COPPE UFRJ
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

A BUS-BASED OPPORTUNISTIC SENSING NETWORK

Pedro Henrique Cruz Caminha

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia Elétrica.

Orientadores: Luís Henrique Maciel Kosmalski Costa
Rodrigo de Souza Couto

Rio de Janeiro
Outubro de 2020

A BUS-BASED OPPORTUNISTIC SENSING NETWORK

Pedro Henrique Cruz Caminha

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

Orientadores: Luís Henrique Maciel Kosmalski Costa
Rodrigo de Souza Couto

Aprovada por: Prof. Luís Henrique Maciel Kosmalski Costa
Prof. Rodrigo de Souza Couto
Prof. Marcelo Gonçalves Rubinstein
Prof. Michele Nogueira Lima
Prof. Miguel Elias Mitre Campista

RIO DE JANEIRO, RJ – BRASIL
OUTUBRO DE 2020

Caminha, Pedro Henrique Cruz

A bus-based opportunistic sensing network/Pedro Henrique Cruz Caminha. – Rio de Janeiro: UFRJ/COPPE, 2020.

XVI, 95 p.: il.; 29, 7cm.

Orientadores: Luís Henrique Maciel Kosmalski Costa
Rodrigo de Souza Couto

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2020.

Referências Bibliográficas: p. 89 – 95.

1. Internet of Things. 2. Vehicle-based Sensing. 3. Mobile Wireless Networks. I. Costa, Luís Henrique Maciel Kosmalski *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha família, meus amigos e amigas

Agradecimentos

Gostaria de agradecer a todas as pessoas que participaram, de alguma forma dessa tese. Primeiro à minha família, em especial à minha mãe Olga, meu pai Moacyr e meus irmãos Walter e Victor, por sempre terem me dado força pra seguir em frente. Seus conselhos, observações e companheirismo me ajudaram a ser quem eu sou. Agradeço também ao companheirismo da Marilu, da Tequila, do Tigrão e da Dulce, que mostram que amizade é um laço maior do que a própria humanidade.

Agradeço aos meus orientadores Luís e Rodrigo, que me ensinaram muitas coisas importantes para a tese e tantas coisas importantes que não foram para a tese. Agradeço as palavras sempre amigas pra corrigir meus erros. Também agradeço a confiança depositada em mim por diversas ocasiões, incluindo a oportunidade de lecionar uma matéria para a graduação.

Je me remercie de mes superviseurs Anne et Marcelo, qu'on supporté ma faible fluence de la langue française. Vous m'avez appris beaucoup dans un temps très petit.

Agradeço ao professor Abílio pelas importantes sugestões.

Agradeço aos meus professores e colegas do Colégio Pedro II e aos meus professores e colegas do CEFET. Em especial à professora Kátia Cilene, que me ajudou a escolher a engenharia.

Agradeço a todos os professores e colegas do laboratório GTA. Foi um prazer dividir esse espaço com pessoas de tanto valor. As lições que eu aprendi com vocês ultrapassam os limites acadêmicos.

Agradeço ao JB (*in memmoriam*), que produziu comigo minha primeira publicação.

Agradeço também ao Hugo Sadok, por me ajudar a fazer as perguntas certas quando eu tinha as dúvidas erradas.

Agradeço ao Fernando Molano pelo intercâmbio de ideias sobre LoRa.

Agradeço ao Roberto Gonçalves e ao Felipe da Silva por terem participado do projeto que deu origem a essa tese. O trabalho não teria sido o mesmo sem vocês.

Agradeço aos meus colegas de laboratório no meu curto período no LIP6. Em especial, ao Giovanni Farina, por dividir a sala comigo.

Agradeço ao Pré-Universitário Comunitário Rubem Alves, por ter me propor-

cionado tanto aprendizado enquanto ensinava. Agradeço também por ter me feito conhecer algumas pessoas tão maravilhosas. Agradeço à equipe, à professora Geórgia Atella por coordenar o curso, e aos alunos e alunas por me ensinarem tanto.

Agradeço aos meus amigos Bruno Maia e Juliana Loiola, que compartilham comigo o carinho por dinossauros.

Agradeço à Priscila Oliveira, por me apresentar a tantas partes do Rio que que não conhecia.

Agradeço à Raquel Franco, por ser uma amiga tão próxima, mesmo tão distante.

Agradeço aos meus amigos de intercâmbio da Maison du Brésil. Ao cavaco da Luciana Vieira, por ter chorado quando eu precisava, e ao pessoal do sextou, que me acolheu. Agradeço aos papos intermináveis na cozinha do 5ème. Agradeço à Allana, Amanda, Dioclécio, Erato, Farah, Heraldo, Jota, Mari, Rose e Thaíssa por manterem a cozinha sempre arrumada e pela belíssima participação na noëltoyage.

Je me remercie de Amandine, pour m'avoir appris beaucoup sur la France.

I thank Emma for giving me the coolest parrot I've ever seen.

Agradeço à Namibia Guevara por me ajudar a parar de pensar nesta tese quando isso foi necessário.

Agradeço ao Carlos Quinhões por me fazer pensar em guitarra e vídeo-game quando eu deveria pensar na tese.

Agradeço à Júlia Faria por me deixar confuso com perguntas sobre engenharia.

Agradeço ao Lucas Cunha, que compartilha comigo o desprezo por tanta coisa.

Agradeço à Julia Kubrusly por falar sobre o Pará.

Agradeço à Érica Santos por dividir comigo algumas frustrações da vida.

Agradeço aos meus amigos da escola, Daim, Felipe, Juliana, Karla, Tainá e Victor. Nós nos vimos crescer e espero que nos vejamos envelhecer.

Agradeço à Luciana Rodrigues por ser uma inspiração pra mais leitura e reflexão.

Agradeço à minha amiga Lucille por, na reta final, fazer o isolamento ser menos chato.

Gostaria de agradecer ao professor Marcelo Rubinstein, à professora Michele Nogueira e ao professor Miguel Campista por terem aceitado fazer parte da banca, usando seu tempo pra avaliar meu trabalho e propor sugestões.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES). Este trabalho também contou com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), da Fundação de Amparo à Pesquisa do Estado do Rio de Janeiro (FAPERJ), e da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo #15/24494-8.

Finally, I would like to thank the open-source community for sharing their work. Without their contributions, this work would be impossible. More specifically, I

would like to acknowledge Skyclick and Freepik from `https://www.flaticon.com/` for the icons used in this thesis. I would also like to acknowledge OpenStreetMaps for the maps used in this thesis (exceptions are stated in the text).

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

UMA REDE DE SENSORIAMENTO OPORTUNISTA BASEADA EM ÔNIBUS URBANOS

Pedro Henrique Cruz Caminha

Outubro/2020

Orientadores: Luís Henrique Maciel Kosmalski Costa
Rodrigo de Souza Couto

Programa: Engenharia Elétrica

Uma estratégia promissora para implementar uma rede de sensores sem fio móveis (*Mobile Wireless Sensor Network* – MWSN) é empregar ônibus urbanos como sensores. O paradigma de internet das coisas (*Internet of Things* – IoT) pode utilizar a mobilidade dos ônibus e aumentar a cobertura espacial da rede com menos sensores, em comparação com o cenário estático. Devido às limitações de dispositivos IoT, ônibus podem entregar, de maneira oportunista, dados a nós de Névoa localizados em pontos de ônibus. A Névoa pré-processa os dados e os envia para a Nuvem, que os serve às aplicações. Isso cria um compromisso, pois ônibus cobrem a cidade parcialmente, e a frequência dos ônibus - e da coleta de dados - é heterogênea pela cidade. Adicionalmente, a entrega oportunista dos dados pode criar atrasos maiores do que os tolerados pelas aplicações. Esta tese apresenta três contribuições principais. Primeiro, os atrasos devido à entrega oportunista são minimizados, a partir de um problema de localização dos nós de Névoa. Segundo, é proposta uma métrica de cobertura espacial para MWSNs baseada em ônibus e um modelo para a escolha dos ônibus que maximizam a cobertura. Uma outra métrica de cobertura também é proposta, levando em consideração restrições das aplicações servidas pela rede. Também é proposta uma métrica da contribuição de cada ônibus para a cobertura. A terceira e última contribuição é um protótipo para o Sensing-Bus, uma MWSN baseada em ônibus urbanos. Utilizam-se dados de GPS dos ônibus da cidade do Rio de Janeiro para validar as contribuições. Entre outras observações, os resultados mostram que se 16% dos pontos de ônibus forem equipados com nós de Névoa, o maior atraso de entrega na rede é de 32 min. Adicionalmente, 32 ônibus podem cobrir 40% da mesma que a frota completa, de 6.075 ônibus.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A BUS-BASED OPPORTUNISTIC SENSING NETWORK

Pedro Henrique Cruz Caminha

October/2020

Advisors: Luís Henrique Maciel Kosmalski Costa
Rodrigo de Souza Couto

Department: Electrical Engineering

Embedding sensors in urban buses is a promising strategy to deploy a city-wide Mobile Wireless Sensor Network (MWSN). The Internet of Things (IoT) paradigm can take advantage of bus mobility, achieving extended spatial coverage with fewer sensors as compared to a static setup. To overcome the limitations of IoT devices, buses can, in an opportunistic fashion, deliver data to Fog nodes located in bus stops. Fog nodes pre-process data and send it to the Cloud, which makes makes it externally available. The trade-offs are that urban buses only cover part of the city and that the frequency of the buses, and consequently of the data collection, is heterogeneous across the city. Additionally, buses may be unable to deliver the collected data on time due to the opportunistic communication to the Fog node. In this thesis, we present three main contributions. First, we propose a method to minimize delivery delays when there is a limited number of Fog nodes. In our second contribution, we propose a coverage metric to bus-based MWSNs and an optimization model to maximize coverage for a constrained number of participating buses. We also propose a more restrictive coverage metric, that takes into account the delivery delay and the measurement frequency of each sensed region, relating these metrics to different applications. We propose a metric for bus coverage contribution, showing that the importance of each bus depends on the applications the system serves. Finally, our third contribution is a prototype for SensingBus, a bus-based MWSN. We use real GPS traces from the bus fleet of Rio de Janeiro to validate our contributions. Among other remarks, our results show that if 16% of bus stops are equipped with Fog nodes, the maximum delivery delay is about 32 minutes. We also show that 32 buses can cover about 40% of the region covered by all the 6,075 buses of the fleet.

