Variable `regionNumber` represents the id of the area near the bus stop that will trigger the event `EventRegion` when the bus enters it.

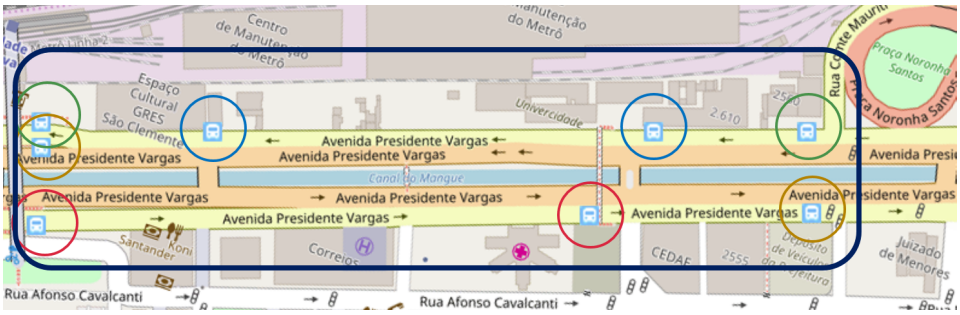


Figure 4.5: Bus stops and their microregions within a broader region. Map from OpenStreetMap[4].

We repeated this experiment 10 times. Table 4.2 presents the average delay for notifications to reach two targets with a margin of error of plus or minus 0.04% at a 95% level of confidence. Target 0 is connected to the gateway selected by the PoA-Manager according to Target 0 geographic position, that is, even if it has moved through the city, it is currently connected to the gateway whose network latency to ContextNet is as small as possible. The second target, Target 1, is connected at any gateway other than the optimal gateway. Each target in this experiment represents a passenger at a bus stop waiting for their bus. In this case, the difference between the average notification time was 118 ms. That is, the time elapsed from the creation of the notification to its delivery to Target 2 was 31.30% greater than to notify Target 1.

For applications involving people such as the one involving bus arrivals discussed here, we can ignore this difference since a person would not be able to notice it, but if we consider applications involving only machines, that time

Listing 4.1: EsperTech CEP code to detect that a bus is arriving at a bus stop. The variable regionNumber represents the number of the region.

```
1 %esperepl
2 create schema Bus(data java.util.Date, ordem string,
     linha string, latitude double, longitude double,
     velocidade double, groups java.util.HashSet, uuid
     java.util.UUID);
3 create schema EventRegion(bus Bus, region integer,
     time long);
4
5 @name(BusArriving) select * from EventRegion having
     EventRegion.region = regionNumber;
```

Table 4.2: Delay between an event and the mobile node notification.

Target	Delay
Target 0	377 ms
Target 1	495 ms

can be relevant. For example, let us consider an application where autonomous vehicles are traveling on a road. An application can coordinate the movement of vehicles on the road while connected to MUSANet, informing them of the current state of traffic lights and when their status will change. Also, each vehicle can inform the infrastructure of its intentions, such as turning right or left at the next crossing street. An essential and critical warning could be the need for sudden braking due to an unexpected obstacle like a pedestrian or animal crossing the road. The coordinating application can inform the other vehicles that are behind to brake as well upon being notified of the need for breaking the vehicle. If we consider vehicles at a speed of 60 km/h, a delay of 118 ms allows the vehicle to travel another 2 meters approximately, which could cause a collision. In many situations, vehicle-related decisions [126] need to be made in near real-time. Other applications in areas such as augmented reality, interactive games, and smart grid [127], also need a low latency between their servers and the final users.

Instructions on how to reproduce this experiment can be found on the Where is my Bus page[5] on the MUSANet website.

4.4 Discussion

In this Chapter, we presented applications developed to investigate the performance of MUSANet, including the integration of its parts – InterSCity, ContextNet, and Mobile-Hub.

The Bus Monitoring applications were used to validate ContextNet configured as a distributed middleware and check the feasibility of connecting ContextNet with InterSCity. These applications showed how versatile the architecture is, which allows its use both in the development and production phases and paves the way for the development of new applications following the steps shown on the MUSANet website [117].

We also performed several experiments connecting Edge Nodes at ContextNet gateways using proximity in terms of the number of network hops as a parameter. We demonstrated the importance of a connection management

[5]https://musanet.meslin.com.br/where-is-my-bus

policy. This policy can be implemented by the developer using ContextNet's PoA-Manager.

During the development phase of these applications, to investigate the behavior of the various applications, we used several tools to obtain data related to the performance of the system and also to make simulations or emulations that resemble real environments. MRTG [119] with SNMP [120], tc, Wireshark [128], sar, and JVisualVM [129] are examples of some tools used. Other tools can easily be installed in MUSANet as it is a system based on pulic domain software and operating system.

The Where is My Bus? applications show how the use of CEP can help the development of applications based on event flows.

For the integration of InterSCity with ContextNet, we have developed an API available on the MUSANet website. We also reconfigured ContextNet to create a geographically-distributed version.

5
Exploring the Testbed

In this Chapter, we introduce four applications that explore the experimentation potential of MUSANet as a testbed. We chose to implement real applications that would be typical of current Smart Cities because the development of an actual application has several specific functional and non-functional requirements. The programmers need to comply with these requirements, unlike with applications built exclusively to evaluate performance when the requirements can be adapted or simplified. In this way, we demonstrate how the developers can implement their application, from design and initial performance testing to a real-world deployment using MUSANet.

The first application, described in Section 5.1, is a three-tier application. This application demonstrates how the mobile support available in all three tiers of MUSANet helps programmers develop an application. Next, we create an application, presented in Section 5.2, that uses CEP on the Edge to reduce data flow to stationary servers by sending consolidate data. Finally, we create two I/O- and CPU-bound applications to investigate the effects of processing the same data in different tiers. These applications are described in Sections 5.3 and 5.4. Figure 5.1 presents the logical diagram of the four applications listed above.

5.1
Three-Tier Application

REGIONALert is an application for managing messages, notifications, and alerts. Messages are sent to users according to their location or preferences. We have three goals with this experiment. First, we want to show how smart city elements can be modeling in InterSCity. Second, this experiment depicts how to input data from external sources into MUSANet elements using the cloud tier. And finally, we exemplify how an application can be divided into loosely-coupled independent tasks, how each task can be assigned to a tier, and how communication can be done between tasks and processes, even if different vendors develop them. This experiment explores the use of MUSANet in terms of process distribution and modeling of physical and logical elements of a Smart City. To accomplish this task, we implemented this application

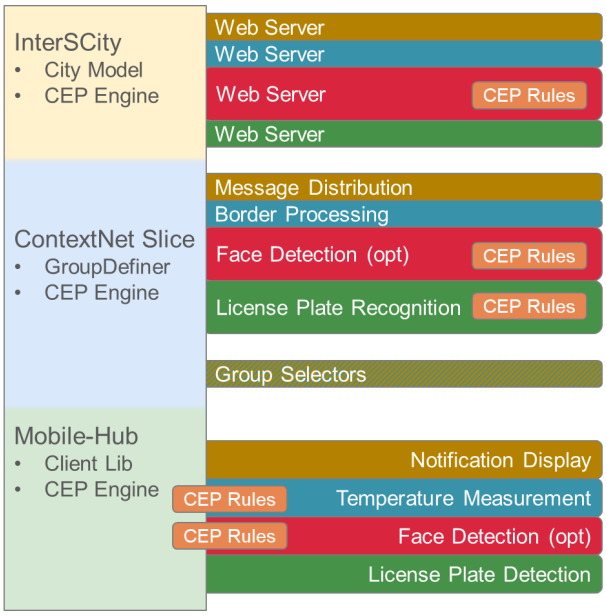


Figure 5.1: This Figure shows the logic diagram of four MUSANet applications: REGIONAlert (in brown), Heat Island Border (in blue), Face Detection (in red) and ALPR (in green).

using a three-tier processing model: at the edge, on the user's Android device; in the fog, collecting user position data and sending relevant messages; and in the cloud, responsible for the smart city entity model and inputting event data generated from an external application.

The REGIONALert application [34] sends announcements from external entities such as the Civil Defense, Highway Patrol, or Tourism Secretary to end users. REGIONALert consists of a set of two Web Services for publishing announcements, a Processing Node capable of retrieving published announcements, and a mobile application installed on the end-user's mobile phone implemented over the infrastructure provided by the MUSANet system. Events can be generated manually by entering data in an external desktop or web application that sends data to REGIONALert web services or collecting data using Reasoning [130], such as a REGIONALert implementation for taxi drivers. In this case, taxi drivers who have registered an interest in serving passengers at one or more airports in the city can be advised that they should be driving to the airport because there is the information of a large number of planes arriving and the number of taxis in the vicinity will not be sufficient to attend to demand.

When using REGIONALert, users can register several areas of interest created by the city administration, such as areas near their residence, their work, their most common route, the place where their elderly parents live, and more. The user's smartphone keeps the system up-to-date as to their location

using the Android mobile application. Whenever a new alert is generated for the user's areas of interest or the area where the user is, the application notifies the user via text, beep, or vibration of their smartphone, according to the settings made in the mobile application by them. Alerts generated for regions outside the user's areas of interest and outside the area where the user is currently located will not be sent. A sequential-numbering message system ensures that the user will not receive repeated messages. All of this guarantees that all messages received by users are of effective interest to them. To the best of our knowledge, most alert systems implemented in large cities send messages without considering the positioning of users.

The REGIONALert system is composed of two independent parts as shown in Figure 5.2: alert generation, maintained – possibly by a Civil Defense Department – as part of the city's infrastructure, and user management, maintained by a third-party organization. There is also a third part of the system: the user smartphone with the third-party mobile application running over Mobile Hub.

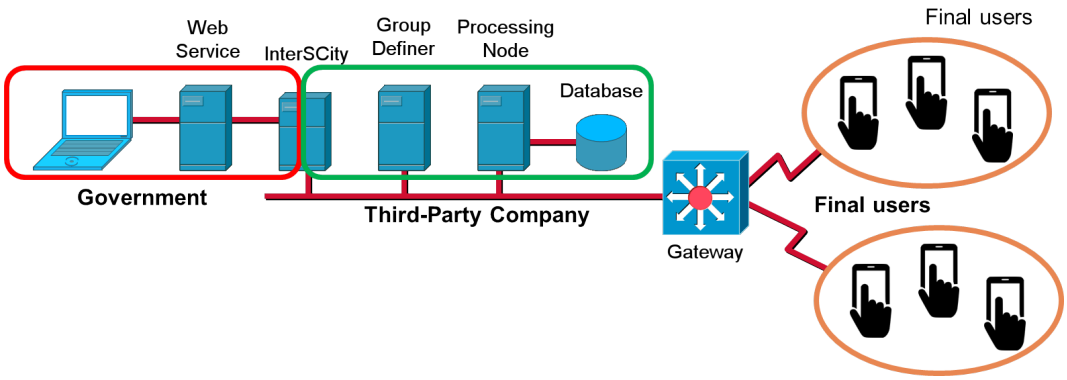


Figure 5.2: REGIONALert system architecture: each module (inside a delimiter) can be deployed by an independent developer.

Figure 5.3 shows how computing is distributed among the MUSANet tiers. We have a web server used for entering new notifications, third-party database servers, and new user registration logic in the cloud. In the fog, the ContextNet GroupDefiner, through the grouping policies defined in the GroupSelectors, associates users with groups characterized by the user's location in neighborhoods or areas. Processing Nodes use this information, along with user preferences, to send notifications made available on InterSCity located in the cloud. On the edge, the applications installed on users' smartphones send their location to ContextNet and display messages received from the fog.

The bottom tier, composed of the Mobile Hub, periodically sends contextualized information from the user to the infrastructure, keeping the Processing Node updated continuously. The Processing Node, located in the

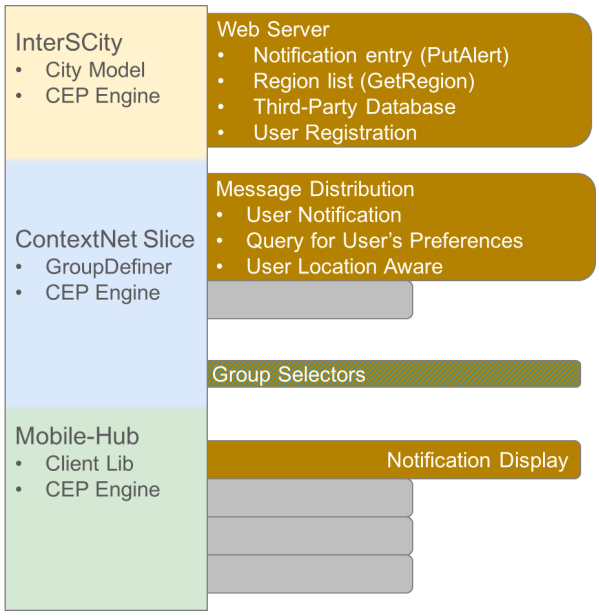


Figure 5.3: REGIONAlert application logic block diagram.

middle tier, can send broadcast or unicast messages over the Internet to notify users.

To allow any to issue announcements, we have created the *PutAlert* Web Service, accessible through the HTTP POST method that requires the announcement start timestamp, its lifetime or end timestamp, the announcement text, and the list of areas affected by the announcement. After storing the new announcement in InterSCity, the PutAlert publishes in the *InterestedInNotices* topic managed by the InterSCity broker to let all third-party organizations interested in announcements know that a new alert has been registered. The creation of new announcements has restricted access because it modifies the InterSCity database. A second web service, called *GetRegion*, is used to get the list of created regions.

The upper tier, composed of InterSCity, is responsible for storing alerts with their region location, expiration date, and text. Since the InterSCity can accept complex queries, including geographic locations, alerts can be quickly recovered.

The implementation of user management is entirely up to the third-party organization. In our prototype, we used a MySQL database to store user data, including the areas which they would like to be warned. To be notified whenever a new alert is generated, the third-party organization Processing Node must subscribe to the InterSCity broker as an actuator with "InterestedInNotices" capability as a topic and provide a service through a public URL and a TCP port. In this case, the InterSCity will notify all registered resources with "InterestedInNotices" capability.

For each new announcement[1], the city hall administration creates a new resource in InterSCity with the announcement information for future reference, and publishes it as "InterestedInNotices" to the broker so that all subscriber applications become aware of the creation of a new announcement. All Processing Nodes that subscribe to "InterestedInNotices" are notified that a new announcement is available. The Processing Nodes search for the announcement in InterSCity using the areas of the announcement and the current timestamp as keys. The current timestamp must be used in the search so that the Processing Node obtains only valid announcements. With the list of announcements, the Processing Node sends the text to all the users present in the areas related to the announcement through the ContextNet Gateway using a groupcast message. The Processing Nodes also search the MySQL database for users who have registered these areas as areas of interest and sends individual messages to them. Note that the Processing Node does not need to know which users are in each group because the ContextNet Gateway is responsible for sending the messages to each of the users individually. The queries from Processing Nodes to InterSCity are made using HTTP.

The Processing Node also monitors the conversation between GroupDefiner and Gateway to find out when a user enters a new area. Whenever a user enters new areas, the Processing Node queries InterSCity to find announcements for those areas. If any, the announcements are retrieved by the Processing Node and sent to the user.

The end user must install the REGIONALert mobile application and the Mobile Hub middleware. Through this mobile application, the user registers their regions of interest, informs the infrastructure of their geographical coordinates, and receives messages containing the announcements of their areas of interest or their current location.

Instructions on how to reproduce this experiment can be found on the REGIONAlert page[2] on the MUSANet website.

5.2 Edge Processing using CEP

This application detects heat islands in a smart city using temperature measurement stations distributed throughout the town or embedded in buses. Our goal with this application is to demonstrate the importance of local processing (on the Edge) combined with CEP to save bandwidth by transmitting consolidate information instead of raw data. Without edge processing, all

[1]The announcement source and how it is generated are not in the scope of this work.
[2]https://musanet.meslin.com.br/regionalert

temperature data collected needs to be sent to the stationary infrastructure. However, with CEP on the Edge, only temperature variations that trigger the alarm are sent. At the end of the section, we compare Edge and Fog processing and their influence on bandwidth occupation.