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface, p. 6
CCDF	Complementary Cumulative Distribution Function, p. 51
CDF	Cumulative Distribution Function, p. 53
DAWN	Density Adaptive routing With Node deadline awareness, p. 9
FETRANSPOR	Rio de Janeiro Federation of Passenger Transportation Companies, p. 31
GPRS	General Packet Radio Services, p. 6
GPS	Global Positioning System, p. 16
GSM	Global System for Mobile Communications, p. 6
HTTPS	Hypertext Transfer Protocol Secure, p. 74
HTTP	Hypertext Transfer Protocol, p. 73
IoT	Internet of Things, p. 1
MCLP	Maximal Covering Location Problem, p. 47
MILP	Mixed-Integer Linear Programming, p. 27
MWSN	Mobile Wireless Sensor Network, p. 1
NB-IoT	Narrow Band IoT, p. 20
OSRM	Open Source Routing Machine, p. 58
POI	Points Of Interest, p. 12
SSID	Service Set Identification, p. 73
TCP	Transmission Control Protocol, p. 73
V2I	Vehicle to Infrastructure, p. 32
WSN	Wireless Sensor Network, p. 3

Chapter 1

Introduction

Smart cities aim at improving the quality of citizens' life by providing new public services or enhancing the efficiency of the existing ones. The key idea consists of monitoring several aspects of the city and using the collected information as the fundamental input to intelligent applications, such as waste management, air quality monitoring, and noise monitoring, only to cite a few [1]. Each application has requirements about the information they use to work properly. For instance, geographical regions may have to be sensed with a minimum frequency or data has to be delivered with a maximum delay. In this scenario, the Internet of Things (IoT) paradigm can be a useful tool to collect data for each smart city application. In a nutshell, the basic principle of IoT is to add communication, processing, and sensing capabilities to everyday objects [2].

To provide effective sensing in smart cities, one has to spread IoT devices in large geographic regions. Such a strategy might prove prohibitively expensive. For example, in the case of street lighting, every light pole would be equipped with sensors and communication interfaces. Considering a big city with several thousands of light poles distributed over a large area, adding IoT devices to every light pole imply significant deployment and maintenance costs. One alternative is to consider a Mobile Wireless Sensor Network (MWSN) that covers the entire region of interest. As sensors move through the region, their individual coverage is enlarged, avoiding additional costs with IoT devices. This approach, termed as *mobile sensing* in the literature, creates a trade-off between mobility cost and the time to cover a given area [3]. Additionally, regions may not get covered at all times. Sensors gather data about each region with a given frequency, which is ultimately a consequence of sensors movement. Since applications need a region to be sensed with a minimum frequency, some regions might be visited by sensors, but not enough times to provide data for a certain application. In this sense, mobility may imply a trade-off to the spatial coverage.

Another aspect that must be addressed is data transmission from the sensors

to the final applications. If we expect a sensor to transmit data as soon as data is gathered, it needs to be permanently connected to the servers where applications run. This means that all the regions where sensors gather information must also be covered by some network. Therefore, for sensors to immediately deliver data, all the targeted region must be covered by gateways, a situation which can also lead to preposterous costs. The network coverage can be reduced by delivering data in an opportunistic fashion, a communication paradigm enabled by mobility [4]. With this strategy, sensors gather and store data until a connection to a gateway is possible. When sensors are in the communication range of a gateway, they can finally deliver data.

Opportunistic delivery also poses a trade-off that must be managed, since data is not delivered immediately. The delivery delay is the time elapsed between the moment a sensor gathers some piece of data and the time the application receives the data. Relying on opportunistic delivery can increase this interval significantly. Considering that applications have requirements regarding a maximum delivery delay, opportunistic communication can make data useless to the applications. For this reason, it is important to make sure that the delivery delay is minimal. Additionally, when reasoning whether a region is covered by the MWSN, it is important to consider if the data about this region is delivered on time.

One way of granting mobility to sensors is to embed them into buses, in line with the IoT paradigm [5]. In this scenario, data is gathered through a bus-based MWSN. The advantage of using buses to transport sensors is threefold. First, buses cover a significant area of cities. Second, the additional cost of carrying IoT devices in buses is negligible. Third, the route of a given bus line is generally the same, with reasonably regular intervals, providing predictable coverage. Moreover, bus lines usually present some itinerary overlapping, which means that one location might be on the route of several buses. Thus, several measurements of this location can be collected, fulfilling the minimum measurement frequency requirement for different applications.

Typically, IoT devices have limited processing power and storage capacity, while smart city applications collect and analyze a large amount of data. Thus, a solution to store and process the collected data is to use a cloud computing infrastructure [6]. IoT devices are responsible for sensing and actuation, while heavier processing tasks are performed in the cloud, which has more processing power and is able to gather all data collected in a distributed fashion. In a bus-based MWSN, this is important since it is not expected that buses carry around hardware capable of intensive processing or heavy storage. A major concern regarding this approach is the network traffic exchanged between devices and the cloud, which might be high. Furthermore, security is a concern, given that IoT devices may be unable to run secure protocols

to communicate with the cloud. We overcome these limitations by employing a fog computing infrastructure between IoT devices and the cloud. Fog computing is an infrastructure that provides storage, computing, and networking services. In contrast to the cloud, the fog has fewer resources to offer, but it is closer to the end devices. The role of the fog is to pre-process data before it reaches the Internet [7]. By combining these elements, Li *et al.* [8] propose a three-level architecture to develop a Cloud of Things. With the Cloud of Things, we can employ more sophisticated security protocols before the messages are sent over the Internet. Additionally, we can aggregate, filter, and compress messages in the fog, reducing the traffic sent to the cloud and network costs.

To address the problems listed before, this thesis presents three main contributions:

- A method to minimize the delivery delay by an optimal positioning of a limited number of gateways.
- A metric to define the coverage of the applications served by a bus-based MWSN.
- A prototype to test the adequacy of each node in the three-level architecture for a bus-based MWSN.

This work also introduces SensingBus, a system that uses the concept of mobile sensing in smart cities. To offer sensor mobility, SensingBus leverages the mobility of bus lines of public transportation systems. Thus, using the IoT paradigm, buses in SensingBus are equipped with sensors and a communication interface to collect and send the collected data to Fog nodes located at bus stops. The fog sends data to the Cloud, after performing preliminary processing. SensingBus applies the three-level architecture of Li *et al.* to the bus-based sensing.

We observe that even though the literature presents works to minimize delays on static [9, 10] and mobile [11] Wireless Sensor Networks (WSNs), their focus is to optimize trajectories or packet routing. Optimizing trajectories is not possible on a mobile network composed of buses that follow fixed trajectories. Also, packet routing is not significant to the case we consider in this text, in which buses deliver data directly to the Fog nodes. It is also possible to find extensive work on the coverage of vehicle-based MWSNs [12–14]. These metrics are suited for MWSNs that are constantly connected. We propose a coverage metric that uses the streets covered by buses, observing the measurement frequency and the delivery delay as requirements of the targeted applications. Regarding the last contribution, there are other bus-based MWSN prototypes in the literature [5, 13, 15, 16]. These prototypes do not

consider a Fog level on their architecture. In this work, we implement the three-level architecture and perform experiments related to processing, communication, and mobility.

The remainder of this thesis is organized as follows. Chapter 2 reviews the related work. Chapter 3 discusses the network we envision in this thesis. Chapter 4 presents a method to minimize the delivery delay and applies it to a real dataset. Chapter 5 proposes a coverage metric for bus-based MWSNs, also applying it to a real dataset. Chapter 6 presents a prototype and the results of experiments made with it. Finally, in Chapter 7 we conclude the thesis and discuss possibilities of future work.

This text is based on the publications which resulted from the thesis. Chapter 4 is based on [17–19]. Chapter 5 borrows material from [20–24]. Finally, Chapter 6 is an adaptation of [25, 26]

Chapter 2

Related Work

To position this thesis with respect to the literature, we start by describing the works which proposed vehicle-based MWSNs. We outline the differences between our prototype and other prototypes. We proceed by comparing our delivery delays analysis with other approaches that measure and minimize delays. In the sequence, we conclude the related work section comparing our coverage metric proposal with other coverage metrics found in the literature.

Sensor networks are usually composed of many Sensing nodes that sense data about the environment and send this data to one or more sinks. The sink, then, sends data to a so-called task manager node, responsible for storing, processing, and serving data to users [27]. In the case of mobile sensing, sensors move around the targeted areas, gathering data. Figure 2.1 illustrates a typical scenario of bus-based sensing.

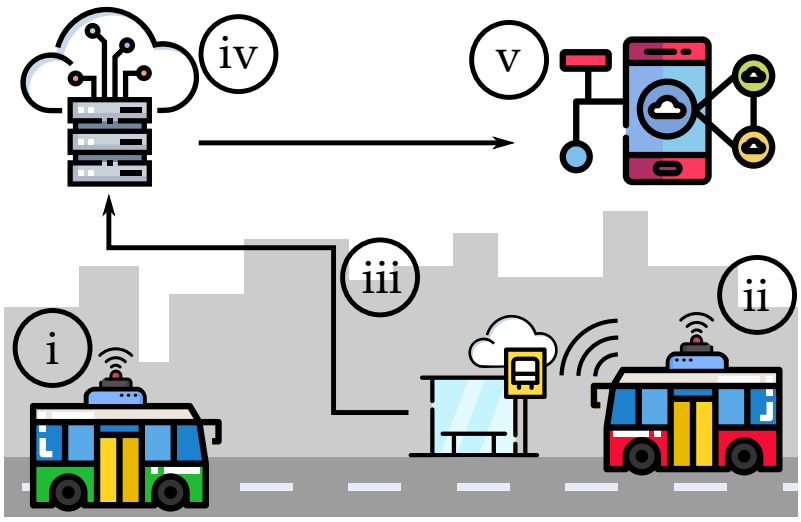


Figure 2.1: Overview of a bus-based data gathering and distribution system.

In Figure 2.1, a bus collects data throughout its trajectory (*i*), while another bus (*ii*) delivers data to a sink placed into a bus stop. The sink sends the data to

the cloud node (*iii*). The cloud node processes the data (*iv*) and serves it to the corresponding applications (*v*). In the present work, the sinks are Fog nodes that receive data and can also perform pre-processing.

The present work studies an opportunistic scenario, where buses transfer data only when a Fog node (i.e., a sink) is encountered. In the literature, it is possible to find works where sensor nodes can route messages between themselves until a sensor node can deliver data to a sink. Other works assume that the delivery network covers the whole of the targeted regions. These differences can change the expected delivery delay and also induce different communication strategies, related to the routing between vehicles. We compare our proposal to works that apply these other strategies, to delimit scenarios and conditions in which each proposal is more adequate. The next section overviews the literature on vehicle-based sensing, to distinguish the present work and the state of the art.

2.1 Vehicle-based sensing

The pioneer project BusNet [15] monitors potholes on the road surface using sensors embarked on buses. Buses are used as sensors and also as data mules. Buses sense data and store it. When a bus is in communication range to a secondary bus stop, it decides on whether to deliver the sensed data to the secondary bus stop or to receive the data previously stored on this bus stop by other buses. A bus delivers data if it is moving away from the primary bus stop, and receives the data stored in the secondary bus stop if the bus is moving towards a primary bus stop. Therefore, buses take raw data from a secondary bus stop to another one until data reaches the main bus stop. Once in the main bus stop, the information is retrieved and served to the road conditions monitoring application. This thesis evaluates the impact of a similar schema on delivery delays. Additionally, we study the coverage of the system and adapt the coverage metric for the data requirements of different applications. As the last difference, the present work considers a three-level architecture, with fog and cloud computing.

Projects Mosaic [13, 28, 29] and Opensense [16, 30] use urban buses to monitor the air quality of cities. Sensor nodes are installed in vehicles and send raw data to the cloud using a GSM (Global System for Mobile Communications)/GPRS (General Packet Radio Services) module. The cloud is responsible for processing data and delivering it to end-users through an API (Application Programming Interface. However, air quality sensors are not reliable in the presence of mobility. The sensors work measuring light scattering caused by particles. The movement of buses changes the flow inside sensors, creating measurement distortions for these sensors. Mosaic designs an algorithm to improve the accuracy of particle sensors in a mo-

bile scenario, using information about the bus speed to correct the measurements. Opensense proposes log-linear models to infer the air quality of regions that are not visited by buses and, therefore, not directly measured. In Mosaic and Opensense, buses are connected all the time with the cloud, being able to send data with negligible delay. In the present thesis, we test the accuracy of the sensors that measure humidity, temperature, light intensity, and barometric pressure. Additionally, we study an architecture with delay-tolerant delivery.

The work of Apte *et al.* [31] and the work of Von Fischer *et al.* [32] explore the predictable mobility of Google Street View cars. Apte *et al.* embark sensors in the cars to measure the air quality in the city of Oakland, proposing methods to implement similar services in other cities. Similarly to this work, Apte *et al.* use street segments of the city to define the regions that can be covered. Von Fischer *et al.* use the improved coverage provided by Google Street View cars to detect gas leaks and rapidly communicate them to maintenance teams, avoiding accidents. Gas leak sensors are not reliable in the presence of mobility. Therefore, to infer the gas leak magnitude, sensors send data to a prediction algorithm that uses the speed of the wind to correct the measurements. In their work, Apte *et al.* and Von Fischer *et al.* focus their efforts on improving the results from the available data. The present thesis focuses on strategies to improve the data gathered by the system.

Alsina *et al.* [33] design a bus-based MWSN for noise monitoring. The paper evaluates the costs and equipment requirements to implement this network. They also propose strategies to build a noise map of the city, canceling the noise from the bus carrying the sensors. The authors conclude that the application of noise monitoring can exploit the mobility of urban buses to improve its coverage, an assumption we share with them.

SmartSantander uses vehicles as part of a multi-purpose WSN to gather data about the city of Santander [5]. Other urban objects are used to sense and actuate in the city, providing services to its citizens. The work presents a framework to integrate IoT in the smart city environment and deploys a city-wide network, showing the feasibility of this system and some applications that can benefit from it, such as environmental monitoring and smart parking. In the communication paradigm used, data is delivered immediately to the applications. This means that there is no concern about delivery delays, but there is also a high cost with network infrastructure.

Dias *et al.* [34] analyze the throughput of an opportunistic network composed of public buses and show that this network is capable of transmitting enough data for many applications, including sensing. Our work considers a similar network, proposing an algorithm to allocate its sinks. We thus analyze its behavior to delimit the applications that can use buses as a mobility platform.

Table 2.1: Related work and their approach on vehicular sensing.

Reference	**Tolerance to delivery delays**	**Three-level architecture**	**Multi-application**	**Vehicle type**
[15]	Yes	No	No	Buses
[13, 28, 29]	No	No	No	Buses
[16, 30]	No	No	No	Buses
[31]	No	No	No	Google Street View cars
[32]	No	No	No	Google Street View cars
[33]	No	No	No	Buses
[5]	No	Yes	Yes	Buses, service vehicles, and taxis
Present work	Yes	Yes	Yes	Buses

Mosaic [13, 28, 29], Opensense [16, 30], Apte *et al.* [31], Von Fischer *et al.* [32], and Alsina *et al.* [33] focus their work on processing the data gathered by vehicle-based MWSNs. They aim at obtaining the best results given the gathered data. In the present work, we place our efforts in obtaining the best data possible in terms of the applications requirements. Additionally, in the prototype, we try to analyze the feasibility of the three-level architecture in terms of storage, networking, and processing capabilities. We believe that the data processing performed by these works can benefit from the fog and the cloud, enabling new applications. Table 2.1 summarizes the main differences between the prototype we develop in this thesis and the systems proposed in the literature. The columns of the table respectively point to the reference for the work, indicate if the work supports delay-tolerant data delivery, if it employs the same three-level architecture, if the work is designed to serve multiple applications, and the type of vehicle employed in the sensing tasks.

The works listed before show that vehicle-based urban sensing is a promising option to collect data for smart city applications. This thesis proposes models that can be used by such works to minimize the delivery delay or to estimate the covered area by each one of them.

2.2 Fog node placement and delivery delay

In Chapter 4, we analyze the delay induced by opportunistic delivery in a bus-based MWSN. We also propose a method to place a limited number of Fog nodes while minimizing the delivery delay. The problem is equivalent to the problem of positioning sinks or gateways to minimize the delivery delay in an MWSN.

There is a rich literature on sink positioning for static WSNs, focusing on different

quality of service metrics. Wong *et al.* propose an algorithm to decide a location for the sinks on a WSN that achieves the lowest latency possible [9]. The work shows that an optimal solution is not always feasible because of its time complexity, and proposes an approximation algorithm, capable of obtaining a satisfactory solution in feasible time. Since the time to store and forward data is usually higher than data propagation time over the network links, Wong *et al.* use the number of hops as the metric to minimize. Our work differs from [9] since we consider MWSNs where buses provide mobility to nodes, reducing the costs of deployment while keeping the covered area.

Umer *et al.* [10] propose a routing protocol that employs clustering to select sinks on a static WSN deployed in a hospital. This network is delay-sensitive and, thus, critical information employs dedicated paths. Their strategy to minimize delays through sinks selection is not applicable in our scenario. We focus on a scenario where sensors are mobile and routing between them is not possible.

In the case of mobile WSNs, it is more common to find studies that take advantage of sink mobility to enhance communication between sinks and sensor nodes. Using Hilbert curves, Ghafoor *et al.* [11] define trajectories for mobile sinks on WSNs with static sensors. A Hilbert curve defines a mapping between 1-dimensional and 2-dimensional space. Therefore, a Hilbert curve is capable of defining a path for a sink to cover the area where sensors are scattered. The order of a Hilbert curve defines the density of the mapping, consequently defining the number of times a sink passes by the same region. Their work increases the order of the Hilbert curve in areas with more Sensing nodes. These strategies were designed to operate on networks with static sensors and their use with mobile sensor nodes, such as considered in our scenario, is not investigated.

The study of Hu *et al.* makes an analysis of sink placement in MWSNs [35]. They propose a metric that, given a sensor mobility pattern, estimates the amount of data that sinks can gather. Then, they propose an optimization method to place static sinks and maximize the amount of exchanged data. Liang and Fan also propose a method to place sinks in an MWSN [36]. They formulate a multi-objective optimization problem to minimize costs while maximizing the communication coverage of sinks. In the present work, we focus on minimizing the delivery delay when placing the Fog nodes.

In MWSNs with opportunistic data delivery, contacts are not always predictable. Therefore, one challenge is to properly deliver data after it is gathered. One possible strategy to improve the probability of early delivery is to replicate through the network the messages containing sensed data. The protocol DAWN (Density Adaptive routing With Node deadline awareness) [37] makes every node share the node density of their neighborhood with their neighbors. With this information, and with a

Table 2.2: Related work and their approach on sink positioning and delivery delay.

Reference	Considers sink mobility	Considers sensor mobility	Considers optimal placement	Replication overhead
[9]	No	No	Yes	No
[10]	No	No	Yes	No
[11]	Yes	No	No	No
[35]	No	Yes	No	No
[36]	No	Yes	No	No
[37]	No	Yes	Yes	Yes
[38]	No	Yes	Yes	Yes
Present work	No	Yes	Yes	No

deadline to deliver the data, a node tries to locally decide the next hop or the next hops, in case of message replication. Of course, the cost of replicating messages can become impractical. Feng *et al.* propose the Distance-aware Replica Adaptive Data Gathering protocol [38]. This protocol aims to find a reasonable trade-off between replicating messages and delivering data timely. In the present thesis, we take the predictability of bus routes to our advantage, avoiding any replication at all.

Table 2.2 summarizes the differences between this thesis and the related work regarding sink positioning and delivery delay. The columns present the reference of the work, indicate if the work considers sink mobility, if the work considers sensor mobility, if it places sinks to find optimal delivery delay, and if there is replication overhead, respectively.

2.3 Coverage of vehicle-based MWSNs

The coverage of MWSNs is an important metric for the quality of data generated by them [39]. In the previous sections, we discussed similarities between the projects Mosaic and Opensense, but they use very different strategies to study the coverage of their proposals.

Mosaic divides the city using a grid and sets a score for every grid cell. This score depends on the number of routes that include the grid cell [13]. Additionally, Mosaic proposes an algorithm to select the best buses to receive Sensing nodes and cover points of interest spread in the city. We propose a coverage metric based on street segments and a method to select the best buses to receive Sensing nodes while maximizing coverage.

Opensense [16] divides the city into street segments and uses log-linear models to predict pollution data on segments that are not directly measured. The results obtained by Opensense show that, for urban environments, the street segmentation

is more appropriate than grid partitioning. This happens because the number of visits is not uniformly distributed inside a grid cell, but it is uniformly distributed inside a street segment. We propose a metric that uses street segmentation and estimates coverage by taking into account the number of measurements in each street segment. This makes it possible to identify places where there are not enough measurements to serve as input to the applications. The present thesis also proposes an optimization method to maximize the covered region using a limited number of buses. Another important difference is that neither Mosaic nor Opensense consider the data delivery delay. For this reason, their coverage models are not suited for opportunistic delivery when the applications have requirements on delivery delay.