Heat islands [131] are regions of a city where the ambient temperature rises a lot compared to the surrounding average temperatures. They are caused by overbuilding, poor ventilation, sparse vegetation, air pollution, among others. Heat islands are hazardous for elderly people or citizens that suffer from hearth and blood flow problems [132]. Through temperature sensors embedded in buses, the municipality can obtain temperature data from all over the city while monitoring the bus fleet.

Figure 5.4 shows the computing distributed among MUSANet tiers for Heat Island application In the cloud, we have a web server as the end-user interface (not implemented). the Processing Nodes in the MUSANet's slices, in the fog, compute the heat island border using information received from the mobile applications running on the edge. These mobile applications use CEP rules to detect large temperature variations in a short time, which characterizes the heat island border. This application uses GroupSelectors to load balance gateway connections and Processing Nodes computation. Appendix B.1 shows the CEP rules used to detect when the bus enters or leaves the heat island.

In this application [34], the Mobile Hub obtains external temperatures through a Bluetooth thermometer and uses them as input data to the Mobile Hub CEP engine to search for heat islands in the City. Listing 5.1 shows the rules to detect the heat island border, one rule for entering and other for exiting the island. According to Doering [133], it is less expensive to use mobile sensors

Listing 5.1: EsperTech CEP rule to detect heat island boundary. The first rule triggers when the bus enters the heat island, while the second trigger on its exit.

```
1 %esperepl
2 create schema Temperature(value float, latitude
     double, longitude double, date java.util.Date);
3
4 @name(EnterHeatIslandBorder) select avg(value) as
     avgTemperature, latitude, longitude, value from
     Temperature#length(4) having value > avg(value) *
     1.2;
5 @name(ExitHeatIslandBorder) select avg(value) as
     avgTemperature, latitude, longitude, value from
     Temperature#length(5) having value < avg(value) /
     1.2;
```

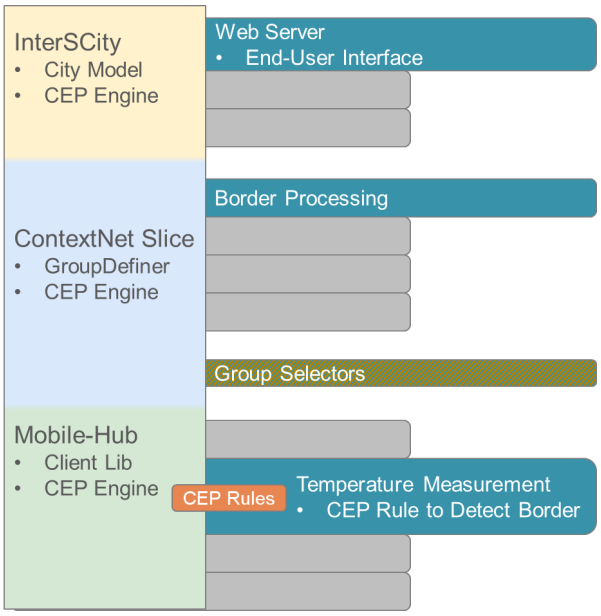


Figure 5.4: Heat Island application logic block diagram. At the cloud, we have a end-user interface implemented as a web server. In the fog, the Processing Nodes compute the heat island border using information received from the mobile nodes. On the edge, mobile nodes use CEP rule to detect temperature variations.

to mesure environmental conditions than create many stationary stations. If necessary, sensors can also be placed where buses do not pass, forming an ad-hoc Bluetooth network or even use participatory crowdsensing [134] through inexpensive Bluetooth sensors [135]. The data captured by the sensor network are transmitted, node by node, to the bridge node, which is located close to the bus route. When a Bus with Mobile Hub is within the range of its Bluetooth radio, the bridge node transmits the temperature information obtained by the other nodes to the bus with Mobile Hub to process the information as Figure 5.5 shows.

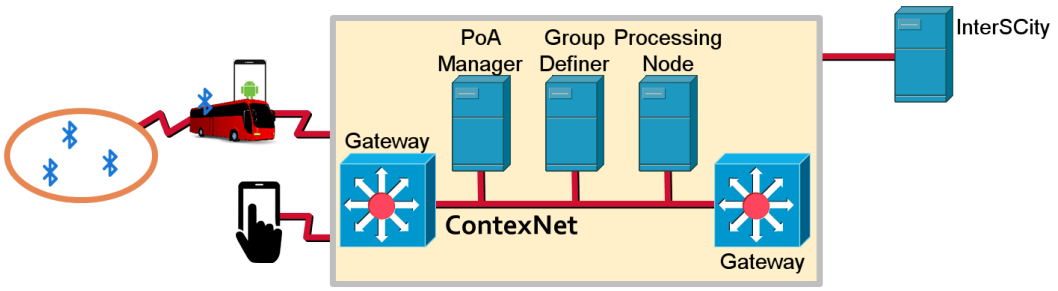


Figure 5.5: Heat Island prototype block diagram.

Buses or other vehicles equipped with temperature sensors travel around the city. When a considerable temperature variation is detected, the CEP triggers a message for stationary infrastructure, the ContextNet. Within

ContextNet, the Processing Node receives information from the buses that detected a possible heat island border to find out its location and temperature range. If required, the Processing Node can feed REGIONALert to send an alert message to all users located in the area stating the existence of the heat island, its position and temperature so that the population can avoid its effects [132] that may include even mortality of the elderly, according to a study conducted by Rosenthal at Columbia University in New York [136].

Upon detection of a new heat island, the Processing Node sends the data to InterSCity to allow for historical and immediate analysis.

If there was no local processing in the Mobile Hub located on the buses and therefore close to the sensors, all collected data should be sent to the stationary infrastructure, implying great use of the radio, resulting in high energy and bandwidth consumption.

Considering a heat island of approximately 3000m^2 in the center of the city of Rio de Janeiro, city buses would have spent an average of 8% of their travel time inside the Island of Heat, representing approximately 24 minutes. Using the local processing based on the Mobile Hub CEP, only temperature data when entering or leaving the heat island would be reported to the stationary facility, which would represent between 3 and 4 transmissions per bus during the day. If there were no local processing, considering that a bus sends temperature data every 10 seconds, it would generate 1,800 temperature data per day. All of these data would have to be transmitted to ContextNet, whether or not there was a heat island, which would considerably increase the bandwidth occupation and energy consumption.

Although the bus electrical subsystem continuously power the Android device radio, and, therefore, power consumption is no longer a problem, we should consider that this is just one particular case of a MUSANet-based application. Moreover even though it is not necessary or essential to save power energy, the significant decrease in data transmission achieved with local processing allows many other applications to share the same network bandwidth. Examples of other applications are the detection of holes and other imperfections in the asphalt using impact sensor in the buses, gas levels such as O_3, CO_2, through its respective sensors, detection of burned or obstructed lamps in public lighting through light sensors, and more.

Instructions on how to reproduce this experiment can be found on the Heat Island Detection page[3] on the MUSANet website.

[3]https://musanet.meslin.com.br/heatislanddetection

5.3 CPU- and I/O-Bound Application

In this section, we depict the Face Detection application. This application detects frontal faces using the OpenCV library [137]. We choose this library because it allows us to develop the same Java application to run on the Edge (on Android smartphones with Mobile-Hub) or in the Fog (on stationary servers). Our goal with this experiment is to compare processing on the Edge, sending consolidated data to the fog, and processing in the Fog that receives raw data through an I/O- and CPU-bound application. To make this comparison, we selected some data transmission technologies and considered the effective bandwidth offered by them. The same application can detect full-body, eyes, license plates, coins, and so on just by adding a publicly-available XML configuration file.

The applications described so far have used data flows from sensors that generated a transfer of a few bytes per second between Mobile Hub and ContextNet. Another common point of these applications was how the developers distribute the processing among the tiers. They distributed the processing only based on the applications' requirements and not on the tier characteristics, such as processing power or network bandwidth. In these applications, the developer cannot choose the tiers where the processing would be performed since the applications' requirements dictated the choice of the tier, and, mainly, did not allow the processing to migrate from one tier to another.

The Face Detection application was the first application we developed that uses a video stream and can process data either on the edge or in fog, or even both. In order to compare processing costs on an Android device and a stationary server, we chose to implement the application using the OpenCV[4] open-source library, since this library can be used on Android devices and desktop computers. The MUSANet website presents in detail how to implement the application on the edge or in the fog. Figure 5.6 shows the Face Detection Android app detecting faces using the painting Operários[5] by Tarsila do Amaral.

Figure 5.7 shows how the computation is distributed through the MUSANet tiers in the Face Detection application. The application has a Front-End (not implemented) in the cloud, formatting the end-users output, for example, allowing for a security authority to check the number of people in an area. The face detection can be accomplished in two tiers: in the fog or on the edge. For faster processing, the application can detect faces in the fog.

[4]Available at https://github.com/opencv/opencv
[5]http://tarsiladoamaral.com.br/obra/social-1933/

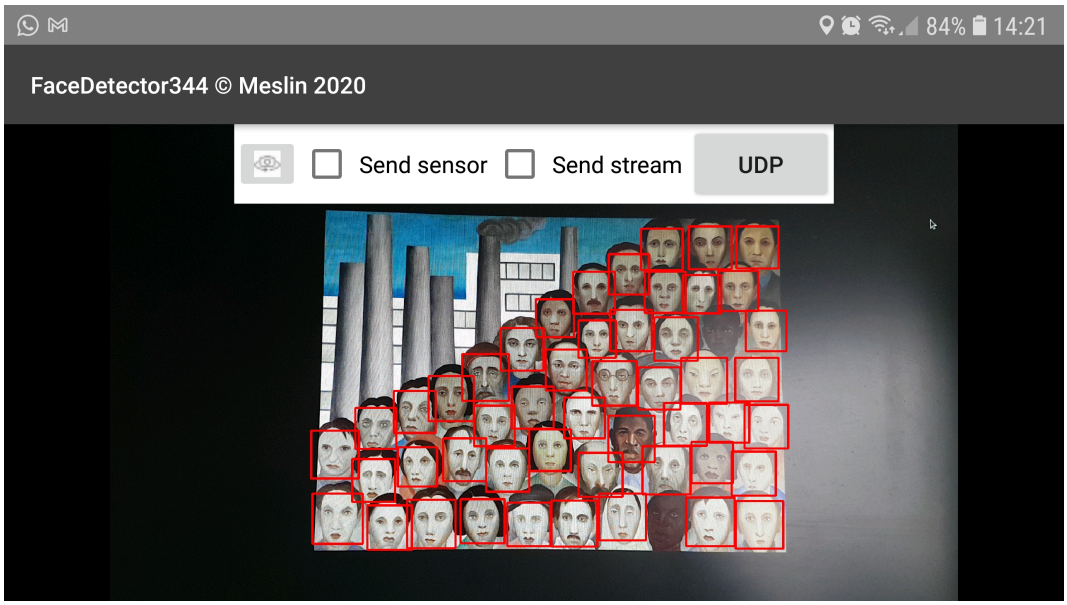


Figure 5.6: Example of face detection. This is an example of a Face Detection application using Tarsila do Amaral ink called OPERÁRIOS, 1933, oil on canvas, 150 x 205 cm. The operators can choose to send sensor data, video stream, or both. They can also choose among UDP, TCP, or MR-UDP by touching the button at the right.

In this case, the Processing Nodes will receive the video stream sent by the edge devices. Also, in the fog, we can use CEP rules to detect crowds based on the size and amount of faces detected. We present an example of this CEP rule in Appendix B.2. Instead of detecting faces in the fog, to save network bandwidth, if the programmer chooses to detect faces on the edge and send a list of rectangles (the red rectangles in Figure 5.6) to the fog, the CEP rule to detect crowds can be executed in the Mobile-Hub CEP or Processing-Node engines. In both cases, the rule's entry that will generate the event related to the crowd will be the flow of rectangle generated by detecting the faces.

To study the performance of the application under different distributed scenarios, we have developed two versions of the same application. The first version runs the face detection algorithm on Android devices, capturing images through its camera. The captured images are analyzed locally for face detection. The number of faces detected in the frame is sent to a Processing Node located on ContextNet using the standard ContextNet protocol, MR-UDP.

A second version of the application was developed to process the images on the ContextNet Processing Node. In this version, the Android device is used to send a video stream to the Processing Node. The Processing Node analyzes the flow and counts the number of faces detected in the images received by the Android smartphone camera.

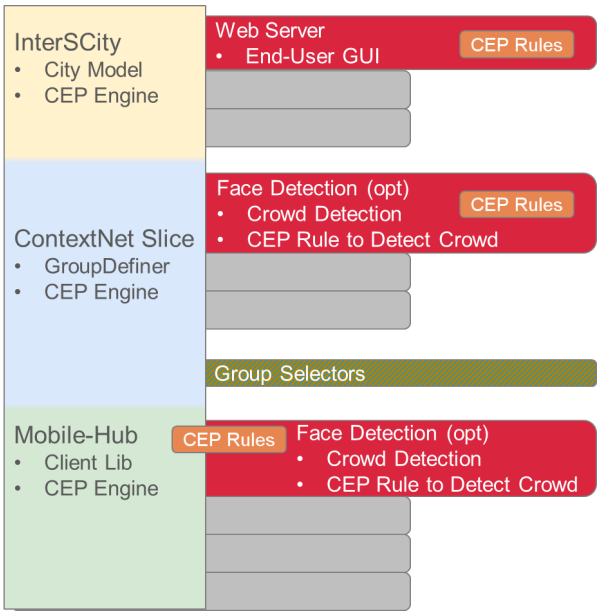


Figure 5.7: Face Detection application logic diagram.

During the testing phase, we found that using the MR-UDP protocol [102] to send large data streams, such as a video stream, is highly inefficient. Although based on the UDP protocol, which is not reliable, MR-UDP adds reliability by implementing confirmation of each received packet and retransmission of packets considered lost. For transfers of small amounts of information (less than 1500 bytes), this protocol is highly efficient for mobile devices. However, when the amount of bytes increases, for example, in an image or, mainly, in a video, the confirmation of each packet received adds considerable overhead, making it unsuitable. As a work-around for this problem, we took a new approach using TCP. Unlike MR-UDP, TCP implements windowing; that is, the confirmation of received data is done by groups of packets and not by each packet, reducing overhead. However, retransmission of packets lost in a video stream is unnecessary since the received frame can be reconstituted based on the previous frames with little loss of quality. Since the UDP protocol does not implement confirmation or retransmission, for this particular case, video streaming, the UDP protocol presented the best results.

The application can detect a face in 0.6 seconds on average using a Samsung Galaxy S7 cell phone through its main camera, in Full-HD resolution (1920x1080 pixels). A set with 160 faces in the same resolution takes, on average, 1.4 seconds to be detected. A Processing Node running on a computer with an Intel Core i7-10510U processor and 16 GB of main memory detects an average face in 0.05 seconds, that is, approximately 28 times faster.

In a real application, face detection at the edge of the network on an Android device would hardly compromise an application's performance. An

application that needs more image processing could experience performance losses when running on an Android device. To investigate this effect, we chose the recognition of license plates, presented in Section 5.4.

Instructions on how to reproduce this experiment can be found on the Face Detection page[6] on the MUSANet website.