Ali and Dyo propose a coverage metric for a bus-based WSN focused on road surface inspection [12]. Their proposed metric divides the city into street segments and considers a street segment as covered if at least one bus line passes through it. Ali and Dyo also propose a method to choose a limited number of bus lines while maximizing coverage. The authors test the method using a dataset containing the routes of London buses. In this thesis, we initially propose a similar approach, but we consider the lengths of the streets when calculating the coverage. The basic idea of our metric is that longer streets potentially produce more data. We also propose a method to maximize coverage when the number of participating buses is limited. Then, we propose a more general metric, that considers both delays and measurement frequency when calculating the coverage. The problem studied by Ali and Dyo can be interpreted as a particular case of the present work. Their work is a case where applications need only one measurement, have no delay restrictions and every street segment has unitary length.

Zhao *et al.* propose a coverage metric for vehicle-based WSNs that takes into account the time between measurements [14, 40]. In their work, time is discretized into slots and the area of the city is divided into a grid. A cell in the grid is covered for a given time slot if and only if a participating vehicle is inside the grid cell during the time slot. They also propose a metric called Inter-Cover Time, which is the time elapsed between two consecutive samples of the same grid cell. Another contribution of Zhao *et al.* is a metric called Opportunistic Coverage Ratio, which is the expected number of covered grid cells on a given time interval. Using these metrics, Zhao *et al.* define a notion of coverage quality and propose an algorithm to find the best vehicles to sense a certain region with a given coverage quality. In our work, we use the measurement frequency to determine whether there is enough information in a given time interval, taking advantage of the predictability of bus routes. Additionally, we add the delivery delay to our coverage metric, making sure that applications get data in a timely manner. Zhao *et al.* do not consider the delivery delay in their metrics. We also propose a method to optimally chose

Table 2.3: Related work and their approach on sensing coverage.

Reference	**Considers measurement frequency**	**Considers delivery delay**	**Covered unit**	**Multi-purpose**
[13]	Yes	No	Grid tile	No
[16, 30]	Yes	No	Street segment	No
[12]	No	No	Unweighted street segment	No
[14, 40]	Yes	No	Grid tile	Yes
[41]	No	No	Point of interest	No
[42]	Yes	No	Place	Yes
[43]	No	No	Place	Yes
Present work	Yes	Yes	Street segment	Yes

vehicles to sense a certain region, but we define regions as street segments, with a different definition of coverage.

The work by Zhang *et al.* introduces a coverage metric for Mobile Crowdsensing Networks that takes into consideration the quality of data gathered by the network [41]. In their metric, points of interest (POI) are sensed if a participating user is within the sensing range of the POI. Since POIs are weighted according to their importance, the total coverage is the sum of weights of the union of all sensed POIs. Crowdsensing coverage is also explored by Chon *et al.* [42]. They consider a region is covered if it is visited by at least one participant in the sensing tasks. Masutani [43] proposes a route control method to maximize the coverage of vehicle crowdsensing. In his work, the coverage metric is based on sensing demands. It considers a certain area covered if it is visited by at least one car during a given time window. The time window is the time the sensing task is valid. These three works do not take into account the measurement frequency nor the delivery delay of data, making them suited only for applications with no constraints regarding these metrics.

Table 2.3 shows the main aspects of the papers related to MWSN coverage. The columns respectively indicate the reference of the work, if the work takes the measurement frequency into account, whether the work takes the delivery delays into account, the approach used by the work to divide the studied region, and if the work is designed for multiple applications. The table shows that the present work, to the best of our knowledge, is the only that proposes a coverage notion considering the measurement frequency and the delivery delay of sensed data. This makes the proposed metric capable of estimating the coverage of a larger number of applications, since it considers their restrictions.

Chapter 3

Bus-based Urban Sensing

In this thesis, we consider a bus-based mobile wireless sensor network. In this chapter, we outline the assumptions we make about the scenario and context where the network is deployed. Therefore, we list the requirements of such a system and then we characterize its main aspects, defining an architecture and its elements. Since a bus-based MWSN is built to serve smart city applications, we start by describing the target applications. This description can show us the main challenges to be addressed by a bus-based MWSN.

3.1 Target applications

There is a wide range of smart city applications that can benefit from data gathered by buses. Most of them fall into the categories of smart mobility, smart environment, and smart living [44]. A few examples are pothole detection, air quality monitoring, and noise monitoring. Buses can also serve as data mules for sensors in buildings, electricity meters, smart lamps, smart trashcans, and others. For such applications to work properly, data must meet certain requirements. In the case of an MWSN with delay-tolerant data delivery, the critical requirements are related to temporal adequacy of data [39]. In this context, temporal adequacy refers to the sensing frequency of each area, and the time it takes since a bus collects data until it is available to applications and users. The survey conducted by Zanella *et al.* [1] identifies applications, the measurement frequency needed, and the delay tolerated by those applications. Sinaeepourfard *et al.* [45] also provide the measurement frequency needed for some smart city applications, but gives no information on delays.

As Table 3.1 shows, different applications have different requirements on measurement frequency and maximum delivery delay. The values are defined by Zanella *et al.* [1] and Sinaeepourfard *et al.* [45], as indicated in the table. We note that the waste management application tolerates a delay up to six times the

Table 3.1: Smart city applications and their data needs in terms of minimum measurement frequency and maximum tolerated delay.

Application	**Tolerated delay (s)**	**Measurement frequency** (day^{-1})	**Source**
Waste management	1,800	24	[1]
Air quality monitoring	300	48	[1]
Noise monitoring	300	144	[1]
Electricity meter	not defined	96	[45]
Gas meter	not defined	96	[45]
Temperature	not defined	96	[45]
Weather	not defined	48	[45]

delay tolerated by air quality monitoring and noise monitoring. This happens because the conditions of waste management infrastructure change more slowly than the conditions of air quality and noise in a city.

The column *Tolerated delay* in Table 3.1 represents the time tolerated by an application between the moment a piece of data is collected until the piece of data is delivered to the application. If data arrives later than the specified, the application is unable to use it. The column *Measurement frequency* means that a region must be visited several times a day. The number of times per day depends on the application. We do not take into account a visit for which the measurements are not delivered on time. The relationship between these two metrics is not direct. It is possible that an application needs many measurements about a certain region, but the measurements are not immediately used. Therefore, the application has strict measurement frequency requirements, but loose delivery delay requirements.

3.2 SensingBus

The models and prototypes developed in this thesis are applied in the context of SensingBus, a bus-based MWSN proposed in this thesis. SensingBus aims to provide the following functionalities to its users:

- **Data gathering:** buses working as Sensing nodes gather data along their paths;
- **Data pre-processing:** Fog nodes pre-process data before they send data over the Internet. Among other things, Fog nodes can remove inconsistent data, compress data or implement robust security protocols;
- **Data transmission:** data travels from buses to the final applications, using the Internet, the fog and the cloud infrastructure;

- **Data storage:** the cloud stores data gathered by buses;
- **Data query:** applications can query SensingBus about historical data.

To provide these functionalities, SensingBus embarks Sensing nodes into urban buses and attaches Fog nodes to bus stops. The urban buses gather data and deliver data directly to the Fog nodes, opportunistically.

3.2.1 Networking context

As pointed by Zhou *et al.*, cellular networks are reaching their operational limits due to the large amount of data exchanged by users [46]. The authors argue that data offloading through opportunistic mobile networks is one of the main strategies to circumvent this problem. In this strategy, delay-tolerant data is piggybacked on mobile nodes and delivered to gateways in their paths, instead of using traditional cellular networks [47].

In the literature, there are works that propose leveraging the mobility of buses to offload data from the cellular networks [15, 34, 46]. In this case, buses serve as data mules and offload data using gateways spread around the city. Buses carrying sensors can use the same gateways to deliver data collected throughout their paths. Another advantage of using opportunistic communication is to still be able to sense regions not covered by cellular networks, a situation that still happens in some cities.

We assume that sensors carried by buses gather data about the city and store them until a connection with a gateway is possible. We also assume that all stored data can be delivered during a single connection. We develop this assumption further in Chapter 4. Gateways, located at bus stops, receive data when buses are within communication range, and use the Internet to send this data to a cloud server. The architecture of the system, described in the next section, defines the task of each of the elements.

3.2.2 System architecture

SensingBus follows the three-level architecture for IoT proposed by Li *et al.* [8], as illustrated in Figure 3.1. In the sensing level, sensors embarked on buses gather data about the city, using the IoT paradigm. Then, sensed data is sent to the fog level. Fog nodes pre-process the data and send it to the cloud level, using the Internet. In the cloud level, data is processed, stored, and served to the final applications. We describe each level in the next sections. We discuss the implementation issues and protocols in Chapter 6.

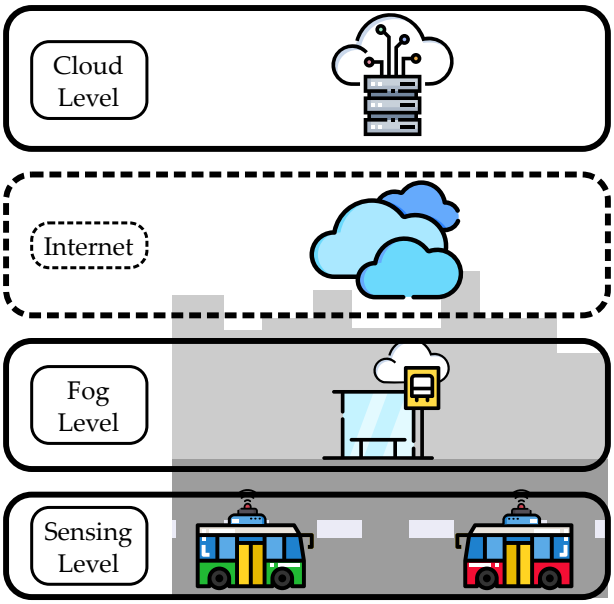


Figure 3.1: The architecture of SensingBus.

Sensing level

The Sensing level is responsible for gathering data about the city and sending it to Fog nodes. Along with the sensed data, the Sensing node registers the timestamp and geographic coordinates of each sample. This level is composed of the set of all Sensing nodes located in the buses. The high-level architecture of a Sensing node is shown in Figure 3.2.

The Controller of the Sensing node manages all tasks of the node. At a specific sampling rate, the Controller reads the coordinates and time indicated by the GPS Receiver (Global Positioning System) and also reads the measurements of every sensor in the Sensor Bank. The coordinates, time, and measurements are associated and written into the Persistent Memory. At every iteration, the Controller also queries the Wireless Interface to check whether a connection is established with a Fog node. The Sensing node connects to the Fog node using a private network, where all the other devices are trusted. Whenever a connection is established, the Controller sends all the data stored in the Persistent Memory to the Wireless Interface. This data is sent to a Fog node, and can thus be deleted from the Persistent Memory in the Sensing node. The size of the Persistent Memory must be enough to hold all data gathered between any two consecutive encounters of the Sensing node with a Fog node.

One design objective of SensingBus is to have inexpensive Sensing nodes, so even a constrained budget is enough to equip a significant amount of buses. Thus, Sensing nodes are simple and constrained in terms of computational resources. Since data must be stored by Sensing nodes before sending them to Fog nodes, Sensing nodes have persistent memory requirements that must be addressed. We study these

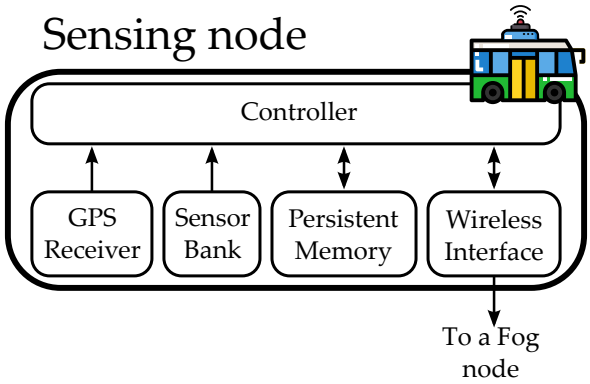


Figure 3.2: Architecture of the Sensing nodes in SensingBus.

requirements in Section 4.2.

3.2.3 Fog level

The Fog level follows the fog computing paradigm [7]. Fog nodes are located at the edge of the network. Their computational power is at an intermediate level between Sensing nodes and nodes in the Cloud level, which are cloud servers. Basically, Fog nodes receive raw data from the Sensing level, pre-process it, and send it over the Internet to the Cloud level. The pre-processing is fundamental to our system since it can provide important functionalities that improve performance and security. These functionalities include data aggregation, data compression, cryptography, and others. Figure 3.3 shows the Fog node architecture.

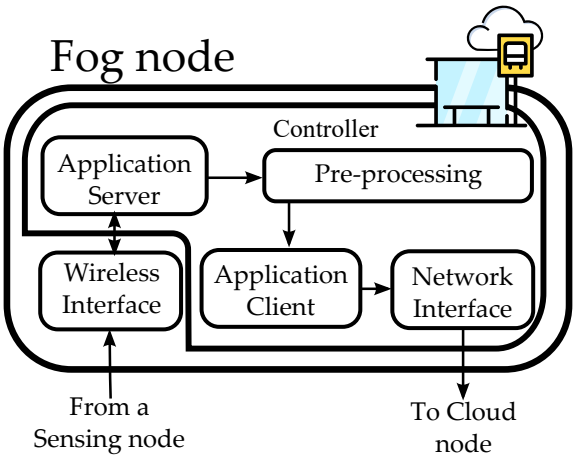


Figure 3.3: Architecture of the Fog nodes in SensingBus.

Fog nodes act as both application clients and servers, depending on the context. From the viewpoint of Sensing nodes, Fog nodes are application servers receiving data. Additionally, Sensing nodes also view Fog nodes as access points, since Fog nodes create private networks for communication between Fog and Sensing nodes. On the other hand, to the nodes in the Cloud level, Fog nodes are application clients.

They connect to the Cloud level using the Internet and forward data received from the Sensing nodes.

Communication between Sensing nodes and Fog nodes happens on a private network, therefore, a certain security level is assured. The messages between Fog nodes and Cloud nodes travel over the Internet. Therefore, it is important that Fog nodes use protocols that implement authorization and data integrity. In Chapter 6 we detail the implementation of authorization and data integrity.

3.2.4 Cloud level

The Cloud level is the final destination of the collected data. Inside it, data is stored, processed, and made available to users. There are many reasons for having these tasks performed by a cloud service, but the most important are the elasticity and the availability of the cloud. Elasticity is important because we expect intensive algorithms to run over data from time to time. Availability is important because SensingBus might act as the core of fundamental services, such as flood warnings. Figure 3.4 illustrates the architecture of the Cloud node.

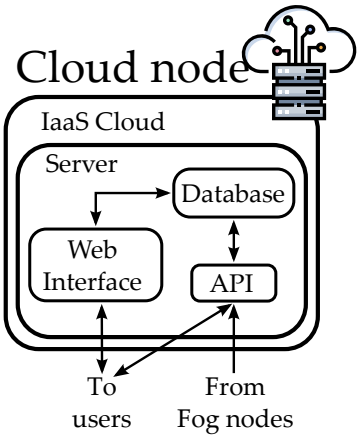


Figure 3.4: Architecture of the cloud node.

The web server of the Cloud node waits for messages from Fog nodes and data queries from users. Fog nodes can only add data to the database, whereas users can only read data from the database. The Cloud node consists of a virtual machine running on a distributed IaaS cloud. This arrangement provides elasticity, allowing the processing and storage of the Cloud node to grow and shrink, on-demand. Moreover, availability is favored by the distributed aspect of the IaaS cloud. Different sites of the distributed cloud can replicate the same Cloud node, improving the resilience of the system.

There must be some mechanism to ensure that Fog nodes know if they have their messages intercepted and even discarded before reaching the Cloud nodes. Additionally, Cloud nodes must be capable of checking the certificates presented by

Fog nodes, preventing malicious devices from inserting data into the Cloud node database. In Chapter 6, we solve this by using a Certificate Authority that certificates the Cloud nodes and all the Fog nodes.

The architecture of SensingBus is the starting point for the models and prototype developed in this work. The next chapters describe the models, the prototype, and the experiments performed with them.

Chapter 4

A Delay Optimization Model for Bus-based MWSNs

As described in Chapter 3, the Sensing node and the Fog node are one hop apart in our scenario. Additionally, we assume that a single encounter is enough to deliver all data gathered between the present encounter and the immediately previous encounter. We discuss this assumption further in Section 4.3. Regarding the wireless communication technology, we can employ IEEE 802.11p, IEEE 802.11n, NB-IoT (Narrow Band IoT) or other wireless technology. Each technology has a different range and data rates, producing different communication coverage and throughput.

When data is gathered, the Sensing node i.e., the bus waits until a connection with a Fog node is established. Next, the Sensing node sends the data to the Fog node. Finally, the Fog node sends this data to the Cloud node through the Internet. The delay from the data being gathered until it is available to the users is the time the Sensing node waits for a connection with a Fog node plus the time the Fog node takes to send it to the Cloud node. The maximum time that a Sensing node waits for a connection is equivalent to the time a bus takes to travel from a bus stop vicinity to the next bus stop vicinity, assuming both bus stops have Fog nodes installed. The time the Fog node takes to send the data to the Cloud node is the Internet latency between the Fog and the Clod nodes. Typically, the time for a bus to travel from one bus stop vicinity to the vicinity of another bus stop is of a few minutes, and the time to send data between nodes is of a few milliseconds. In this sense, we expect that the time to send the data is negligible when compared to the time waiting for a connection. Therefore, this thesis models the delay as the time between contacts, neglecting the time to send the data from the Sensing node to the Cloud node. In the next section, we formalize this definition and analyze the effect of the Fog nodes positioning on the delays.

4.1 Delays on a constrained number of Fog nodes

We assume that, given a certain budget, the number of chosen Fog nodes is smaller than the number of bus stops. Therefore, the problem is to decide which bus stops should work as Fog nodes. This section models the envisioned network and the delay added to data sensed by this network. Table 4.1 summarizes the notations used in this chapter.

Table 4.1: Notations used modeling the delivery delays.

Notation	Description	Type
$\mathcal{B}$	Buses moving around the city	Set
$\mathcal{S}$	Bus stops that are eligible to be Fog nodes	Set
$\mathcal{C}$	Pairs (b, s) for every bus b that makes contact with bus stop s	Set
$\mathcal{I}$	Bus stops that must be Fog nodes	Set
n_{budget}	Number of allowed Fog nodes	Parameter
d_{bsr}	The time elapsed from the contact of bus b with bus stop s until the contact of bus b with bus stop r, with $s, r \in \mathcal{S}$	Parameter
S_b	Sequence of bus stops that make contact with bus b, ordered by contact time	Parameter
$S_b(i)$	Function that returns the i-th element of the sequence S_b	Parameter
$T_b(i)$	Function that returns the time when bus b makes contact with the ith element of the sequence S_b	Parameter
δ_b	Sequence of delays suffered by data gathered by bus b	Parameter
$\delta_b(i)$	Function that returns the delay suffered by data gathered between the ith and the $(i+1)^{th}$ element of the sequence S_b	Parameter
$\delta_{rem}(s)$	The removal delay of bus stop s	Parameter
n_b	Number of bus stops in the path of bus b	Parameter
M_g	Data gathered between the contact of a bus with two consecutive bus stops	Parameter
M_t	Data transferred by a bus on a single contact with a bus stops	Parameter
gen_{rate}	The data generation rate by a sensor node	Parameter
tx_{rate}	The transmission rate between a bus and a bus stop	Parameter
c_{time}	The contact time between a bus and a bus stop	Parameter
Δ_{max}	Maximum possible delay between any two Fog nodes of the network	Variable
x_s	Boolean that indicates whether s is chosen as a Fog node	Variable
y_{brs}	Boolean that indicates if, for bus b, bus stop s is the next Fog node when departing from bus stop r	Variable

Let $b \in \mathcal{B}$ be a bus equipped with a Sensing node, $s \in \mathcal{S}$ a bus stop that is a Fog node candidate, S_b an ordered list in which every element $S_b(i)$ is the ith $(1 \leq i \leq n_b)$ stop s that b makes contact with, and $T_b(i)$ the instant when the bus b makes contact with the ith stop in S_b. It is possible to see the sequence S_b as the path of b through the stops with which b makes contact. When a bus $b \in \mathcal{B}$ gathers data throughout its path in S_b, such data gets delayed when the bus does not contact some Fog node. Data gathered right after b loses contact with $S_b(1)$ is delayed by $T_b(2) - T_b(1)$. Every other piece of data gathered by b on the way between $S_b(1)$

and $S_b(2)$ has a smaller delay, until b makes contact with $S_b(2)$. The same can be applied to any pair $(S_b(i), S_b(i+1))$ that happens in the path S_b. For the sake of simplicity, the expression $(T_b(i+1) - T_b(i))$ is defined as the delay between $S_b(i)$ and $S_b(i+1)$.