5.4 A more CPU- and I/O-Bound Application

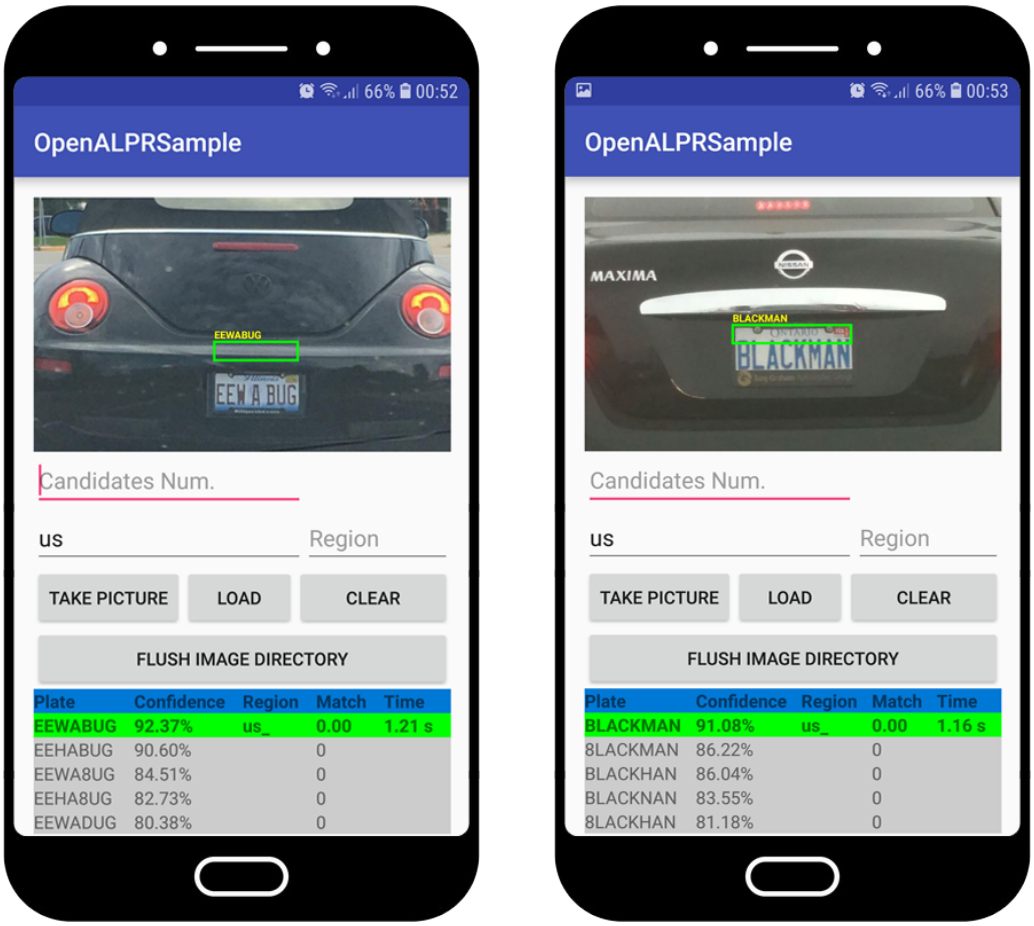
The Automatic License Plate Recognition application detects and recognizes license plates using the open-source version of the OpenALPR library [138]. Like the Face Detection application presented in Section 5.3, this application is CPU- and I/O-bound and can be executed both on the Edge or in the Fog. However, this application requires more intensive use of CPU than the face detection algorithm. This application aims to analyze the computing options on the edge and in the fog, including the distribution of the computation between these two tiers.

To investigate the effects of automatic license plate recognition, we created two applications. The first performs the entire recognition process on the edge of the network, on Android devices, and sends the results to a stationary server. The second application receives images from a camera located on the network's edges and detects license plates in the fog, using a ContextNet Processing Node. The source codes are available on GitHub[7] [8]. To allow a better comparison, both applications were developed using the free and open-source library OpenALPR.

The plate recognition begins with its detection. The algorithm uses an XML file generated after training a neural network using approximately 10,000 license plate images to perform the detection. In this work, we use XML files provided by the ALPR library developer since the objective is to measure costs, and the development of a real application is outside our scope. Once detected, the plate is recognized, and the probable values are compared with a model with possible variations of letters and numbers to improve the result.

Regarding the precision of the results, both implementations showed the same degree of reliability, varying between 80% and 95%. Figure 5.8 shows the results of the capture and recognition of license plate on a smartphone. As we can see, depending on the image quality, which includes the quality of the original license plate, the application running on a smartphone can recognize a license plate in 1 second on average, with more than 90% confidence.

[6]https://musanet.meslin.com.br/face-detection
[7]Android version: https://github.com/meslin8752/OpenALPRSample
[8]Server version: https://github.com/meslin8752/OpenALPR



5.8(a): License plate recognized in 1.21 seconds and 92.37% confidence.

5.8(b): License plate recognized in 1.16 seconds and 91.08% confidence.

Figure 5.8: Example of license plate recognition by an Samsung S7 Android device.

The version for a stationary server that runs on a ContextNet Processing Node can recognize a plate in 200 ms, as Figure 5.9 shows. To perform the recognition, Processing Node needs to receive the images captured by an IP camera or a smartphone. This data transmission consumes an average of 800 kb/s of bandwidth, reaching 3.2 Mb/s [139] for Full-HD videos.

Alternatively, the developer could choose to distribute the processing between the edge and the fog. On the edge, the Android device could detect the license plates using the OpenCV library as implemented in Section 5.3 instead of implementing the full license plates recognition algorithm. Detection is much faster – 17 plates can be detected in 1 second and allow only images that actually have plates to be transmitted, saving bandwidth.

Table 5.1 shows the time required to transfer the image in Full-HD resolution (1080x1920 pixels) from an Android smartphone to a ContextNet Processing Node. The experiment was repeated 1000 times for each technology presented. Through Table 5.1, we can see that the MR-UDP protocol is not

Plate Number	Confidence 232 ms
- EEWABUG	93.206856
- EEHABUG	87.830269
- EEWA8UG	85.560028
- EENABUG	84.237457
- EEWABU6	83.204773

5.9(a): License plate recognized in 232 ms.

Plate Number	Confidence 206 ms
- BLACKMAN	90.746094
- 8LACKMAN	83.971054
- BLACKHAN	83.631454
- BLACKMAM	83.470184
- BLCKMAN	83.200081

5.9(b): License plate recognized in 206 ms.

Figure 5.9: Example of license plate recognition by a stationary server. CPU Core Intel I7, 10th generation.

suitable for large image transfers. The UDP protocol was not considered because it is not a reliable protocol. That is, it does not implement the recovery of lost datagrams. In this case, unlike in video transmission, if part of the image is lost during transmission, the entire image is compromised and will have to be discarded.

Table 5.1: Transmission time of a 1.4 MB image using TCP and MR-UDP over some network layer 1 technologies.

	TCP Time (ms)	MR-UDP Time (ms)
3G @ 2 Mb/s	5,706	51,662
4G @ 100 Mb/s	112	3,948
5G @ 1 Gb/s	14	2,896
802.11a @ 54 Mb/s	209	3,907
802.11b @ 11 Mb/s	1,137	8,245
802.11n @ 600 Mb/s	19	3,960

Network protocols and data serialization may add considerable overhead to data transmission. For example, transfer via TCP a Full-HD image of 1.4 MB (1,360,336 bytes) transfers in only one direction may result in up to 8,294,873 bytes transferred depending on the serialization type used to pack the data. This overhead consumes network bandwidth, adds delays, and influences the device's energy consumption on edge, and should be one of the points considered when deciding how to and where to implement the application's computing distribution.

Using the cell phone to detect and recognize a license plate has a high cost in terms of time, but the transmission of consolidated data, in this case, the license plate's text, saves a lot of bandwidth. When we moved the processing to the Processing Node within a ContextNet slice, the processing time dropped by approximately 85%. Still, the video stream can consume much of the bandwidth, depending on the technology involved.

However, using what we have learned from the Face Detection application, we can divide license plates' detection and recognition processing into the bottom and middle tiers. The bottom tier can easily detect license plates, separate the plate image from the rest of the captured image, and transmit it to the Processing Node. In this case, the transmitted image would be a tiny fraction of the original image, allowing for bandwidth savings. The Processing Node, instead of looking for a license plate in a large, high-resolution image, would only recognize the license plate in a small image, allowing for reducing processing time.

To evaluate this alternative, we have implemented a third version of the License Plate Recognition application. Using a 4k resolution camera, it was possible to detect, without recognizing, a license plate in less than 500 ms on a Samsung S7 smartphone. Unlike face detection, a license plate is much simpler to detect – a license plate is basically a rectangle with a known form-factor and color. The processing time to extract the portion of the image that contains the license plate was 1 to 2 ms. The image transmission cost a few dozen bytes, and the processing on the stationary server was below 200 ms. The use of the MUSANet testbed to implement this ALPR prototype allows us to prove that this solution presents better performance than the other two previous versions, saving processing time and network bandwidth. As the production environment is identical to the test environment, we expect this result to be repeated after the application is deployed.

One can use the Automatic License Plate Recognition application to measure vehicles' average speed on a route, as Figure 5.10 shows. Two Android devices running Mobile-Hub on the edge, positioned at a known distance, can send license plates that they detect – enriched with context such as time and location – to a Processing Node in a ContextNet slice in the fog. Processing Node recognizes each received license plate and feeds a CEP engine with this information, including the timestamp. The CEP rule will instruct the CEP engine to compare a flow of license plates sent by the second Android device with the most recent license plates sent by the first device. If a license plate sent by the second device is within the set of license plates sent by the first device, considering a specific time interval, the detected vehicle has exceeded the average speed allowed on the route. This information can be stored in the cloud using InterSCity, sent to or queried by the final user using the front-end application.

For example, suppose a road has a speed limit of 90 km/h. To detect vehicles that exceed this speed, we can place two checkpoints 250 meters apart. If a vehicle, identified by its license plate, passes through the two checkpoints

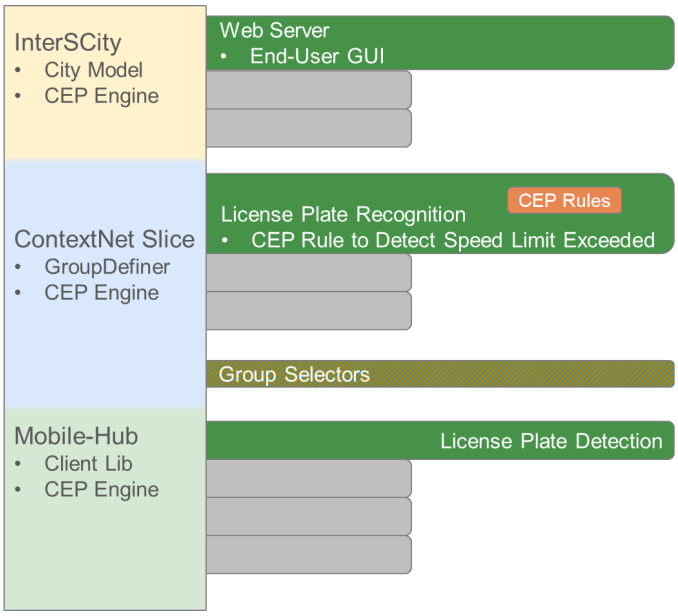


Figure 5.10: ALPR application logic block diagram.

in less than 10 seconds, the vehicle has exceeded the average speed limit for that segment. Listing 5.2 shows the CEP rule used in this example. Note that each checkpoint must include its time and location context. Listings B.6 and B.7 shows this CEP rule running with some vehicles passing through the checkpoints.

Instructions on how to reproduce this experiment can be found on the Automatic License Plate Recognition (ALPR) page[9] on the MUSANet website.

5.5 Discussion

We organized this section as follows. First, we discuss the applications that we use to experience the development and implementation environment of

[9]https://musanet.meslin.com.br/alpr

Listing 5.2: EsperTech CEP code to detect wheather a vehicle has exceeded the average speed limit between two control points.

```
1 %esperepl
2 create schema LicensePlate(plateNumber string,
     timestamp java.util.Date, measuringPoint string);
3
4 @name('Detection') select * from pattern [every p1=
     LicensePlate() -> p2=LicensePlate(p1.plateNumber=p2
     .plateNumber and p1.measuringPoint != p2.
     measuringPoint) where timer:within(10 seconds)];
```

MUSANet in Subsection 5.5.1. Then, we present an application classification according to the possibilities of dividing it into tiers in Subsection 5.5.2. Finally, in Subsection 5.5.3, we show how the support tools have helped us develop each application.

5.5.1 MUSANet Applications

In this Chapter, we presented four real applications using MUSANet: Heat Island detector, using CEP; REGIONALert, a three-tier application; and Face Detection and Plate Recognition, two CPU- and I/O-bound application.

Heat Island is an application developed to MUSANet based on CEP, but, unlike Where is my Bus, presented in Section 4.3, the CEP rules were deployed on the edge. This implementation reduced the data sent to the stationary servers, saving bandwidth and energy. This experience also shows how CEP helps to reduce the program's complexity whenever a data flow must be analyzed. In this application, the CEP rules could also have been applied in Processing Node. However,in this case, all temperatures obtained by the buses on edge would have to be sent to the fog, consuming network bandwidth.

Using the REGIONALert application, we exemplified basic smart-city modeling and three-tier processing distributing. In the cloud data center, the InterSCity models and stores the smart city's elements and information. It also provides a Pub-Sub microservice to notify all subscribed (the Processing Nodes) of new events. In the fog tier, implemented by ContextNet slices, GroupDefiner groups users according to their location, allowing the Processing Node to send relevant messages. The app developed for users' smartphones send their location to ContextNet using Mobile-Hub and receive and show the messages to users on the edge tier. This application exemplifies how processes located in each tier can interact with other processes in another tier.

The Face Detection and ALPR applications show how processing can be analyzed and distributed, not only in one tier, across multiple computers, but also among tiers. Distribution can be done by migrating computation from one tier to another depending on the network, power, or computational resources such as CPU and memory. Besides, the ALPR application showed that the same task – recognizing a license plate – can be divided between two tiers, the Edge, which will detect the existence of one or more license plates in a video, and the Fog, which will recognize the license plate and decode its characters. Listing B.4 on Appendix B.2 shows an example of a CEP rule that one can use to detect crowds. As this rule can be deployed either in the Fog or on the Edge tiers, one can use Magalhães' work [140] to control where to apply the rule.

A future work with the Face Detector application would be to adapt the program to analyze agglomerations. Since face detection is done using a flat image, where there is no depth information. That is, an image with 30 detected faces can be an agglomeration if they are all close or a set of people apart if the faces are far from the camera that captured the image. One way to adapt the application would be to consider that all human beings have approximately the same face size and calibrate the rectangle's size calculated by the application when detecting each face. Once calibrated for a certain device, a simple calculation of triangle similarity can obtain each face's distance to the camera and allow to check if there are many people close to each other or not.

We realized that the MR-UDP protocol, natively used for Mobile Hub and ContextNet to communicate, is not suitable for the transmission of images and videos when transmitting these flows using the Face Detection and ALPR applications. To work around this problem, we added TCP and UDP, preserving MR-UDP for transmitting small data streams.

Regarding how processing is distributed and allocated in the MUSANet tiers, we identified basic patterns that classify applications in three classes: (1) collaborative distributed processing, (2) migratable distributed processing, and (3) multitier distributed processing. Distributed processing is inherent to the architecture of systems that use fog as one of its tiers. In our case, processing is inherently distributed on the edge and in the fog, and can also occur at the cloud.

5.5.2 Processing Classes

Collaborative Distributed Processing Class: in this class, each tier executes part of the application's algorithm according to its special characteristics. The edge tier is responsible for data acquisition and control the actuators. This tier can also consolidate, filter, and aggregate data and insert context information. The fog tier collects the information sent by the edge tier, and takes decisions for its region. If necessary, this tier can also send information to the cloud for global decision taking. The cloud tier stores data for historical consultation or uses it to present to end-users or make global decisions. The programmers have few, if any, options in choosing which part of their algorithm will run on each tier. An example of collaborative distributed processing is the REGIONAlert application presented in Section 5.1. Table 5.2 lists all the macro functions involved in REGIONAlert and correlates them with each MUSANet tier according to their capabilities. As we can notice, there is only one option for allocating functions, so the designer cannot choose

to distribute them in other tiers, regardless of any requirement such as the amount of memory, processing capacity, or energy available in the devices. The Bus Monitoring and the Where is My Bus applications, shown in Sections 4.2 and 4.3, respectively, are also examples of application classes.