It is possible to define δ_b as the sequence of delays for a bus b as follows:

$$\delta_b = \{T_b(2) - T_b(1), T_b(3) - T_b(2), ..., T_b(n_b) - T_b(n_b - 1)\}. \tag{4.1}$$

Note that, in Equation 4.1, the element $\delta_b(i)$ is a function that returns the time a bus takes to go from the stop $S_b(i)$ to the stop $S_b(i+1)$. Given that a given bus b contacts n_b bus stops, $S_b(n_b)$ is the last element in the bus path, an thus element $\delta_b(n_b)$ cannot be defined. Therefore, we define the maximum delay of network, Δ_{max}, as the highest delay of every delay sequence from the buses:

$$\Delta_{max} = \max_{b \in \mathcal{B}} \left(\max_{i \in \{2,...,(n_b-1)\}} (\delta_b(i)) \right). \tag{4.2}$$

4.2 Sensing node memory requirements

Our model assumes that data is gathered by a bus b after the contact with a Fog node on its path $S_b(i)$ and it is completely delivered on the contact with the next Fog node, $S_b(i+1)$. Throughout the travel time, represented by $\delta_b(i)$, data is incrementally stored in the Persistent Memory module. Assuming that the memory module is empty when the bus leaves $S_b(i)$, the total data stored in the memory module when the bus arrives in $S_b(i+1)$ is:

$$M_g = gen_{rate} \times \delta_b(i), \tag{4.3}$$

where gen_{rate} is the data generation rate, i.e., the amount of data generated by the sensor bank and GPS Receiver module by the unit of time. The Persistent Memory must be capable of storing the maximum amount of data expected for it. Since we assume a constant gen_{rate}, the maximum amount of data is achieved when $\delta_b(i)$ is maximum. In Section 4.1, the maximum $\delta_b(i)$ is defined as Δ_{max}. In Section 4.7.1 we show that, for a network using the buses of the city of Rio de Janeiro, it is possible to achieve the value of 2 h for Δ_{max}. Additionally, using the results from Sinaeepourfard *et al.* [45], we estimate that the average amount of data generated by typical smart city sensors in 2 h is 717 B. Therefore, the amount of data to be stored in the memory module is 717 B for each sensor in the sensor bank. In Chapter 6, we build a Sensing node prototype that, in 2 h, generates 64 kB per sensor. This is explained by the fact that we employ higher sampling rates than the expected by

Sinaeepourfard *et al.* Therefore, it is also possible to estimate the generation rate of sensors operating with different sampling rates.

Given the reasoning before, we can assume that the Sensing nodes must have a Persistent Memory module capable of storing 64 kB for each sensor in the Sensor Bank.

4.3 Assuming full delivery on single contact

In this thesis, we assume that Sensing nodes are able to completely deliver the data gathered between stops $S_b(i)$ and $S_b(i+1)$ when a contact with the Fog node in $S_b(i+1)$ occurs. We refer to this as the assumption of *full delivery on a single contact*. Given that M_t is the amount of data transferred from the bus to the Fog node at the bus stop $S_b(i+1)$, our assumption holds if:

$$M_g = M_t. \tag{4.4}$$

Inside the Sensing node, when the wireless interface makes contact with a Fog node, data is transferred first from the Persistent Memory to the Controller, then from the Controller to the Wireless Interface and, finally, from the Wireless Interface to the Fog node. Each one of these transmissions might have different transmission rates, and the minimum of these rates limits the transmission as a whole. Therefore, we define tx_{rate} as the minimum of these transmission rates. The transmission between the Fog node and the Sensing node is only possible while there is contact between them. The time of contact, denoted by c_{time} is the time during which the bus is connected to a Fog node. The time c_{time} is a function of the bus speed, route, and the transmission range between a Fog node and a Sensing node. For the assumption of full delivery on a single contact to be valid, the tx_{rate} must be big enough to unload, during c_{time}, all the data generated by the sensor bank during $\delta_b(i)$. This holds true if the following inequality is satisfied, with M_g as defined in Section 4.2:

$$tx_{rate} \times c_{time} \geq M_g. \tag{4.5}$$

To estimate the tx_{rate} and c_{time} of a worst-case scenario, we used measurements and estimations from different works, respecting the worst-case scenarios when applicable. The work of Borges *et al.* [48] measured the contact time between a bus and a bus stop, using IEEE 802.11. Borges *et al.* concluded that, on average, a bus makes contact with a bus stop for 65 s if it stops at it. Additionally, it makes contact for 32 s if the bus does not stop. The work of Rubinstein *et al.* [49] measured vehicle to vehicle communication using IEEE 802.11, achieving average throughput of 1.81 MB/s throughout the interval of contact, using a relative speed of 120 km/h.

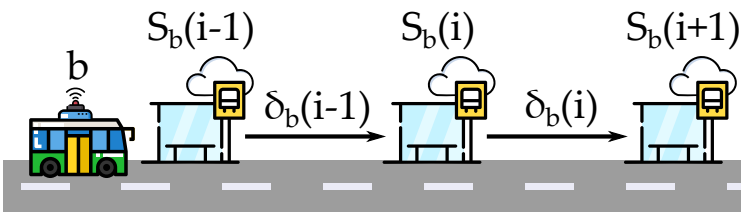
Defining c_{time} as 32 s and tx_{rate} as 1.81 MB/s, we can use Inequation 4.5 to estimate that the maximum amount of gathered data is 57.92 MB.

Using the results from Section 4.2, we can conclude that a bus that makes contact with a Fog node for 32 s, every 2 h and achieves an average throughput of 1.81 MB/s could carry more than 80,000 sensors in its sensor bank and still unload all the contents in the memory module if using the sampling rates from Sineepourfard *et al.*. In Chapter 6, we show that the Sensing node in our prototype can carry about 900 sensors and still unload all the data generated in a single contact. Therefore, it is reasonable to assume that the Sensing node can completely deliver gathered data on a single contact.

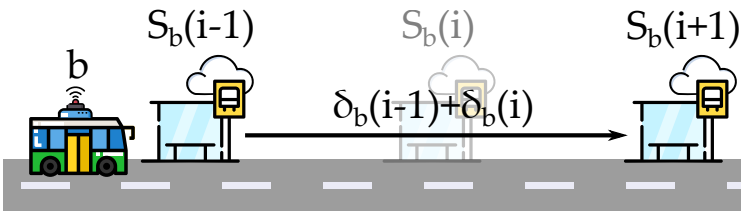
4.4 Candidate Fog node removal

In the problem, every bus stop $s \in \mathcal{S}$ is initially a Fog node candidate. We define the bus stop s as removed if s is not chosen as a Fog node. When a bus stop s is removed, the nodes cannot deliver data to it and must deliver the data to the next Fog node on their path.

The removal of s has effects on the sequence δ_b, illustrated in Figure 4.1. If a bus b has s as the element $S_b(i)$ and s is removed, b must deliver to $S_b(i+1)$ all the data that otherwise would be delivered to s, in $S_b(i)$. This means that, when $S_b(i)$ is removed, $\delta_b(i-1)$ becomes the old $\delta_b(i-1)$ plus $\delta_b(i)$. Additionally, the old $S_b(i)$ is removed from the sequence S_b, the old $\delta_b(i)$ is removed from the sequence δ_b, and every element after $S_b(i)$ and $\delta_b(i)$ is shifted one position behind.



(a) Delay before the removal of a single Fog node.

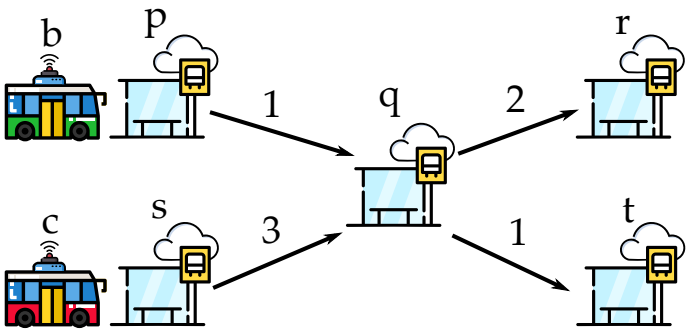


(b) Delay after the removal of a Fog node.

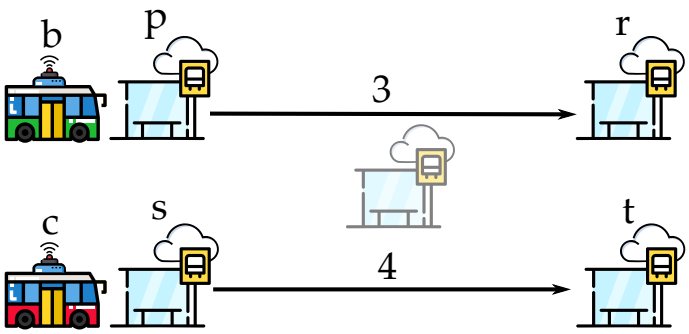
Figure 4.1: Effects in delay caused by the removal of a Fog node.

Different buses are affected differently by a Fog node candidate removal, since buses can follow different paths. Figure 4.2 illustrates the removal of the candidate

$q \in \mathcal{S}$ for two different buses, $b, c \in \mathcal{B}$. The buses b and c are moving along the bus stops $p, q, r, s, t \in \mathcal{S}$ along different paths. The sequence of bus stops for b is $S_b = (p, q, r)$ and its sequence of delays is $\delta_b = (1, 2)$. The sequence of bus stops for c is $S_c = (s, q, t)$ and $\delta_c = (3, 1)$ is its sequence of delays. When q is removed, S_b becomes (p, r), δ_b becomes (3), S_c becomes (s, t), and δ_c becomes (4).



(a) Delays before the removal of a single Fog node.



(b) Delays after the removal of a single Fog node.

Figure 4.2: Effects in delay of multiple buses caused by the removal of a Fog node.

We then define the operation of Fog node candidate removal $s \in \mathcal{S}$ as:

1. Remove s from the set $\mathcal{S}$
2. For every $b \in \mathcal{B}$:
   (a) If there is some $S_b(i) = p$, for every i:
      i. Remove of $S_b(i)$ from S_b;
      ii. Assign $\delta_b(i-1) := \delta_b(i-1) + \delta_b(i)$;
      iii. Remove $\delta_b(i)$ from δ_b.

The removal of the Fog node candidate s changes one or more delays in δ_b, if and only if s is part of S_b. We define the *removal delay* of s ($\delta_{rem}(s)$) as the maximum delay produced by the removal of s:

$$\delta_{rem}(s) = \max_{b \in \mathcal{B}} \left(\max_{i \in \{2, \ldots, (m_b - 1)\}} (\{\delta_b(i-1) + \delta_b(i) | S_b(i) = s\}) \right). \tag{4.6}$$

A Sensing node might make contact with the same Fog node more than once because the bus can drive near the same bus stop during different stages of its path. Therefore, $S_b(i)$ and $S_b(j)$ can return the same s, for $b \in \mathcal{B}$, $s \in \mathcal{S}$, and $i \neq j$. Additionally, a contact between a bus b and a Fog node on the bus stop $S_b(i)$ does not imply that the bus serves passengers on $S_b(i)$.