Table 5.2: Macro-functions allocation for the REGIONAlert application, an example of Collaborative Distributed Class.
✓: tier supports the activity.
X: tier does not supports the activity.

	Cloud	Fog	Edge
Data entry	✓	X	X
Permanent data storage	✓	X	X
Display message	X	X	✓
Model	✓	X	X
Send messages	X	✓	X
Send user's location	X	X	✓
Users' location aware	X	✓	X

Migratable Distributed Processing Class: in this class of application logic, processing or part of the algorithm can migrate from one tier to another according to the moment when the application is executed. For example, with more bandwidth and power available on the edge devices, the programmer can migrate part of the edge processing to the fog. On the other hand, if there is a need for quick decision-making or any network disconnection, programmers should consider allocating processing on edge. The consolidated information will be sent to the fog when the connection is reestablished. An example of migratable processing is the Face Detection application presented in Section 5.3. Table 5.3 lists all the macro functions involved in the Face Detection application and correlates them with each MUSANet tier according to their capabilities. Some functions can be performed only on a specific tier (such as image capture and permanent data storage), while other functions can be performed in more than one tier (such as face detection and face count). This last type of function allows programmers to choose the best tier to deploy it according to the run-time characteristics or even to migrate the function during execution. Face detection can be performed quickly on mobile devices but is performed even faster on stationary servers such as those used in the fog. If there is sufficient bandwidth or need for regional decision making, face detection can be done in the fog. If the edge device is low on energy or the network bandwidth is severely constrained, face detection can migrate from fog to edge. In this case, only the face quantity or the position and size of the faces in relation to the camera are sent to the stationary servers. The Heat Island Detection 5.2 application

is another example of this class of applications because the CEP rule can be installed either in the fog or on the edge.

Table 5.3: Macro-functions allocation for the Face Detection application, an example of Migratable Distributed Processing Class.
✓: tier supports the activity.
X: tier does not supports the activity.

	Cloud	Fog	Edge
Image capture	X	X	✓
Face detection	✓	✓	✓
Face count	✓	✓	✓
Crowd detection	✓	✓	✓
Permanent data storage	✓	X	X

Multitier Distributed Processing Class: in this class, the processing is divided among some tiers of MUSANet in a pipeline similar to a car assembly or decoding instructions on a processor. One tier performs part of the processing and sends the data to the next tier. Unlike in the collaborative processing class, one or more tiers could do all the processing, but they do not do it because another tier could do it more efficiently. An example of an application that uses this class is the License Plate Recognition application 5.4. Table 5.4 depicts the main macro-functions of the License Plate Recognition application. The edge tier could capture the image, recognize the license plates, and send only the license plate text to Processing Node. But suppose instead of performing all this processing, the edge tier detects license plates and sends only the images of the license plates. In that case, Processing Nodes can recognize license plates faster without consuming a lot of bandwidth.

Table 5.4: Macro-functions allocation for the Plate Recognition application, an example of Multitier Distributed Processing Class.
✓: tier supports the activity.
X: tier does not supports the activity.

	Cloud	Fog	Edge
Image capture	X	X	✓
Plate detection	✓	✓	✓
Plate recognition	✓	✓	✓
Permanent data storage	✓	X	X
End-user GUI	✓	X	X

5.5.3 MUSANet Development Tools

In this chapter, we used several tools presented in Chapter 4 to learn about applications' behavior during the development process. We use the TC – Traffic Control – app to limit bandwidth and add packet delays as described in Section 4.3. We also use it to simulate various network technologies, varying the bandwidth and the delay, to investigate the influence of the type of transmission medium on the face detection application. For installation and use instructions, see the MUSANet website [117].

Through Oracle JVisualVM [129], we were able to monitor the ContextNet Gateway's behavior when attempting to connect many mobile devices. The tool allows for examining the CPU usage, allocated memory, and heap size. It also allows for clearly see the moments when the Garbage Collector from Java releases the application memory. Comparing the graphics of memory consumption, CPU, and the Garbage Collector, we could see that the ContextNet Gateway allocates many resources in memory at the time of a connection and then releases them. Consequently, if many connections occur in a brief period, the Garbage Collector will not have been triggered to free up memory, preventing new connections from being made or existing connections from being maintained.

MRTG [119] is a tool widely used by network administrators to monitor traffic at crucial points and trigger alarms if there are any abnormal situations. The tool offers a graphical interface via a web browser, allowing visual analysis of the network situation near real-time. The sampling time can be configured, but increasing the sampling rate too much can compromise the system's performance. Although we used MRTG in practically all applications during the development phase, its use to monitor traffic on the network made it possible to prove the problem that had already been noticed when using the face detection application. The Face Detection application visually demonstrated problems when we transmitted the video stream via MR-UDP or TCP. MRTG showed the large data stream graphically.

Wireshark [128] is a monitoring tool that is also widely used by network administrators. With this tool, we can capture packets, analyze data flows from connections, and much more. This tool was also handy, together with the MRTG, to prove that the problem with the "Face Detection" application was really in the protocols used. The tool also allowed us to analyze the transmission size of a slice of an image containing the license plate to support our decision to implement the license plate recognition application, implementing the system as a Divided Distributed Processing Class.

In addition to the external tools, we use several programming patterns to obtain data to evaluate application performance. We believe that the three most important standards were synchronizing threads so that all processing could start simultaneously, the synchronization to ensure that all threads have finished processing to calculate the elapsed processing time. Listing 1 presents the first code pattern. It is a fragment of Java code that can be used to synchronize all threads to start computing simultaneously. In this example, threads can be created inside a repeat instruction within the synchronization block. Listing 2 shows the code that must be used by the thread, after its initialization and before starting the actual computation, considering that the programmer has chosen to use the thread start synchronization pattern. Next, Listing 3 presents the code to synchronize all threads at the end of processing. This code waits for all threads to finish processing to proceed. And finally, Listing 4 computes the time from the start of processing to the end with millisecond precision. The commands to obtain the start time and end time must be placed immediately before and after the computation, respectively.

Listing 5.3: Java code to synchronize threads to start computing. The object `canStart` is a global object class

```
1 synchronized(canStart) {
2     <create here all threads>
3 }
```

Listing 5.4: Java code to be inserted in each thread after startup processing, before the main computing instructions.

```
1 synchronized(canStart) {}
```

Listing 5.5: Java code to wait for all threads to finish. The variable `nThreads` represents the number of threads, and the variable `eachThread` is a tread list.

```
1 for(int i=0; i<nThreads; i++) {
2     try {
3         eachThread[i].join();
4     } catch (InterruptedException e) {
5         e.printStackTrace();
6     }
7 }
```

Listing 5.6: Java code to get the elapsed time.

```
1 startTime = System.currentTimeMillis();
2     <do all computing here>
3     <don't forget to wait for all threads to finish>
4 stopTime = System.currentTimeMillis();
```

6
Conclusions and Future Work

We organized this chapter into three sections. First, in Section 6.1, we briefly describe our research and implementation work. Then, in Section 6.2, we depict the main contribution of this research work. Finally, in Section 6.4, we point out some future work derived from this.

6.1
Research and Implementation Work

The architecture we propose in this research work allows for data from sensors to be processed at several levels in a distributed way, in order to minimize data traffic and delays. Data processing can be used to trigger notifications which can be stored and/or used to feed other alarms. Alarms and applications can be used to send commands to actuators in specific regions. ContextNet is able to provide multiple Gateways to the MUSANet infrastructure that can be spread throughout the city in a fog. The PoA-Manager will ensure that the mobile smart object will always be aware of the best ContextNet Gateway to connect to. In a deployment in a densely populated environment of sensors and users such as hospitals, stadiums, and airports, the PoA-Manager can distribute smart objects connections to the nearest gateways, mitigating congestion or disconnections when accessing ContextNet.

To implement MUSANet, we needed a scalable component that could model a smart city supporting mobility. We chose to use InterSCity because it meets all the project's functional requirements: it is scalable, elastic, based on microservices, models city elements, storing data in a structured way while allowing it to be retrieved with high-level queries. To integrate InterSCity into the project, we studied its architecture and understood how it works. Although InterSCity exposes a restful interface based on HTTP, during the development of the first applications, we realized the difficulty of interfacing the Processing Nodes with InterSCity, so we decided to develop an API to facilitate communication via HTTP.

The integration between ConextNet and Mobile-Hub was simple in the first tests since they were initially designed to work together. The ClientLib API in Java, available for Android and desktop applications, allows the

developer to connect Mobile-Hub with ContextNet. Later tests showed that the communication protocol used by them is not suitable for transferring large amounts of data – images or videos. To get around this problem, we had to add two protocols in the communication between Mobile-Hub and the ContextNet Gateway. We added TCP for file transfers and UDP to send large data streams, such as videos.

One of our goals was to create a scalable environment where the systems architect or programmer could develop their applications for Smart Cities that could be implemented in the real world. To determine the scalability, we developed an application to check the maximum number of connections that each of the ContextNet gateways could support. This same application could also be used to check the throughput and the amount of data lost when the gateways were subjected to an extreme number of connections and data sending. The results obtained were satisfactory, mainly concerning data loss, where we got values very close to zero. The gateways also supported data throughput much greater than the sensors attached to them could send: each gateway supported the connection of 8,000 buses sending data every 0.5 seconds.

We have developed a series of benchmark applications to evaluate the architecture in terms of performance, the number of possible connections, and communication delays. The results showed that the architecture scaled well concerning the number of resources in the city and data flow.

We also tried out the testbed with several real Smart-City applications:

- REGIONAlert: We developed this application to exemplify the use of each layer of MUSANet architecture. This application is an example of a Collaborative Distributed Processing Class application.
- Heat Island Detection: An application that makes intensive use of data processing on the edge with the help of CEP to minimize the data transfer to Fog. Other examples of CEP usage can be found in the Appendix B.
- Face Detection: The first application CPU- and I/O-bound application we developed. This application is an example of a Migratable Distributed Processing Class application. Appendix B.2 shows an example of how to use CEP in this application to detect crowd.
- Automatic License Plate Recognition: As an example of a Multitier Distributed Processing Class application, we modified the Face Detection application to detect license plates and combine it with the ALPR algorithm to speed up processing while minimizing communication overload. Appendix B.3 describes how to use CEP to detect vehicles that exceed the speed limit on a path.

6.2 Contribution

The main contributions of this work are: (1) the creation of a free three-tier testbed for smart-city applications and (2) the development of a design and implementation taxonomy for applications that share the same environment without interference between them.

To the best of our understanding, no testbed based entirely on free software provides tools to support the architect and the developer to analyze the performance of their applications and at the same time have scalability and support for deploying applications in real environments.

During application development and testing, we identified three types of process allocation patterns at the layers: collaborative distributed processing, migratable distributed processing, and multitier distributed processing. These patterns are helpful when determining the process distribution across MUSANet's tiers during the designing phase.

The developed middleware can be used from the application design phase as it contains several tools to support the developer. As we create the entire middleware base using free software, mostly open-source, the developer can, if necessary, include new debugging and project analysis tools.

Because it is scalable, elastic, and consumes resources compatible with those available in each tier, MUSANet is also suitable for the final implementation of applications. Using the same development, testing, and deployment environment represents a great advantage for the developer and a differential for MUSANet. Once the system under development has been tested and approved through the MUSANet testbed, it is ready to be deployed without major modifications to its programming.

During the development of this thesis, the research work generated the following publications:

- A. MESLIN; N. RODRIGUEZ; M. ENDLER. **A Scalable Multilayer Middleware for Distributed Monitoring and Complex Event Processing for Smart Cities**. In: 2018 IEEE International Smart Cities Conference (ISC2), p. 1–8, Kansas City, MO, USA, Sept. 2018. IEEE. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8656961
- A. MESLIN; N. RODRIGUEZ; M. ENDLER. **Scalable Mobile Sensing for Smart Cities: The MUSANet Experience**. IEEE Internet of Things Journal, 7(6):5202–5209, Jun. 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9018282

– A. MESLIN; N. RODRIGUEZ; M. ENDLER. **Supporting Multiple Smart-City Applications based on MUSANet, a Common IoMT Middleware**. In: WORKSHOP EM Clouds E Aplicações, p. 14, Rio de Janeiro, Brazil, Dec. 2020. Sociedade Brasileira de Computação. [Online]. Available: https://sol.sbc.org.br/index.php/wcga/article/view/12441

As part of the development of this work, we gave a three-week course on MUSANet at the Federal University of Maranhão, São Luis, in July and August 2019.

6.3 Revising Proposal and Research Questions

In this section, we will summarize how we approached the problems presented in Section 1.1, listed below:

Problem Statement I:

The middleware for a large smart city must be scalable, layered- and geo-distributed with support to mobile devices.

Problem Statement II:

Programmers need a test environment that emulates several real environments, varying characteristics such as bandwidth, number of nodes in each layer, data loss rate, delays, disconnections, etc., so that they can know the behavior of their applications during the development process.

To resolve Problem Statement I, we answered research questions RQ1 and RQ3, listed below:

Research Question I:

How to provide an architecture that helps investigate the best tier/layer for processing Smart-City information?

Research Question III:

How to provide a distributed scalable smart city infrastructure with support for mobile sensing, with a convenient roaming model?

Our answer to research questions RQ1 and RQ3 was developing a three-tier middleware, Cloud, Fog, and Edge, for Smart Cities. All tiers have processing capability, allowing for distributing application tasks for Smart Cities in their

tiers and within each tier. To validate the architecture in terms of scalability, we have developed several applications to stress its various levels. All three tiers of MUSANet natively support mobile devices and roaming.

To solve Problem Statement II, we answered research question RQ2, listed below:

Research Question II:

At which level should IoT data be processed in a Smart City?

To answer the research question RQ2, we have implemented a series of tools and methodologies for monitoring, emulating, and capturing data. The use of these tools has helped us to understand better the behavior of the applications we developed. That is, the use of appropriated tools can help developers to distribute the tasks of their applications in the tiers of MUSANet.

6.4 Future Work

In this thesis, we present MUSANET, a three tier middleware for smart cities. The investigations we carried out were implemented using InterSCity, ContextNet version, and Mobile-Hub. InterSCity was deployed with all of its microservices running in a data center located in the cloud. The implementation of InterSCity in the cloud, although highly functional and scalable, could provide a better result, reducing the delay in communicating with ContextNet's Processing Nodes if some of its microservices that are more related to obtaining data from sensors and sending commands to actuators were deployed within the ContextNet slices. Examples of such microservices would be the Resource Adapter, the Data Collector, and the Actuator Controler.

The ContextNet version 2.7 that we used in this thesis' experiments uses DDS as a communication channel between the ContextNet modules. Currently, version 3.0 of ContextNet is in the testing phase and uses Kafka to replace DDS. We believe that it would be interesting to investigate the use of Kafka also externally for communication with the Mobile-Hub. This substitution could make ContextNet and CDDL [141, 142], a version of Mobile-Hub that uses MQTT instead of MR-UDP, compatible.

Although the three layers of MUSANet, as presented in this thesis, make up a fully functional middleware, integration with other services could make the architecture more generally applicable. Examples of other services would be the inclusion of artificial intelligence, serverless, and blockchain modules.

The Mobile Hub, through its library, has access to the power conditions of the Android smartphone. However, if the mobile device is based on another

platform, such as Raspberry Pi, there is still no provision for consulting the battery level or how the device is being powered. There is also no standard for most of these other kinds of mobile devices for information about their power source. Due to these limitations, in this work, we did not analyze energy consumption quantitatively. We consider of paramount importance to create a standardized tool so that the programmer can check the energy status regardless of the device used. Through this information, the programmer can programmatically change the application's behavior, saving energy when its level is considered low or critical. Energy savings can be achieved through various means [35, 143], such as reducing or turning off the radios, be it 3G/4G/5G, Wi-Fi, or Bluetooth, or even decreasing or suspending data processing, among others.

Finally, during our studies, we developed a set of applications for MUSANet to investigate its behavior in relation to connections, scalability, deployment, and distribution of computing. Investigating the behavior of MUSANet with other applications that have other characteristics or functional requirements could enhance understanding of the patterns we have identified in this work.

Bibliography

[1] PECAR, M.; PAPA, G.. **Transportation problems and their potential solutions in smart cities**. In: 2017 International Conference on Smart Systems and Technologies (SST), p. 195–199, Osijek, Oct. 2017. IEEE. (Cited on page 15.)

[2] GIL-GARCIA, J. R.; PARDO, T. A.; NAM, T.. **What makes a city smart? Identifying core components and proposing an integrative and comprehensive conceptualization**. Information Polity, 20(1):61–87, July 2015. (Cited on page 15.)

[3] MEDVEDEV, A.; FEDCHENKOV, P.; ZASLAVSKY, A.; ANAGNOSTOPOULOS, T.; KHORUZHNIKOV, S.. **Waste Management as an IoT-Enabled Service in Smart Cities**. In: Balandin, S.; Andreev, S.; Koucheryavy, Y., editors, Internet of Things, Smart Spaces, and Next Generation Networks and Systems, volumen 9247, p. 104–115. Springer International Publishing, Cham, 2015. Series Title: Lecture Notes in Computer Science. (Cited on page 15.)

[4] GUNGOR, V. C.; SAHIN, D.; KOCAK, T.; ERGUT, S.; BUCCELLA, C.; CECATI, C.; HANCKE, G. P.. **A Survey on Smart Grid Potential Applications and Communication Requirements**. IEEE Transactions on Industrial Informatics, 9(1):28–42, Feb. 2013. (Cited on page 15.)

[5] BHATTACHARYA, T. R.; BHATTACHARYA, A.; MCLELLAN, B.; TEZUKA, T.. **Sustainable smart city development framework for developing countries**. Urban Research & Practice, 13(2):180–212, Mar. 2020. (Cited on page 15.)

[6] JUNG, D.; TRAN TUAN, V.; QUOC TRAN, D.; PARK, M.; PARK, S.. **Conceptual Framework of an Intelligent Decision Support System for Smart City Disaster Management**. Applied Sciences, 10(2):666, Jan. 2020. (Cited on page 15.)

[7] BATTY, M.; AXHAUSEN, K. W.; GIANNOTTI, F.; POZDNOUKHOV, A.; BAZZANI, A.; WACHOWICZ, M.; OUZOUNIS, G.; PORTUGALI, Y..

Smart cities of the future. The European Physical Journal Special Topics, 214(1):481–518, Nov. 2012. (Cited on page 15.)

[8] SILVA, B. N.; KHAN, M.; HAN, K.. **Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities**. Sustainable Cities and Society, 38:697–713, Apr. 2018. (Cited on page 16.)

[9] LIM, C.; KIM, K.-J.; MAGLIO, P. P.. **Smart cities with big data: Reference models, challenges, and considerations**. Cities, 82:86–99, Dec. 2018. (Cited on page 16.)

[10] SANCHEZ, L.; MUÑOZ, L.; GALACHE, J. A.; SOTRES, P.; SANTANA, J. R.; GUTIERREZ, V.; RAMDHANY, R.; GLUHAK, A.; KRCO, S.; THEODORIDIS, E.; PFISTERER, D.. **SmartSantander: IoT experimentation over a smart city testbed**. Computer Networks, 61:217–238, Mar. 2014. (Cited on pages 16 and 26.)

[11] VAJDA, V.; FURDÍK, K.; GLOVA, J.; SABOL, T.. **The EBBITS Project: An Interoperability platform for a Real-world populated Internet of Things domain**. In: Proceedings of the International Conference Znalosti (Knowledge), p. 5, Ostrava, Czech Republic, 2011. Technical University of Ostrava, Czech Republic. (Cited on page 16.)

[12] BONOMI, F.; MILITO, R.; ZHU, J.; ADDEPALLI, S.. **Fog computing and its role in the internet of things**. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12, p. 13, Helsinki, Finland, 2012. ACM Press. (Cited on page 17.)

[13] CHEN, Y.; FARLEY, T.; YE, N.. **QoS Requirements of Network Applications on the Internet**. Information Knowledge Systems Management, 4(1):22, 2004. (Cited on page 17.)

[14] KARLEKAR, J.; ZHOU, S. Z.; LU, W.; LOH, Z. C.; NAKAYAMA, Y.; HII, D.. **Positioning, tracking and mapping for outdoor augmentation**. In: 2010 IEEE International Symposium on Mixed and Augmented Reality, p. 175–184, Seoul, Korea (South), Oct. 2010. IEEE. (Cited on page 17.)

[15] RIZWAN, P.; SURESH, K.; BABU, M. R.. **Real-time smart traffic management system for smart cities by using Internet of Things and big data**. In: 2016 International Conference on Emerging Technological Trends (ICETT), p. 1–7, Kollam, India, Oct. 2016. IEEE. (Cited on page 17.)

[16] KUMARI, A.; TANWAR, S.; TYAGI, S.; KUMAR, N.. **Fog computing for Healthcare 4.0 environment: Opportunities and challenges**. Computers & Electrical Engineering, 72:1–13, Nov. 2018. (Cited on page 17.)

[17] ALAZAWI, Z.; ALANI, O.; ABDLJABAR, M. B.; ALTOWAIJRI, S.; MEHMOOD, R.. **A smart disaster management system for future cities**. In: Proceedings of the 2014 ACM international workshop on Wireless and mobile technologies for smart cities - WiMobCity '14, p. 1–10, Philadelphia, Pennsylvania, USA, 2014. ACM Press. (Cited on page 17.)

[18] JIANG ZHU; CHAN, D. S.; PRABHU, M. S.; NATARAJAN, P.; HAO HU; BONOMI, F.. **Improving Web Sites Performance Using Edge Servers in Fog Computing Architecture**. In: 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, p. 320–323, Redwood City, Mar. 2013. IEEE. (Cited on page 17.)

[19] WILLIS, D.; DASGUPTA, A.; BANERJEE, S.. **ParaDrop: a multi-tenant platform to dynamically install third party services on wireless gateways**. In: Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture - MobiArch '14, p. 43–48, Maui, Hawaii, USA, 2014. ACM Press. (Cited on page 17.)

[20] SATYANARAYANAN, M.; SCHUSTER, R.; EBLING, M.; FETTWEIS, G.; FLINCK, H.; JOSHI, K.; SABNANI, K.. **An open ecosystem for mobile-cloud convergence**. IEEE Communications Magazine, 53(3):63–70, Mar. 2015. (Cited on page 17.)

[21] RAZZAQUE, M. A.; MILOJEVIC-JEVRIC, M.; PALADE, A.; CLARKE, S.. **Middleware for Internet of Things: A Survey**. IEEE Internet of Things Journal, 3(1):70–95, Feb. 2016. (Cited on page 17.)

[22] SANTANA, E. F. Z.; CHAVES, A. P.; GEROSA, M. A.; KON, F.; MILOJICIC, D.. **Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture**. ACM Computing Surveys, 50(6):37, 2016. (Cited on page 17.)

[23] NGU, A. H. H.; GUTIERREZ, M.; METSIS, V.; NEPAL, S.; SHENG, M. Z.. **IoT Middleware: A Survey on Issues and Enabling technologies**. IEEE Internet of Things Journal, 4(1):1–1, 2016. (Cited on page 17.)

[24] GUPTA, L.; JAIN, R.; CHAN, H. A.. **Mobile Edge Computing - an important ingredient of 5G Networks**, Mar. 2016. (Cited on page 17.)

[25] ZYRIANOFF, I.; HEIDEKER, A.; SILVA, D.; KLEINSCHMIDT, J.; SOININEN, J.-P.; SALMON CINOTTI, T.; KAMIENSKI, C.. **Architecting and Deploying IoT Smart Applications: A Performance–Oriented Approach**. Sensors, 20(1):84, Dec. 2019. (Cited on page 17.)

[26] JAVADZADEH, G.; RAHMANI, A. M.. **Fog Computing Applications in Smart Cities: A Systematic Survey**. Wireless Networks, Dec. 2019. (Cited on page 17.)

[27] DIZDAREVIĆ, J.; CARPIO, F.; JUKAN, A.; MASIP-BRUIN, X.. **A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration**. ACM Computing Surveys, 51(6):1–29, Jan. 2019. (Cited on page 17.)

[28] LAMPORT, L.. **The part-time parliament**. ACM Transactions on Computer Systems (TOCS), 16(2):133–169, 1989. (Cited on page 20.)

[29] LAMPORT, L.. **Paxos made simple**. ACM Sigact News, 32(4):18–25, 2001. (Cited on page 20.)

[30] ONGARO, D.; OUSTERHOUT, J.. **In Search of an Understandable Consensus Algorithm**. In: USENIX Annual Technical Conference, p. 16, Philadelphia, Pennsylvania, USA, June 2014. USENIX. (Cited on page 20.)

[31] TALAVERA, L. E.; ENDLER, M.; VASCONCELOS, I.; VASCONCELOS, R.; CUNHA, M.; E SILVA, F. J. D. S.. **The mobile hub concept: Enabling applications for the internet of mobile things**. In: Pervasive computing and communication workshops (PerCom workshops), 2015 IEEE international conference on, p. 123–128, St. Louis, MO, USA, 2015. IEEE. (Cited on pages 23, 24, 37, and 38.)

[32] MESLIN, A.; RODRIGUEZ, N.; ENDLER, M.. **A Scalable Multilayer Middleware for Distributed Monitoring and Complex Event Processing for Smart Cities**. In: 2018 IEEE International Smart Cities Conference (ISC2), p. 1–8, Kansas City, MO, USA, Sept. 2018. IEEE. (Cited on pages 24 and 47.)

[33] MESLIN, A.; RODRIGUEZ, N.; ENDLER, M.. **Scalable Mobile Sensing for Smart Cities: The MUSANet Experience**. IEEE Internet of Things Journal, 7(6):5202–5209, June 2020. (Cited on page 24.)

[34] MESLIN, A.; RODRIGUEZ, N.; ENDLER, M.. **Supporting Multiple Smart-City Applications based on MUSANet, a Common**

IoMT Middleware. In: Workshop em Clouds e Aplicações, p. 14, Rio de Janeiro, Brazil, Dec. 2020. Sociedade Brasileira de Computação. (Cited on pages 24, 63, and 67.)

[35] PERRUCCI, G. P.; FITZEK, F. H. P.; WIDMER, J.. **Survey on Energy Consumption Entities on the Smartphone Platform**. In: 2011 IEEE 73rd Vehicular Technology Conference (VTC Spring), p. 1–6, Budapest, Hungary, May 2011. IEEE. (Cited on pages 24 and 89.)

[36] LUCKHAM, D.. **The power of events: An introduction to complex event processing in distributed enterprise systems**. Springer, Berlin, 2008. (Cited on pages 24, 32, and 37.)

[37] HIGH, P.. **The Top Five Smart Cities In The World**, Mar. 2015. https://www.forbes.com/sites/peterhigh/2015/03/09/the-top-five-smart-cities-in-the-world/. (Cited on page 26.)

[38] SÁNCHEZ, L.; GUTIÉRREZ, V.; GALACHE, J. A.; SOTRES, P.; SANTANA, J. R.; CASANUEVA, J.; MUÑOZ, L.. **SmartSantander: Experimentation and service provision in the smart city**. In: Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on, p. 1–6. IEEE, 2013. (Cited on pages 26 and 28.)

[39] BAKICI, T.; ALMIRALL, E.; WAREHAM, J.. **A smart city initiative: the case of Barcelona**. Journal of the Knowledge Economy, 4(2):135–148, 2013. (Cited on pages 27 and 56.)

[40] GEA, T.; PARADELLS, J.; LAMARCA, M.; ROLDAN, D.. **Smart Cities as an Application of Internet of Things: Experiences and Lessons Learnt in Barcelona**. In: 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, p. 552–557, Taichung, Taiwan, July 2013. IEEE. (Cited on page 27.)

[41] SINAEEPOURFARD, A.; GARCIA, J.; MASIP-BRUIN, X.; MARÍN-TORDERA, E.; CIRERA, J.; GRAU, G.; CASAUS, F.. **Estimating Smart City sensors data generation**. In: Ad Hoc Networking Workshop (Med-Hoc-Net), 2016 Mediterranean, p. 1–8, Vilanova i la Geltru, Spain, 2016. IEEE. (Cited on pages 27 and 56.)

[42] BARCELONA CITY COUNCIL. **Sentilo**, 2013 (last visited in 2021). [Online] https://www.sentilo.io/wordpress/. (Cited on page 27.)

[43] OPEN GEOSPATIAL CONSORTIUM. **The Home of Location Technology Innovation and Collaboration | OGC**, 2021 (last visited 2021). [Online] https://www.ogc.org/. (Cited on page 27.)

[44] ZANELLA, A.; BUI, N.; CASTELLANI, A.; VANGELISTA, L.; ZORZI, M.. **Internet of Things for Smart Cities**. IEEE Internet of Things Journal, 1(1):22–32, Feb. 2014. (Cited on page 27.)

[45] YIGITCANLAR, T.; KANKANAMGE, N.; VELLA, K.. **How Are Smart City Concepts and Technologies Perceived and Utilized? A Systematic Geo-Twitter Analysis of Smart Cities in Australia**. Journal of Urban Technology, p. 1–20, May 2020. (Cited on page 28.)

[46] SHELTON, T.; ZOOK, M.; WIIG, A.. **The 'actually existing smart city'**. Cambridge Journal of Regions, Economy and Society, 8(1):13–25, Mar. 2015. (Cited on page 28.)

[47] AMSTERDAM SMART CITY. **Amsterdam Smart City**, 2021 (last visited 2021). [Online] https://amsterdamsmartcity.com/. (Cited on page 28.)

[48] ANGELIDOU, M.. **Four European Smart City Strategies**. International Journal of Social Science Studies, 4(4):18–30, Mar. 2016. (Cited on page 28.)

[49] MAHIZHNAN, A.. **Smart Cities: The Singapore Case**. Cities - The International Journal of Urban Policy and Planning, 16(1):13–18, 1999. (Cited on page 28.)

[50] YAU, K.-L. A.; LAU, S. L.; CHUA, H. N.; LING, M. H.; IRANMANESH, V.; CHARIS KWAN, S. C.. **Greater Kuala Lumpur as a smart city: A case study on technology opportunities**. In: 2016 8th International Conference on Knowledge and Smart Technology (KST), p. 96–101, Chiangmai, Feb. 2016. IEEE. (Cited on page 28.)

[51] ADJIH, C.; BACCELLI, E.; FLEURY, E.; HARTER, G.; MITTON, N.; NOEL, T.; PISSARD-GIBOLLET, R.; SAINT-MARCEL, F.; SCHREINER, G.; VANDAELE, J.; WATTEYNE, T.. **FIT IoT-LAB: A large scale open experimental IoT testbed**. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), p. 459–464, Milan, Italy, Dec. 2015. IEEE. (Cited on page 28.)

[52] BELLI, L.; CIRANI, S.; DAVOLI, L.; GORRIERI, A.; MANCIN, M.; PICONE, M.; FERRARI, G.. **Design and Deployment of an IoT Application-**

Oriented Testbed. Computer, 48(9):32–40, Sept. 2015. (Cited on page 28.)

[53] ROSSETTO, S.; SILVESTRE, B.; RODRIGUEZ, N.; BRANCO, A.; KAPLAN, L.; LOPES, I.; BOING, M.; SANTANA, D.; LIMA, V.; GABRICH, R.. **CeuNaTerra: testbed para espaços inteligentes**. In: XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, p. 8, Vitória, ES, Brazil, 2015. SBRC. (Cited on page 28.)

[54] CREPALDI, R.; FRISO, S.; HARRIS, A.; MASTROGIOVANNI, M.; PETRIOLI, C.; ROSSI, M.; ZANELLA, A.; ZORZI, M.. **The Design, Deployment, and Analysis of SignetLab: A Sensor Network Testbed and Interactive Management Tool**. In: 2007 3rd International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities, p. 1–10, Lake Buena Vista, FL, USA, 2007. IEEE. (Cited on page 28.)