A last consideration about the model is the fact that it is not possible to remove some bus stops. For instance, bus stops that are the first or last bus stop in the path of some bus $b \in \mathcal{B}$ must be Fog nodes. If we remove a bus stop that is the first in the path of b, the delay calculation is invalidated; if we remove a bus stop that is the last in the path of b, data is lost, because b gathers data and is not able to deliver it. We define as $\mathcal{I}$ the set of bus stops that must be chosen as Fog nodes.

4.5 Optimal Fog node placement

In some cases, the number of possible Fog nodes is constrained by a given budget. We define as n_{budget} the number of Fog nodes available to be placed onto the bus stops in $\mathcal{S}$. As shown in Section 4.4, there is a relationship between the placement of the Fog nodes and the delays experienced by data. Therefore, it is possible to formulate an optimization problem to choose n_{budget} stops to receive Fog nodes while minimizing the maximum delay suffered by data.

We model this problem as a p-center problem, a known problem in the literature [50]. We then propose a heuristic algorithm to find suboptimal solutions. Finally, we use a suboptimal solution to speed up the algorithm to find the optimal solution.

4.5.1 The p-center problem

The p-center problem is an NP-hard problem [50]. It consists of a set $\mathcal{S}$ of facility candidates and a set $\mathcal{C}$ of demands that must be satisfied by the set of facilities. To every demand, there is a distance between this demand and the candidates that can satisfy this demand, if they are chosen to be facilities. Each demand is satisfied by the nearest facility. The problem consists of choosing p facilities while minimizing the biggest distance between any demand and the facility that satisfies the demand. Figure 4.3 illustrates an instance of the p-center problem as a bipartite graph, with $p = 2$. In this instance, $\{a, b, c\}$ is the subset of the vertices that represent the facility candidates of the problem, and $\{x, y, z\}$ is the subset of the vertices that represent the demands of the problem. A weighted edge d_{kj} starting on demand k and ending on facility j represents the distance between k and j.

To transform the problem of choosing the minimal delay Fog nodes into the p-

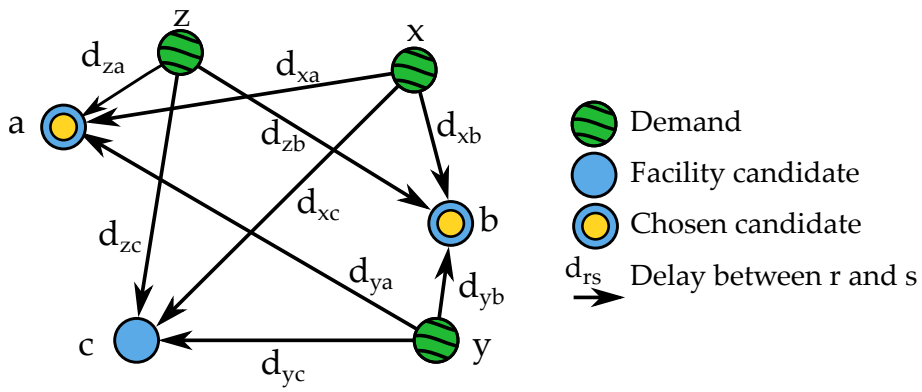


Figure 4.3: An instance of the p-center problem.

center problem, we must first remember that buses gather data, store it and deliver to Fog nodes that are located at bus stops. Therefore, every time a bus $b \in \mathcal{B}$ gathers data, a demand is created. The demand is served when b delivers this data to a Fog node. This means that, once b gathers data, each subsequent bus stop $s \in \mathcal{S}$ on its path is a candidate for serving the demand that is created. The distance between a demand and a facility candidate is the delay generated by delivering the present data to the corresponding bus stop, if we chose this bus stop to be a Fog node.

Every time a bus unloads data, the oldest data on its memory is the one with the biggest delay. With this in mind, we can argue that the demand that matters is the oldest data stored by the bus until it reaches a Fog node. This happens because any data more recent than the oldest is, by definition, not suffering the biggest delay. As discussed in Section 4.3, all data stored is delivered and then erased when b makes contact to s. Therefore, the oldest data a bus holds is the data it gathers as soon as it leaves a bus stop with a Fog node. Hence, we can consider that our demands are the encounters of a bus with a bus stop. It is possible then to define the set $\mathcal{C}$ of demands as the set containing all pairs (b, s) representing any bus $b \in \mathcal{B}$ that makes contact to $s \in \mathcal{S}$:

$$\mathcal{C} = \bigcup_{b \in \mathcal{B}} \bigcup_{i=0}^{n_b - 1} (b, S_b(i)). \tag{4.7}$$

Figure 4.4 illustrates the procedure of transforming the Fog node placement problem into the p-center problem. The bus path S_b is the sequence (w, x, y, z), where $w, x, y, z \in \mathcal{S}$. When the problem is transformed into the p-center problem, the set of demands is $\{w_b, x_b, y_b\}$ and the set of facility candidates is $\{x, y, z\}$. Each edge represents the time elapsed since the bus passed by the demand until the bus encounters a facility candidate that can satisfy the demand.

As defined in Equation 4.7, we do not consider the last bus stop encountered by a bus as a demand, because there is no data collected immediately after it. We define the optimization problem using Mixed-Integer Linear Programming (MILP):

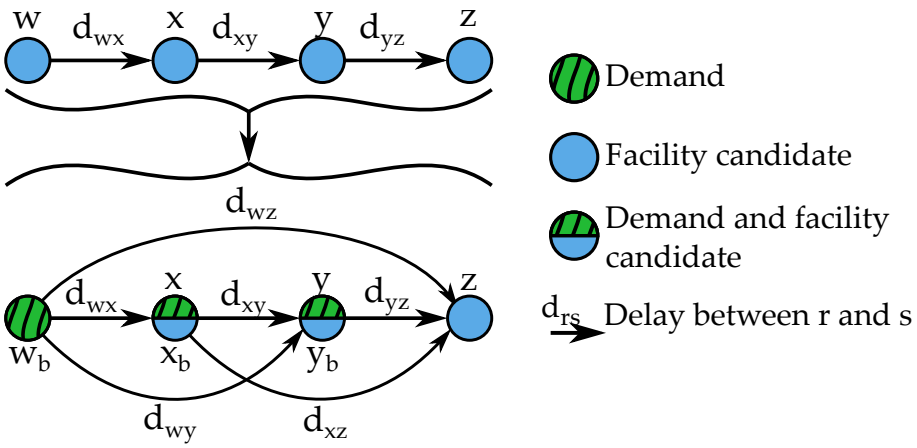


Figure 4.4: Example of the Fog node placement problem transformed into the p-center problem.

$$\text{minimize:} \quad \Delta_{\max}; \tag{4.8}$$

$$\text{s.t.:} \quad \sum_{s \in \mathcal{S}} y_{brs} = 1\,,\ \forall (b,r) \in \mathcal{C}; \tag{4.9}$$

$$y_{brs} - x_s \leq 0, \forall (b,r) \in \mathcal{C}, s \in \mathcal{S} \tag{4.10}$$

$$\sum_{s \in \mathcal{S}} x_s = n_{budget}; \tag{4.11}$$

$$\sum_{s \in \mathcal{S}} d_{brs} y_{brs} - \Delta_{\max} \leq 0\,,\ \forall (b,r) \in \mathcal{C},\ \forall r \in \mathcal{S}; \tag{4.12}$$

$$x_s = 1,\ \forall s \in \mathcal{I}; \tag{4.13}$$

$$\Delta_{\max} \in \mathbb{R}; \tag{4.14}$$

$$y_{brs} \in \{0,1\},\ \forall b \in \mathcal{B},\ \forall r, s \in \mathcal{S}; \tag{4.15}$$

$$x_s \in \{0,1\}\ \forall s \in \mathcal{S}. \tag{4.16}$$

The problem is defined the same way as the p-center problem [50]. Equation 4.8 is the objective of the problem and minimizes $\Delta_{\max}$, which is the biggest delay in the network. The binary variables x_s indicate whether the bus stop s is chosen to be a Fog node. The binary variables y_{brs} indicate if, once bus b leaves bus stop r, the next Fog on its path is encountered at bus stop s. The parameter d_{brs} represents the time it takes for b to go from stop r to stop s. Equation 4.9 is a set of restrictions to make sure that, once b leaves r, data is delivered only to one bus stop. Equation 4.10 states that, if a bus stop s is not chosen to be a Fog node, then no Sensing node

can unload data on it. Equation 4.11 limits the number of chosen bus stops to the budget defined by the problem. Equation 4.12 defines $\Delta_{\max}$ as the biggest delay among the delays experienced in the network. Equation 4.13 forces that stops in $\mathcal{I}$ are chosen. Equation 4.14, Equation 4.15, and Equation 4.16 define the limits and domains of the variables.

4.6 A fast algorithm for Fog node selection

The solution of the Fog node placement in real-life scenarios might not be practical, because of its possible processing time and memory requirements. Therefore, this work proposes a greedy solution, specified in Algorithm 1, that is initialized as if every Fog node candidate was an actual Fog node. At every iteration, the algorithm removes the Fog node candidate with the minimum removal delay, defined in Equation 4.6. The algorithm finishes when the set of Fog nodes candidates has n_{budget} elements or when it is not possible to remove more candidates, since the algorithm must choose at least the candidates in $\mathcal{I}$. In both cases, the algorithm returns the current set of Fog node candidates.

The algorithm parameters are: the set of buses $\mathcal{B}$; the set of Fog node candidates $\mathcal{S}$; the set $\mathcal{I}$ of bus stops that must chosen to be Fog nodes; the vector $\boldsymbol{paths}$, that stores the list S_b for every element $b \in \mathcal{B}$; a function $\delta_b(i)$ for every $b \in \mathcal{B}$; and n_{budget}, the amount of desired Fog nodes.

In a nutshell, Algorithm 1 stores in $\boldsymbol{max\_ts}$ removal delay $\delta_{rem}(s)$ for each stop $s \in \mathcal{S}$ (i.e., the maximum delay that appears when bus stop s is removed). The algorithm then removes the stop s with the lowest $\delta_{rem}(s)$ and such that $s \ni \mathcal{I}$. The algorithm then refreshes $\boldsymbol{max\_ts}$ with the new removal delays for the neighbors of s. This procedure is repeated until there are n_{budget} bus stops or there are no more removable stops.

4.6.1 Complexity analysis

Given the parameters of Algorithm 1, we define m as the size of the largest list in $\boldsymbol{paths}$ and n as the cardinality of $\mathcal{S}$, $|\mathcal{S}|$. If we implement the functions δ_b with lists, the largest one has size $m - 1$.

The time complexity of the algorithm is the complexity of the initial test, on Line 1, plus the complexity of the removal delay list initialization, on Line 3, plus the complexity of Fog node candidates removal, on Line 7.