[55] CHATZIGIANNAKIS, I.; FISCHER, S.; KONINIS, C.; MYLONAS, G.; PFISTERER, D.. **WISEBED: An Open Large-Scale Wireless Sensor Network Testbed**. In: Komninos, N., editor, Sensor Applications, Experimentation, and Logistics, volumen 29, p. 68–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. (Cited on page 28.)

[56] PROVOOST, D. W. M.. **Demo: DingNet: A Simulator for Large-Scale†IoT Systems with Mobile Devices**. In: Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks, p. 267–269, Beijing, China, 2019. Junction Publishing. (Cited on page 28.)

[57] COUTINHO, A.; GREVE, F.; PRAZERES, C.; CARDOSO, J.. **Fogbed: A Rapid-Prototyping Emulation Environment for Fog Computing**. In: 2018 IEEE International Conference on Communications (ICC), p. 1–7, Kansas City, MO, May 2018. IEEE. (Cited on page 28.)

[58] HOQUE, M. A.; HASAN, R.. **VFbed: An Architecture for Testbed-as-a-Service for Vehicular Fog-based Systems**. In: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), p. 1–6, New Orleans, LA, USA, June 2020. IEEE. (Cited on page 28.)

[59] XU, Q.; ZHANG, J.. **piFogBed: A Fog Computing Testbed Based on Raspberry Pi**. In: 2019 IEEE 38th International Performance Computing

and Communications Conference (IPCCC), p. 1–8, London, United Kingdom, Oct. 2019. IEEE. (Cited on page 28.)

[60] POLLEY, J.; BLAZAKIS, D.; MCGEE, J.; DAN RUSK; BARAS, J.; KARIR, M.. **ATEMU: a fine-grained sensor network simulator**. In: 2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004., p. 145–152, Santa Clara, CA, USA, 2004. IEEE. (Cited on page 28.)

[61] TITZER, B.; LEE, D.; PALSBERG, J.. **Avrora: scalable sensor network simulation with precise timing**. In: IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005., p. 477–482, Los Angeles, CA, USA, 2005. IEEE. (Cited on page 28.)

[62] LEVIS, P.; LEE, N.; WELSH, M.; CULLER, D.. **TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications**. In: Proceedings of the 1st international conference on Embedded networked sensor systems, p. 12126–137, Los Angeles, California, USA, 2003. Association for Computing Machinery. (Cited on page 28.)

[63] DEVNET, C.. **Cisco devnet: Apis, sdks, sandbox, and community for cisco developers**, Apr. 2021 (last visited in 2021). [Online] https://developer.cisco.com/site/sandbox/. (Cited on page 28.)

[64] LEE, G.; SAAD, W.; BENNIS, M.. **An online secretary framework for fog network formation with minimal latency**. In: 2017 IEEE International Conference on Communications (ICC), p. 1–6, Paris, France, May 2017. IEEE. (Cited on page 28.)

[65] CHEN, X.; ZHANG, J.. **When D2D meets cloud: Hybrid mobile task offloadings in fog computing**. In: 2017 IEEE International Conference on Communications (ICC), p. 1–6, Paris, France, May 2017. IEEE. (Cited on page 28.)

[66] CORRADI, A.; CURATOLA, G.; FOSCHINI, L.; IANNIELLO, R.; DE ROLT, C. R.. **Smartphones as smart cities sensors: MCS scheduling in the ParticipAct project**. In: 2015 IEEE Symposium on Computers and Communication (ISCC), p. 222–228, Larnaca, July 2015. IEEE. (Cited on page 28.)

[67] CARDONE, G.; CIRRI, A.; CORRADI, A.; FOSCHINI, L.. **The participact mobile crowd sensing living lab: The testbed for smart cities**.

IEEE Communications Magazine, 52(10):78–85, Oct. 2014. (Cited on page 28.)

[68] SVITEK, M.; DOSTAL, R.; KOZHEVNIKOV, S.; JANCA, T.. **Smart City 5. 0 Testbed in Prague**. In: 2020 Smart City Symposium Prague (SCSP), p. 1–6, Prague, Czech Republic, June 2020. IEEE. (Cited on page 29.)

[69] SMART PRAGUE. **Golemio – Data platform | Smart Prague**, 2019 (last visited in 2021). [Online] https://smartprague.eu/projects/golemio-data-platform. (Cited on page 29.)

[70] ORGANICITY. **Organicity – Co-creating digital solutions to city challengesOrganicity**, 2017. http://organicity.eu/. (Cited on page 29.)

[71] SELECT4CITIES. **Competition to Create City Innovation Labs**, 2020 (last visited in 2021). [Online] https://www.select4cities.eu/. (Cited on page 29.)

[72] SERRANO, M.; ISARIS, N.; SCHAFFERS, H.; DOMINGUE, J.; BONIFACE, M.; KORAKIS, T.. **Enabling a Mobility Back-End as a Robust Service (EMBERS)**. In: Building the Future Internet through FIRE, U, p. 1–794. River Publishers, 2017. (Cited on page 29.)

[73] TANEJA, M.; DAVY, A.. **Resource Aware Placement of Data Analytics Platform in Fog Computing**. Procedia Computer Science, 97:153–156, 2016. (Cited on page 29.)

[74] PISANI, F.. **Leveraging Constrained Devices for Custom Code Execution in the Internet of Things**. PhD, Universidade Estadual de Campinas, Campinas, Brazil, 2019. (Cited on page 29.)

[75] BALA, M. I.; CHISHTI, M. A.. **Optimizing the Computational Offloading Decision in Cloud-Fog Environment**. In: 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), p. 1–5, Kottayam, India, Feb. 2020. IEEE. (Cited on page 29.)

[76] SHARKH, M. A.; KALIL, M.. **A Dynamic Algorithm for Fog Computing Data Processing Decision Optimization**. In: IEEE International Conference on Communications Workshops, p. 6, Dublin, Ireland, 2020. IEEE. (Cited on page 29.)

[77] MAHMOUD, M. M.; RODRIGUES, J. J.; SALEEM, K.; AL-MUHTADI, J.; KUMAR, N.; KOROTAEV, V.. **Towards energy-aware fog-enabled**

cloud of things for healthcare. Computers & Electrical Engineering, 67:58–69, Apr. 2018. (Cited on page 30.)

[78] NAAS, M. I.; PARVEDY, P. R.; BOUKHOBZA, J.; LEMARCHAND, L.. **iFogStor: An IoT Data Placement Strategy for Fog Infrastructure**. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), p. 97–104, Madrid, May 2017. IEEE. (Cited on page 30.)

[79] DENG, R.; LU, R.; LAI, C.; LUAN, T. H.; LIANG, H.. **Optimal Workload Allocation in Fog-Cloud Computing Towards Balanced Delay and Power Consumption**. IEEE Internet of Things Journal, p. 1–1, 2016. (Cited on page 30.)

[80] CHENG, H.; XIA, W.; YAN, F.; SHEN, L.. **Balanced Clustering and Joint Resources Allocation in Cooperative Fog Computing System**. In: 2019 IEEE Global Communications Conference (GLOBECOM), p. 1–6, Waikoloa, HI, USA, Dec. 2019. IEEE. (Cited on page 30.)

[81] WANG, P.; ZHENG, Z.; DI, B.; SONG, L.. **HetMEC: Latency-Optimal Task Assignment and Resource Allocation for Heterogeneous Multi-Layer Mobile Edge Computing**. IEEE Transactions on Wireless Communications, 18(10):4942–4956, Oct. 2019. (Cited on page 30.)

[82] VASCONCELOS, D. R.; ANDRADE, R. M. C.; SEVERINO, V.; SOUZA, J. N. D.. **Cloud, Fog, or Mist in IoT? That Is the Question**. ACM Transactions on Internet Technology, 19(2):1–20, Apr. 2019. (Cited on page 30.)

[83] CANALI, C.; LANCELLOTTI, R.. **A Fog Computing Service Placement for Smart Cities based on Genetic Algorithms**. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science, p. 81–89, Heraklion, Crete, Greece, 2019. SCITEPRESS - Science and Technology Publications. (Cited on page 30.)

[84] YOUSEFPOUR, A.; ISHIGAKI, G.; JUE, J. P.. **Fog Computing: Towards Minimizing Delay in the Internet of Things**. In: 2017 IEEE International Conference on Edge Computing (EDGE), p. 17–24, Honolulu, HI, USA, June 2017. IEEE. (Cited on page 30.)

[85] SALAMI, A.; YARI, A.. **A framework for comparing quantitative and qualitative criteria of IoT platforms**. In: 2018 4th International Conference on Web Research (ICWR), p. 34–39, Tehran, Apr. 2018. IEEE. (Cited on page 31.)

[86] AL-FUQAHA, A.; GUIZANI, M.; MOHAMMADI, M.; ALEDHARI, M.; AYYASH, M.. **Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications**. IEEE Communications Surveys & Tutorials, 17(4):2347–2376, 2015. (Cited on page 31.)

[87] AGUILAR, S.; VIDAL, R.; GOMEZ, C.. **Opportunistic Sensor Data Collection with Bluetooth Low Energy**. Sensors, 17(12):159, Jan. 2017. (Cited on page 31.)

[88] MA, Y.; ZHANG, S.; LIN, C.; LI, L.. **A Data Collection Method Based on the Region Division in Opportunistic Networks**. Applied Computational Electromagnetics Society Journal, 32(1):8, 2017. (Cited on page 31.)

[89] SAHOO, J.; CHERKAOUI, S.; HAFID, A.; SAHU, P. K.. **Dynamic Hierarchical Aggregation for Vehicular Sensing**. IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, 18(9):18, May 2017. (Cited on page 31.)

[90] BILAL, K.; KHALID, O.; ERBAD, A.; KHAN, S. U.. **Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers**. Computer Networks, 130:94–120, Jan. 2018. (Cited on page 32.)

[91] CHIANG, M.; ZHANG, T.. **Fog and IoT: An Overview of Research Opportunities**. IEEE Internet of Things Journal, 3(6):854–864, Dec. 2016. (Cited on page 32.)

[92] STOJMENOVIC, I.. **Fog computing: A cloud to the ground support for smart things and machine-to-machine networks**. In: 2014 Australasian Telecommunication Networks and Applications Conference (ATNAC), p. 117–122, Southbank, Australia, Nov. 2014. IEEE. (Cited on page 32.)

[93] MICROSOFT. **Azure IoT Edge documentation**, Apr. 2019 (last visited 2021). [Online] https://docs.microsoft.com/en-us/azure/iot-edge/. (Cited on pages 32, 46, and 47.)

[94] GOOGLE. **Cloud Computing Services | Google Cloud**, Apr. 2008. https://cloud.google.com/. (Cited on pages 32 and 47.)

[95] BRUNEO, D.; DISTEFANO, S.; LONGO, F.; MERLINO, G.; PULIAFITO, A.; D'AMICO, V.; SAPIENZA, M.; TORRISI, G.. **Stack4Things as a**

fog computing platform for Smart City applications. In: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), p. 848–853, San Francisco, CA, USA, Apr. 2016. IEEE. (Cited on pages 32 and 47.)

[96] BATISTA, D. M.; GOLDMAN, A.; HIRATA, R.; KON, F.; COSTA, F. M.; ENDLER, M.. **InterSCity: Addressing Future Internet Research Challenges for Smart Cities**. In: Network of the Future (NOF), 2016 7th International Conference on the, p. 1–6, Buzios, Brazil, 2016. IEEE. (Cited on pages 33 and 34.)

[97] KRYLOVSKIY, A.; JAHN, M.; PATTI, E.. **Designing a Smart City Internet of Things Platform with Microservice Architecture**. In: 2015 3rd International Conference on Future Internet of Things and Cloud, p. 25–30, Rome, Italy, Aug. 2015. IEEE. (Cited on pages 33, 44, and 47.)

[98] FIWARE FOUNDATION. **FIWARE | Open Source Platform for the Smart Digital Future**, 2018. https://www.fiware.org/. (Cited on pages 35, 45, and 47.)

[99] HERRERA, H.; TORRES, R.. **A Middleware for Creating Physical Mashups of Things**. In: Proceedings of the Spring School of Networks, p. 3, Pucón, Chile, 2017. (Cited on page 35.)

[100] ENDLER, M.; BAPTISTA, G.; SILVA, L.; VASCONCELOS, R.; MALCHER, M.; PANTOJA, V.; PINHEIRO, V.; VITERBO, J.. **ContextNet: context reasoning and sharing middleware for large-scale pervasive collaboration and social networking**. In: Proceedings of the Workshop on Posters and Demos Track, p. 2, Lisbon, Portugal, 2011. ACM. (Cited on page 35.)

[101] ENDLER, M.. **LAC – Laboratory for Advanced Collaboration – Scalable Pervasive and Context-aware systems**, 2018 (last visited 2021). http://www.lac.inf.puc-rio.br/. (Cited on pages 35 and 47.)

[102] SILVA, L.; ENDLER, M.; RORIZ, M.. **MR-UDP: Yet another reliable user datagram protocol, now for mobile nodes**. Monografias em Ciência da Computação, nr, 1200:06–13, 2013. (Cited on pages 36, 37, and 72.)

[103] DAVID, L.; VASCONCELOS, R.; ALVES, L.; ANDRÉ, R.; ENDLER, M.. **A DDS-based middleware for scalable tracking, communication**

and collaboration of mobile nodes. Journal of Internet Services and Applications, 4(1):16, 2013. (Cited on pages 36, 52, and 53.)

[104] OMG. **The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification Version 22**, Sept. 2014. (Cited on page 36.)

[105] PATEL, M.; HU, Y.; HÉDÉ, P.; JOUBERT, J.; THORNTON, C.; NAUGHTON, B.; RAMOS, J. R.; CHAN, C.; YOUNG, V.; TAN, S. J.; LYNCH, D.; SPRECHER, N.; MUSIOL, T.; MANZANARES, C.; RAUSCHENBACH, U.; ABETA, S.; CHEN, L.; SHIMIZU, K.; NEAL, A.; COSIMINI, P.; POLLARD, A.; KLAS, G.. **Mobile-Edge Computing - Introductory Technical White Paper**. White Paper, ETSI, Sept. 2014. (Cited on page 44.)

[106] FAZIO, M.; CELESTI, A.; MARQUEZ, F. G.; GLIKSON, A.; VILLARI, M.. **Exploiting the FIWARE cloud platform to develop a remote patient monitoring system**. In: 2015 IEEE Symposium on Computers and Communication (ISCC), p. 264–270, Larnaca, July 2015. IEEE. (Cited on page 45.)

[107] FERNÁNDEZ, P.; SANTANA, J.; ORTEGA, S.; TRUJILLO, A.; SUÁREZ, J.; DOMÍNGUEZ, C.; SANTANA, J.; SÁNCHEZ, A.. **SmartPort: A Platform for Sensor Data Monitoring in a Seaport Based on FIWARE**. Sensors, 16(3):417, Mar. 2016. (Cited on page 45.)

[108] LÓPEZ-RIQUELME, J.; PAVÓN-PULIDO, N.; NAVARRO-HELLÍN, H.; SOTO-VALLES, F.; TORRES-SÁNCHEZ, R.. **A software architecture based on FIWARE cloud for Precision Agriculture**. Agricultural Water Management, 183:123–135, Mar. 2017. (Cited on page 45.)

[109] PERERA, C.; JAYARAMAN, P. P.; ZASLAVSKY, A.; GEORGAKOPOULOS, D.; CHRISTEN, P.. **MOSDEN: An Internet of Things Middleware for Resource Constrained Mobile Devices**. In: 47th Hawaii International Conference on System Science, p. 10, Waikoloa, HI, USA, 2014. IEEE. (Cited on page 45.)

[110] BLACKSTOCK, M.; LEA, R.. **IoT interoperability: A hub-based approach**. In: 2014 International Conference on the Internet of Things (IOT), p. 79–84, Cambridge, MA, USA, Oct. 2014. IEEE. (Cited on page 45.)

[111] ALOI, G.; CALICIURI, G.; FORTINO, G.; GRAVINA, R.; PACE, P.; RUSSO, W.; SAVAGLIO, C.. **Enabling IoT interoperability through oppor-**

tunistic smartphone-based mobile gateways. Journal of Network and Computer Applications, 81:74–84, Mar. 2017. (Cited on page 45.)

[112] LEA, R.. **Hypercat: an iot interoperability specification**. Book/Report/Proceedings 69124, Lancaster University, Apr. 2014. (Cited on page 45.)

[113] VARIA, J.; MATHEW, S.. **Overview of Amazon Web Services**. Technical report, Amazon, 2014. (Cited on pages 46 and 47.)

[114] AMAZON. **Amazon Redshift - Amazon Web Services**, 2019. [Online] https://aws.amazon.com/redshift/. (Cited on page 46.)

[115] IBM. **IBM Cloud**, 2014 (last visited 2021). [Online] https://cloud.ibm.com. (Cited on pages 46 and 47.)

[116] GALACHE, J. A.; YONEZAWA, T.; GURGEN, L.; PAVIA, D.; GRELLA, M.; MAEOMICHI, H.. **ClouT: Leveraging Cloud Computing Techniques for Improving Management of Massive IoT Data**. In: 2014 IEEE 7th International Conference on Service-Oriented Computing and Applications, p. 324–327, Matsue, Japan, Nov. 2014. IEEE. (Cited on page 47.)

[117] MESLIN, A.; RODRIGUEZ, N.; ENDLER, M.. **MUSANet – Mobile Urban Sensor and Actuator Network**, 2020 (last visited 2021). [Online] https://musanet.meslin.com.br/. (Cited on pages 46, 48, 49, 53, 60, 82, and 107.)

[118] RED HAT. **Red hat virtualization**, 2021 (last visited in 2021). [Online] https://www.redhat.com/en/technologies/virtualization/enterprise-virtualization. (Cited on page 48.)

[119] TOBIAS OETIKER; DAVE RAND. **MRTG – The Multi Routing Traffic Grapher**, 2017 (last visited 2020). https://oss.oetiker.ch/mrtg/. (Cited on pages 49, 61, and 82.)

[120] CASE, J.; FEDOR, M.; SCHOFFSTALL, M.; DAVIN, J.. **RFC 1157 – Simple Network Management Protocol (SNMP)**, 1990 (last visited 2020). https://tools.ietf.org/html/rfc1157. (Cited on pages 49 and 61.)

[121] ESPOSTE, A. D. M. D.. **A scalable microservice-based open source platform for smart cities**. Mestrado em Ciência da Computação, Universidade de São Paulo, São Paulo, Sept. 2018. (Cited on page 52.)

[122] M. DEL ESPOSTE, A.; KON, F.; M. COSTA, F.; LAGO, N.. **InterSCity: A Scalable Microservice-based Open Source Platform for Smart Cities:**. In: Proceedings of the 6th International Conference on Smart Cities and Green ICT Systems, p. 35–46, Porto, Portugal, 2017. SCITEPRESS - Science and Technology Publications. (Cited on page 53.)

[123] MAHMUD, R.; SRIRAMA, S. N.; RAMAMOHANARAO, K.; BUYYA, R.. **Quality of Experience (QoE)-aware placement of applications in Fog computing environments**. Journal of Parallel and Distributed Computing, 132:190–203, Oct. 2019. (Cited on page 54.)

[124] GUPTA, H.; VAHID DASTJERDI, A.; GHOSH, S. K.; BUYYA, R.. **iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments: iFogSim: A toolkit for modeling and simulation of internet of things**. Software: Practice and Experience, 47(9):1275–1296, Sept. 2017. (Cited on page 54.)

[125] CHENG, B.; SOLMAZ, G.; CIRILLO, F.; KOVACS, E.; TERASAWA, K.; KITAZAWA, A.. **FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities**. IEEE Internet of Things Journal, 5(2):696–707, Apr. 2018. (Cited on page 55.)

[126] THE ECONOMIST. **Michael Dell plots his return to the public market - The opening Dell**, Dec. 2018. (Cited on page 60.)

[127] MORABITO, R.; COZZOLINO, V.; DING, A. Y.; BEIJAR, N.; OTT, J.. **Consolidate IoT Edge Computing with Lightweight Virtualization**. IEEE Network, 32(1):102–111, Jan. 2018. (Cited on page 60.)

[128] COMBS, G.. **Wireshark – Go Deep**, 1998 (last visited 2021). [Online] https://www.wireshark.org/. (Cited on pages 61 and 82.)

[129] ORACLE CORPORATION. **Visually monitors, troubleshoots, and profiles Java applications**, Dec. 1993 (last visited 2021). [Online] http://docs.oracle.com/javase/8/docs/technotes/tools/unix/jvisualvm.html. (Cited on pages 61 and 82.)

[130] PINHEIRO, V.; BHOWMIK, S.; LIMA, G.; ENDLER, M.; ROTHERMEL, K.. **Dscep: An infrastructure for decentralized semantic complex event processing**. In: 2020 IEEE International Conference on Big Data (IEEE BigData 2020), 2020. (Cited on page 63.)

[131] TAHA, H.. **Urban climates and heat islands: albedo, evapotranspiration, and anthropogenic heat**. Energy and Buildings, 25(2):99–103, Jan. 1997. (Cited on page 67.)

[132] TAN, J.; ZHENG, Y.; TANG, X.; GUO, C.; LI, L.; SONG, G.; ZHEN, X.; YUAN, D.; KALKSTEIN, A. J.; LI, F.; CHEN, H.. **The urban heat island and its impact on heat waves and human health in Shanghai**. International Journal of Biometeorology, 54(1):75–84, Jan. 2010. (Cited on pages 67 and 69.)

[133] DOERING, M.. **High-Resolution Large-Scale Air Pollution Monitoring: Approaches and Challenges**. In: Proceedings of the 3rd ACM international workshop on MobiArch, p. 5–10, Bethesda, Maryland, USA, June 2011. ACM. (Cited on page 67.)

[134] RESTUCCIA, F.; DAS, S. K.; PAYTON, J.. **Incentive Mechanisms for Participatory Sensing: Survey and Research Challenges**. CoRR, 12(2):1–38, Apr. 2016. (Cited on page 68.)

[135] GANTI, R.; YE, F.; LEI, H.. **Mobile crowdsensing: current state and future challenges**. IEEE Communications Magazine, 49(11):32–39, Nov. 2011. (Cited on page 68.)

[136] ROSENTHAL, J. K.. **Evaluating the impact of the urban heat island on public health: Spatial and social determinants of heat-related mortality in New York City**. PhD thesis, Columbia University, 2010. (Cited on page 69.)

[137] CULJAK, I.; ABRAM, D.; PRIBANIC, T.; DZAPO, H.; CIFREK, M.. **A brief introduction to OpenCV**. In: 2012 Proceedings of the 35th International Convention MIPRO, p. 6, Opatija, Croatia, May 2012. Croatian Society for Information and Communication Technology, Electronics and Microelectronics - MIPRO. (Cited on page 70.)

[138] OPENALPR TECHNOLOGY, I.. **Openalpr – Automatic License Plate Recognition**, 2020. https://www.openalpr.com/. (Cited on page 73.)

[139] FERNANDEZ, J. C.; TALEB, T.; GUIZANI, M.; KATO, N.. **Bandwidth Aggregation-Aware Dynamic QoS Negotiation for Real-Time Video Streaming in Next-Generation Wireless Networks**. IEEE Transactions on Multimedia, 11(6):1082–1093, Oct. 2009. (Cited on page 74.)

[140] MAGALHAES, F. B. V.. **Support for Adaptive and Distributed Deployment of CEP Continuous Queries for the IoMT**. In: Anais

do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, p. 14, Rio de Janeiro, Brazil, Dec. 2020. SBC. (Cited on page 78.)

[141] GOMES, B.; MUNIZ, L.; DA SILVA E SILVA, F.; DOS SANTOS, D.; LOPES, R.; COUTINHO, L.; CARVALHO, F.; ENDLER, M.. **A Middleware with Comprehensive Quality of Context Support for the Internet of Things Applications**. Sensors, 17(12):2853, Dec. 2017. (Cited on page 88.)

[142] GOMES, B. D. T. P.; MUNIZ, L. C. M.; DA SILVA E SILVA, F. J.; RÍOS, L. E. T.; ENDLER, M.. **A comprehensive and scalable middleware for Ambient Assisted Living based on cloud computing and Internet of Things**. Concurrency and Computation: Practice and Experience, 29(11):e4043, June 2017. (Cited on page 88.)

[143] DING, F.; XIA, F.; ZHANG, W.; ZHAO, X.; MA, C.. **Monitoring Energy Consumption of Smartphones**. In: 2011 International Conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing, p. 610–613, Dalian, China, Oct. 2011. IEEE. (Cited on page 89.)

A
UML Diagrams

A.1
MUSANet API

Figures A.1 and A.2 represent the APIs that we developed during this work to interface ContextNet's Processing Nodes with InterSCity. Complete documentation and examples of using the APIs are available on the MUSANet website [117].

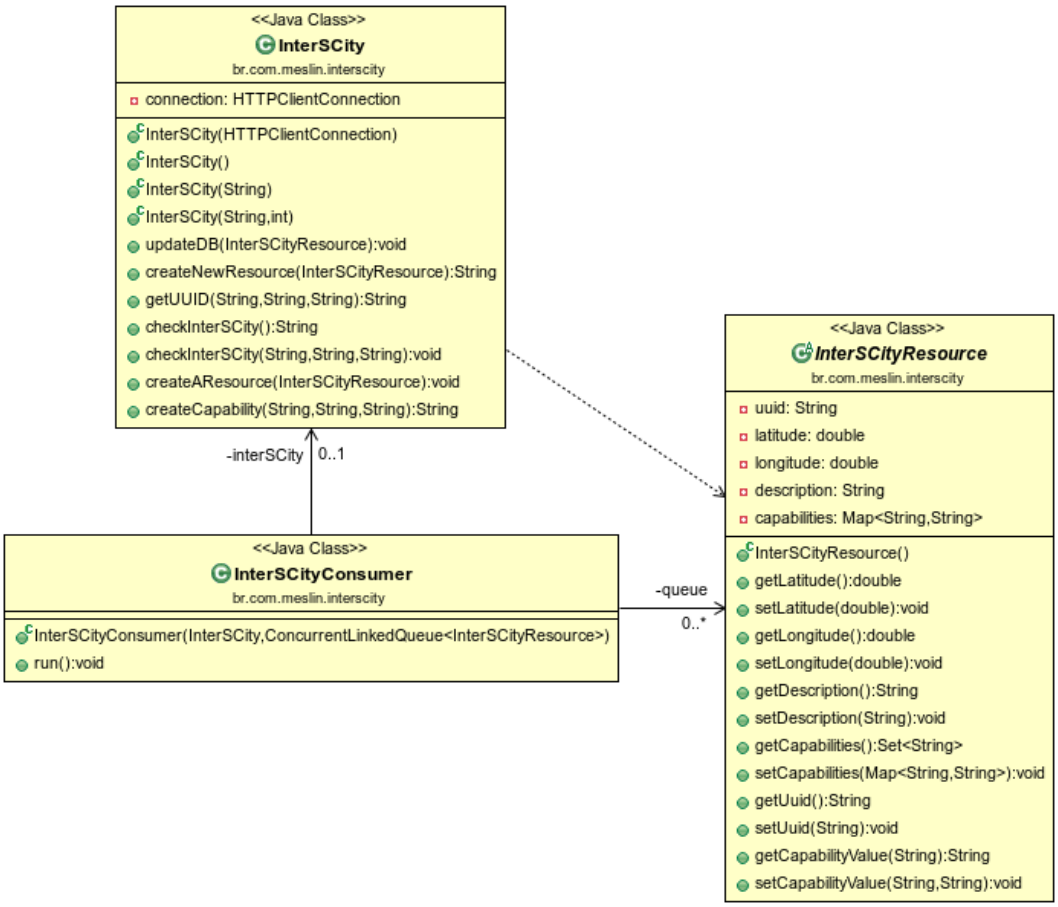


Figure A.1: API-InterSCity: InterSCity API UML class diagram.

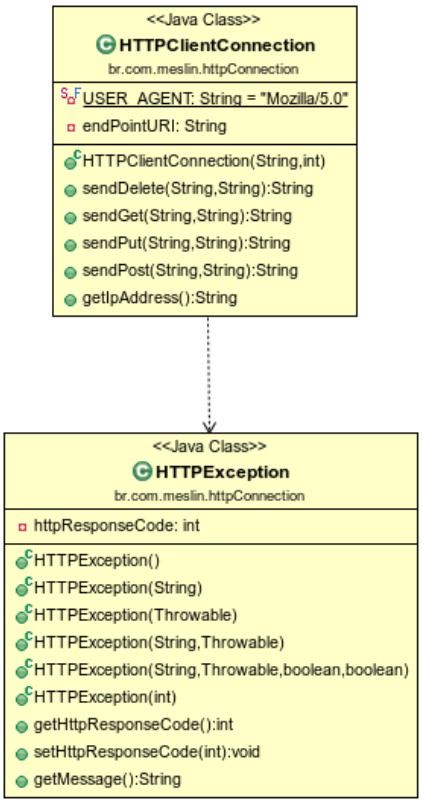


Figure A.2: API-HTTPConnection: HTTP connection API UML class diagram.

A.2 UML Sequence Diagram for REGIONAlert

This appendix presents three different scenarios to exemplify the use of the REGIONALert System.

Scenario 1: In the first scenario, no alerts were created, but there are already users registered in the system moving around the city. Please refer to Figure A.3 for more details.

Suppose one or more users enter region "R". At this point, the Mobile Hub running on the user's smartphone sends the geographic coordinates of the user to the GroupDefiner. Based on the geographical coordinates, GroupDefiner uses the city map divided into areas to find out which groups the user is in. Once the groups are determined, GroupDefiner passes this information to the ContextNet Gateway and the Processing Node. The Processing Node is responsible for checking in InterSCity whether there is any active announcement (expiration time has not yet been reached) related to those areas (groups). According to this scenario, the response will be an empty list of announcements since there are no registered alerts yet.

When necessary, the Civil Defense generates an announcement message containing the announcement text, the start timestamp, the end timestamp,

the list of areas (groups) involved, and a sequential number (automatically generated by the PutAlert Web Service).

The PutAlert stores this message at the InterSCity and publishes via the InterSCity broker a message to all Processing Nodes that subscribe to "InterestedInNotices" topic. The third-party organization's Processing Node receives this publication and seeks for the announcement in the InterSCity, receiving an announcement with a list of areas. Using this list, the Processing Nodes seeks for users at the third-party database that have marked those areas as interested and sends the announcement to all groups and all users.

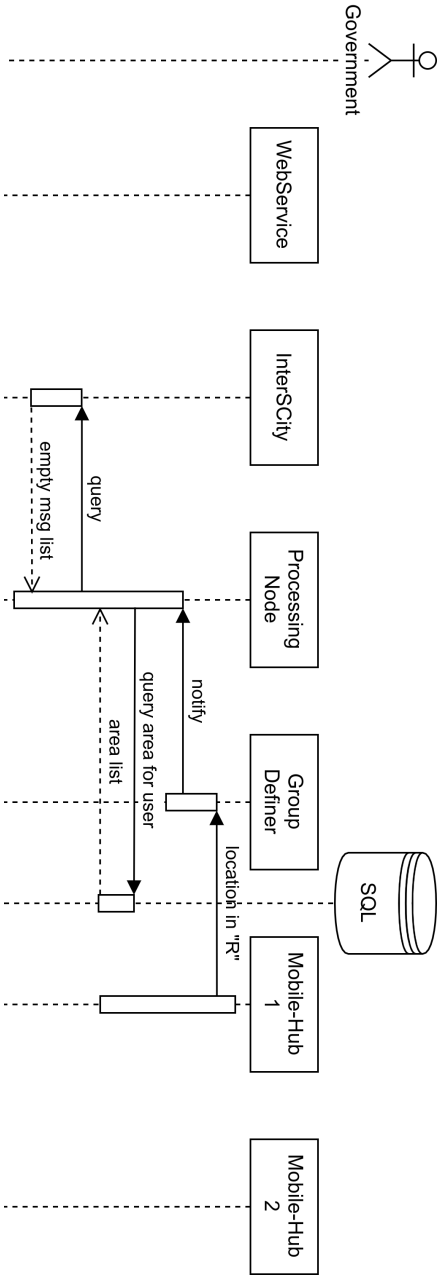


Figure A.3: Scenario 1 – no alerts: this scenario starts with no alerts and some users inside the region named "R".

Scenario 2: This scenario describes a continuation of Scenario 1, where the municipality creates a notification. Please, refer to Figure A.4 for more details.

A user enters the area "R" that already has an alert message registered according to Scenario 1. The GroupDefiner realizes that the user is now in a new area and sends a notification to the ContextNet Gateway and the Processing Node. This notification contains a list of the areas where the user is. The Processing Node, based on this list, queries the InterSCity and receives a list of valid messages for the regions where the user is. The Processing Node sends the unicast message to the user via the Gateway. In this scenario, it is not necessary to send the message to the groups because the other users in the groups have already received it, as described in Scenario 1. If any other user enters the region "R", the process will be repeated for this new user.

Scenario 3: This scenario considers again the user that we presented in Scenario 1 and Scenario 2, within an area that has an alert created and stored in InterSCity. The user has already received the announcement, as discussed in Scenario 2. Please, refer to Figure A.5 for more details.

Suppose that, after receiving the announcement, the user leaves the region but returns immediately, before the lifetime of the message is exhausted. When the user enters the zone a second time, the Mobile Hub informs the user's position and the sequential number of each received message whose lifetime has not been exhausted. The GroupDefiner does not take into account that the user has already been in the region and notifies the Gateway and the Processing Node because the GroupDefiner does not store information about the user to maintain its decoupling from the third-party applications. As in Scenario 2, the Processing Node queries InterSCity and receives a list of valid announcements for the user regions, but it deletes from this list all announcements the user has already received based on the sequential number. If, after deleting the repeated messages, any messages are still left, they will be sent by the Processing Node to the user as described in Scenario 2.

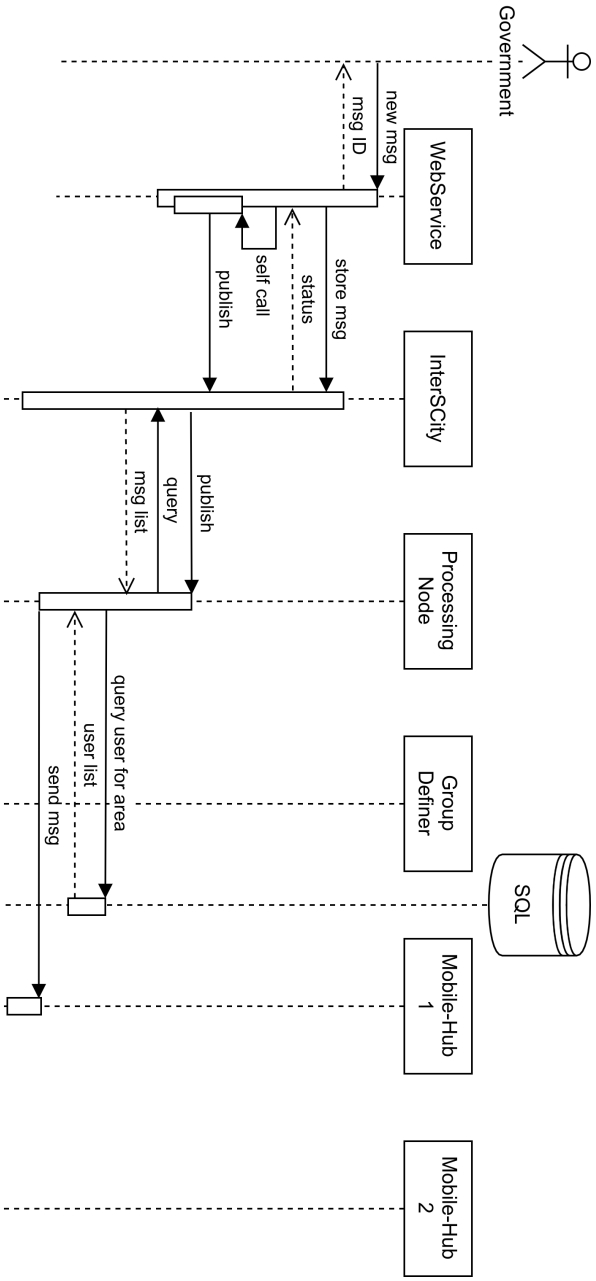


Figure A.4: Scenario 2 – alert trigged: this scenario starts with no alerts and some users inside the region named "R". The Government insert a new alert into the system.

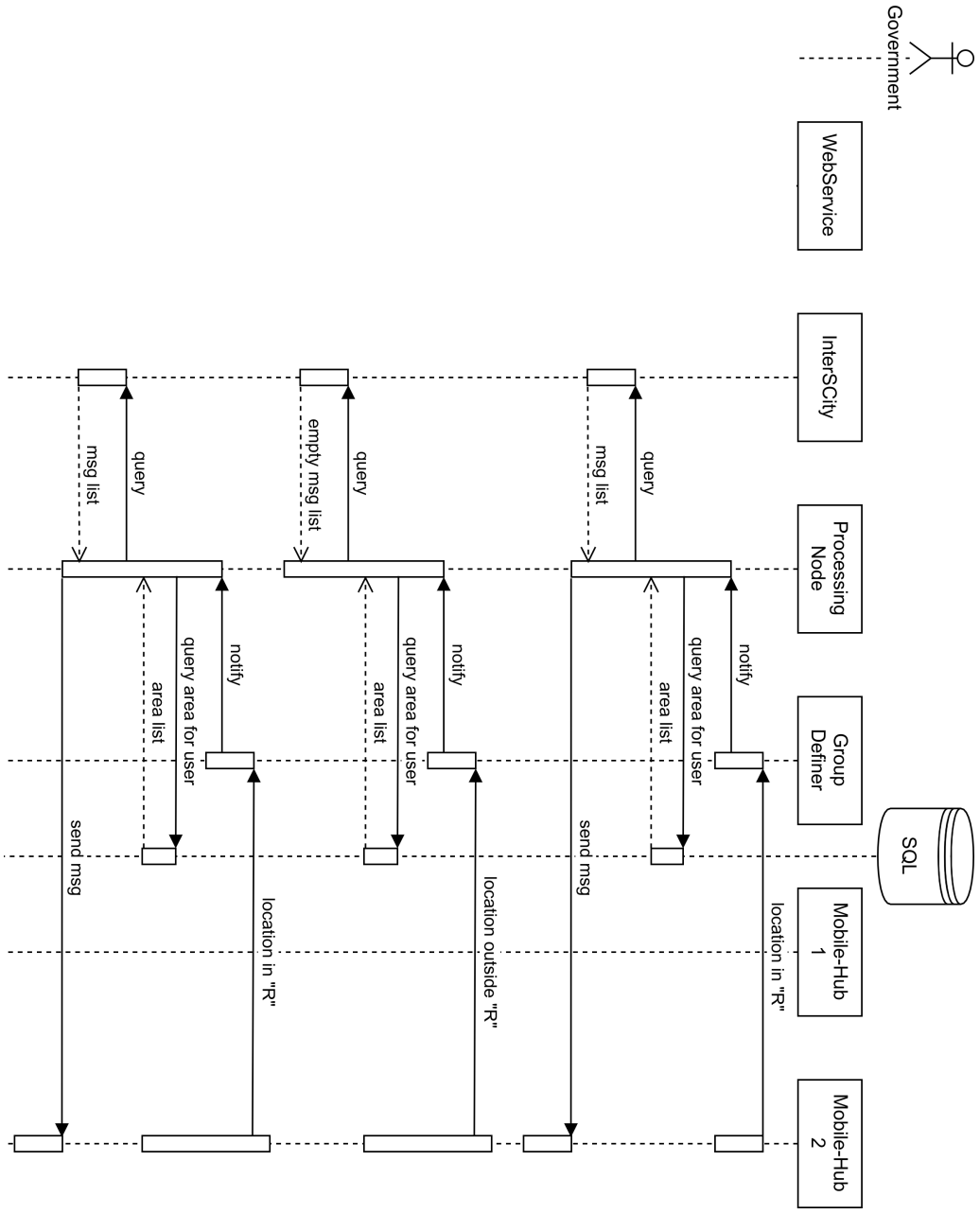


Figure A.5: Scenario 3 – user enters a region with an alert: this scenario starts with no alerts and some users inside the region named "R". The Government inserts a new alert into the system.

B
Examples of CEP Rules

B.1
CEP to Detect Heat Island Boundary

This is an example of heat island boundary detection using the last five temperatures and considering a temperature increasing of 20% on the island border. Listing B.1 presents the CEP rule, Listing B.2 shows a sample scenario, and Listing B.3 outputs the results showing that the bus enter and exit a heat island, its boundary coordinates and temperature. This rule is applied in a Mobile Hub device, however, it can be easily adapted to run on a Processing Node by including the mobile device id to separate the various flows.

Listing B.1: CEP rule to detect heat island border.

```
1 %esperepl
2 create schema Temperature(value float, latitude
     double, longitude double, date java.util.Date);
3
4 @name(EnterHeatIslandBorder) select avg(value) as
     avgTemperature, latitude, longitude, value from
     Temperature#length(4) having value > avg(value) *
     1.2;
5 @name(ExitHeatIslandBorder) select avg(value) as
     avgTemperature, latitude, longitude, value from
     Temperature#length(5) having value < avg(value) /
     1.2;
```

Listing B.2: CEP rule running to detect heat island border.

```
1 %esperscenario
2 t='2021-01-24 20:55:00.000'
3 Temperature={value=20, latitude=-22.938625, longitude
     =-43.192611, date='2021-01-24 20:55:00.000'}
4
5 t=t.plus(10 seconds);
6 Temperature={value=20, latitude=-22.948625, longitude
     =-43.202611, date='2021-01-24 20:55:10.000'}
7
8 t=t.plus(10 seconds);
9 Temperature={value=21, latitude=-22.958625, longitude
     =-43.212611, date='2021-01-24 20:55:20.000'}
```

```
10
11 t=t.plus(10 seconds);
12 Temperature={value=22, latitude=-22.968625, longitude
      =-43.222611, date='2021-01-24 20:55:30.000'}
13
14 t=t.plus(10 seconds);
15 Temperature={value=23, latitude=-22.978625, longitude
      =-43.232611, date='2021-01-24 20:55:40.000'}
16
17 t=t.plus(10 seconds);
18 Temperature={value=29, latitude=-22.988625, longitude
      =-43.242611, date='2021-01-24 20:55:50.000'}
19
20 t=t.plus(10 seconds);
21 Temperature={value=29, latitude=-22.998625, longitude
      =-43.252611, date='2021-01-24 20:56:00.000'}
22
23 t=t.plus(10 seconds);
24 Temperature={value=29, latitude=-23.008625, longitude
      =-43.262611, date='2021-01-24 20:56:10.000'}
25
26 t=t.plus(10 seconds);
27 Temperature={value=29, latitude=-23.018625, longitude
      =-43.272611, date='2021-01-24 20:56:20.000'}
28
29 t=t.plus(10 seconds);
30 Temperature={value=29, latitude=-23.028625, longitude
      =-43.282611, date='2021-01-24 20:56:30.000'}
31
32 t=t.plus(10 seconds);
33 Temperature={value=23, latitude=-23.048625, longitude
      =-43.292611, date='2021-01-24 20:56:40.000'}
```

Listing B.3: Heat island border detected. These outputs show when the bus enter and exit the heat island.

```
1 TimeStatementStreamOutput Event
2 2021-01-24 20:55:50.000
3  EnterHeatIslandBorder
4   Insert
5    EnterHeatIslandBorder-output={avgTtemperature
        =23.75, latitude=-22.988625, longitude
        =-43.242611, value=29.0}
6 2021-01-24 20:56:40.000
7  ExitHeatIslandBorder
8   Insert
9    ExitHeatIslandBorder-output={avgTtemperature=27.8,
        latitude=-23.048625, longitude=-43.292611,
       value=23.0}
```

B.2 CEP to Detect Crowd

Listing B.4 shows a CEP rule to detect crowd. One can use this rule on all MUSANet tiers. Variables `faceArea` and `nFaces` represent the area of each face and the number of faces with that area. The area of each face are inversely proportional to the distance from the camera to the face.

Listing B.4: CEP rule to detect crowd.

```
1 %esperepl
2 create schema Face(width integer, height integer,
     date java.util.Date);
3
4 @name(Crowd) select count(*) from Face#time(1 second)
     having width * height > faceArea and count(*) >
     nFaces;
```

Listing B.5 shows a scenario where there are three faces at first, but one is far away from the others. After two seconds, the third face approaches the others, and, for proof-of-concept, we considered here that three faces together represent a crowd.

Listing B.5: CEP rule running to detect crowd.

```
1  %esperscenario
2  t='2021-01-22 11:00:00.000'
3  Face={width=100, height=100, date='2021-01-22
      11:00:00.000'};
4  Face={width=100, height=100, date='2021-01-22
      11:00:00.000'};
5  Face={width=1, height=1, date='2021-01-22
      11:00:00.000'};
6
7  t=t.plus(2 seconds);
8  Face={width=100, height=100, date='2021-01-22
      11:00:00.000'};
9  Face={width=100, height=100, date='2021-01-22
      11:00:00.000'};
10 Face={width=100, height=100, date='2021-01-22
      11:00:00.000'};
```

B.3 CEP to Detect Speed Limit Exceeded

Listing B.6 presents an example of a CEP rule for detecting vehicles that have exceeded the speed limit. The rule uses the average vehicle speed between two control point.

Listing B.6: EsperTech CEP code to detect whether a vehicle has exceeded the average speed limit between two control points.

```
1 %esperepl
2 create schema LicensePlate(plateNumber string,
     timestamp java.util.Date, measuringPoint string);
3
4 @name('Detection') select * from pattern [every p1=
     LicensePlate() -> p2=LicensePlate(p1.plateNumber=p2
     .plateNumber and p1.measuringPoint != p2.
     measuringPoint) where timer:within(10 seconds)];
```

Listing B.7 presents a scenario with two vehicles passing the control points.

Listing B.7: In this example, the first car crosses the first checkpoint at 2021-01-22 11:00:00.000. The second car crosses this checkpoint at 2021-01-22 11:00:04.000. Both cars cross the second checkpoint at 2021-01-22 11:00:10.000 and 2021-01-22 11:00:10.000, respectively. The first-car average speed is 90 km/h, and the average speed limit of the second car is 112.5 km/h. The second car triggers the CEP rule because it exceeded the speed limit.

```
1 %esperscenario
2 t='2021-01-22 11:00:00.000'
3
4 LicensePlate={plateNumber='AAA-8752', timestamp
     ='2021-01-22 11:00:00.000', measuringPoint='P1'}
5 t=t.plus(4 seconds);
6 LicensePlate={plateNumber='BBB-2578', timestamp
     ='2021-01-22 11:00:04.000', measuringPoint='P1'}
7 t=t.plus(6 seconds);
8 LicensePlate={plateNumber='AAA-8752', timestamp
     ='2021-01-22 11:00:10.000', measuringPoint='P2'}
9 t=t.plus(2 seconds);
10 LicensePlate={plateNumber='BBB-2578', timestamp
     ='2021-01-22 11:00:10.000', measuringPoint='P2'}
```