The initial test on Line 1 is a cardinality comparison of the sets $\mathcal{S}$ and $\mathcal{I}$ with n_{budget}. Using O notation for worst-case scenario, the initial test complexity is $O(1)$.

The initialization of the list of removal delays, on Line 3, is an iteration over

Algorithm 1 Fog Node Placement Algorithm

Data: $\mathcal{B} = \{b_1, \ldots, b_k\}$, $\mathcal{S} = \{s_1, \ldots, s_n\}$, $\mathcal{I} = \{s_{sy}, \ldots, s_{sz}\}$, $\boldsymbol{paths} = (S_{b1}, \ldots, S_{bk})$, $\delta_{b1}, \ldots, \delta_{bk}, n_{budget}$

1: **if** $|\mathcal{S}| - |\mathcal{I}| \leq n_{budget}$ **then return** $\mathcal{I}$ ▷ It is not possible to obtain n_{budget} Fog nodes
2: $\boldsymbol{max\_ts} \leftarrow (0)$ ▷ Initialize $\boldsymbol{max\_ts}$ as a zeroes vector
3: **for** $b \in \mathcal{B}$ **do** ▷ Initialize removal delay list
4: **for** $i \leftarrow 1, |S_b|$ **do**
5: $\boldsymbol{max\_ts}[S_b[i]] \leftarrow \max(\delta_b[i-1] + \delta_b(i), \boldsymbol{max\_ts}[S_b[i]])$
6: $\mathcal{R} \leftarrow \emptyset$
7: **while** $|\mathcal{R}| + |\mathcal{I}| + n_{budget} < |\mathcal{S}|$ **do** ▷ Remove Fog node candidates until budget is achieved
8: **for** $p \in \mathcal{S}$ **do** ▷ Select a candidate with minimum removal delay
9: **if** $(d_{min} == NULL \vee d_{min} < \boldsymbol{max\_ts}[s]) \wedge s \notin I \wedge s \notin R$ **then**
10: $d_{min} \leftarrow \boldsymbol{max\_ts}[s]$
11: $s\_to\_remove \leftarrow s$
12: **for** $S_b \in \boldsymbol{paths}$ **do**
13: **for** $i \leftarrow 1, |S_b|$ **do**
14: **if** $S_b[i] = s\_to\_remove$ **then**
15: $\delta_b[i-1] \leftarrow \delta_b(i-1) + \delta_b(i)$ ▷ Refresh the delay between the previous and the next candidates
16: **if** $\boldsymbol{max\_ts}[S_b[i+1]] < \delta_b(i-1) + \delta_b(i))$ **then**
17: $\boldsymbol{max\_ts}[S_b[i+1]] \leftarrow \delta_b(i-1) + \delta_b(i))$ ▷ Refresh removal delays
18: **if** $i > 1$ **then**
19: **if** $(\boldsymbol{max\_ts}[S_b[i-1]] < \delta_b[i-2] + \delta_b(i-1))$ **then**
20: $\boldsymbol{max\_ts}[S_b[i-1] \leftarrow \delta_b[i-2] + \delta_b(i-1)$
21: Remove$(S_b[i])$ ▷ Remove i from $S_b[i]$
22: $\mathcal{R} \leftarrow \mathcal{R} \cup \{s\_to\_remove\}$ ▷ Insert the removed candidate into the set of removed candidates
23: **return** $\{\mathcal{S} \setminus \mathcal{R}\}$ ▷ Return the set of non-removed candidates

every m elements of S_b, nested in an iteration over every k elements of $\mathcal{B}$. Hence, the complexity of this part is $O(km)$.

The Fog node candidates removal has an external loop, starting on Line 7, in which $|\mathcal{R}|$ grows one unit per iteration. This is limited by the value of n_{budget}. Since n_{budget} is limited by n, the complexity of the Fog node candidates removal is the complexity of its inner loops multiplied by n. Nested to the loop on Line 7, there are two other loops, in sequence. On Line 8, a loop iterates over all n elements in $\mathcal{S}$, checking if $s \in \mathcal{I}$ and if $s \in \mathcal{R}$. We can implement these checks using a binary vector indexed by s, where a 1 in the position s means that $s \in \mathcal{S}$ or $s \in \mathcal{R}$. With this implementation, the complexity of such checks is $O(1)$. Therefore, the time complexity of loop in Line 8 is $O(n)$.

The loop in Line 12 iterates over all k elements S_b of vector $\boldsymbol{paths}$. Nested to the loop on Line 12, another loop iterates over all the elements of every S_b, with a maximum of m. On every iteration, the element δ_b of vector $\boldsymbol{max\_ts}$ is accessed and the elements $\delta_b(i-2)$, $\delta_b(i-1)$, and $\delta_b(i)$ are also accessed. To avoid iterating over the list δ_b to find these elements, a pointer to these elements can be coupled to $S_b(i)$. This way, the access to $\delta_b(i-2)$, $\delta_b(i-1)$, and $\delta_b(i)$ in this loop has worst case complexity $O(1)$. Therefore, the loop in Line 12 has complexity $O(km)$.

Using a binary vector indexed by s to implement $\mathcal{R}$, the insertion of s into the set $\mathcal{R}$ has complexity $O(1)$. Given the complexities of the nested loops, the complexity of the loop that starts on Line 7 is $O(n^2 + kmn + n)$. As a consequence, the time complexity of the Algorithm 1 is given by $O(1 + km + n^2 + kmn + n)$. By the notation O, the time complexity is $O(n^2 + kmn)$, i.e., the biggest between $O(n^2)$ and $O(kmn)$.

4.6.2 Comparison with the optimal solution

To analyze the approximation offered by our greedy approach, we compare the solutions found by our algorithm with the ones obtained by the optimal problem solution. To this end, we generate artificial data that simulates a bus mobility scenario, composed of Fog node candidates. The Fog node candidates are organized in a grid and bus mobility is simulated. We perform the evaluation for 5 buses moving around a grid with 25 bus stops as well as a scenario with 10 buses moving around a grid with 100 bus stops. For every scenario, we generate 30 datasets.

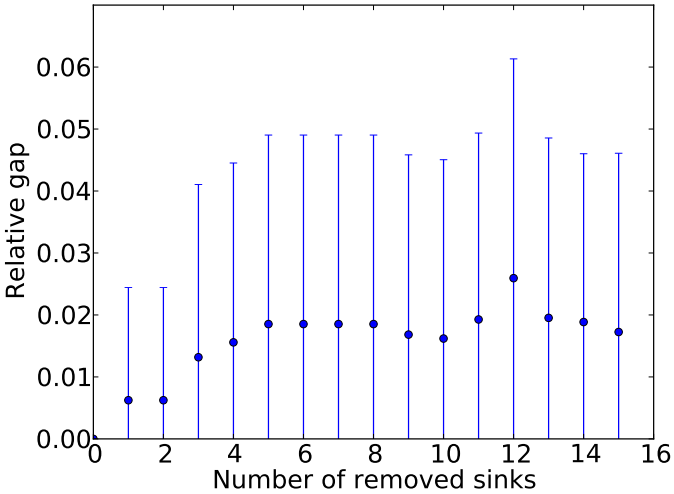
The parameters used on each dataset are: the size of the path of each bus, the delays between stops along a bus path, and the bus stops on each path. To generate the datasets, we sample a real dataset for the size of the paths (normalized by the number of bus stops) and the delays along the path. The bus stop sequence is chosen randomly. We perform the sampling and randomization with uniform distribution.

To solve the problem, we use IBM ILOG CPLEX 12.5.1. Figures 4.5a and 4.5b show the results for 25 and 100 bus stops, respectively, by plotting the relative gap in Δ_{max} achieved by the proposed algorithm and the optimal solution, for every number of removed candidates. The relative gap is the difference between the two results divided by the optimal result. The error bars represent a 95% confidence interval.

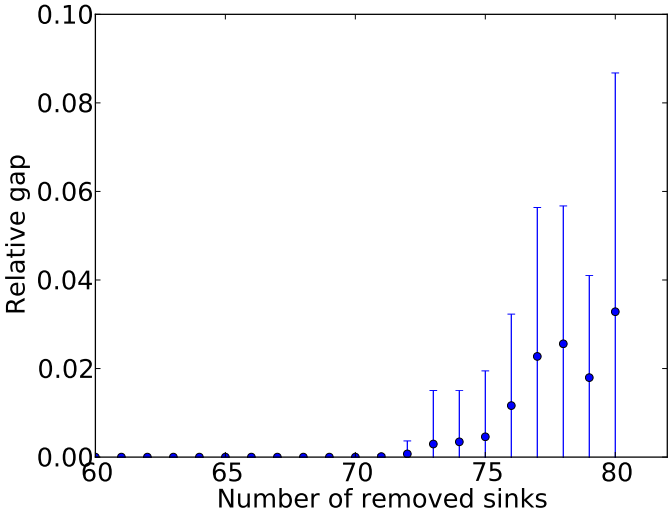
The results show that the proposed algorithm finds solutions close to one optimal solution. In the worst case of the studied scenarios, the algorithm is less than 10% distant from the optimal.

4.7 Applying the algorithm to real-world data

Our case study consists in running Algorithm 1 using a dataset containing the positions of buses and bus stops in the city of Rio de Janeiro, Brazil. The instantaneous GPS positions of buses and the positions of bus stops are published on the website of Rio de Janeiro Federation of Passenger Transportation Companies (FETRANSPOR in the Brazilian acronym) [51, 52]. Bus positions are collected and stored in one file with the instantaneous positions of all buses, updated every minute. Bus stop



(a) Grid with 25 Fog node candidates and 5 buses.



(b) Grid with 100 Fog node candidates and 10 buses.

Figure 4.5: Relative gap in network maximum delay in function of removed Fog node candidates on a 5x5 and 10x10 Fog node candidates grid.

positions, which are fixed, are provided on a separate file. From these datasets, we generate the data we needed to solve the Fog placement problem.

Instantaneous positions of buses are collected for a 24h interval, from November, 30th, 0:00 h, to December, 1st, 0:00 h, in the year of 2016. This data is inserted into a database, together with the positions of bus stops. We define that a bus and a bus stop make contact if the distance between them is less than or equal to 300 m. This range is assumed because it is a typical value for Vehicle to Infrastructure (V2I) communication [53]. A tuple (instant, bus, bus stop) is selected for every instant when a bus makes contact to a bus stop and a new dataset is built with all the tuples. This dataset contains data from 6,272 bus stops and 6,683 buses. From this dataset, it is possible to obtain the sets $\mathcal{B}$, $\mathcal{S}$, S_b, and δ_b.

4.7.1 Dataset analysis

As pointed by Dias [54], the dataset provided by FETRANSPOR may contain some inconsistencies. We analyze the dataset to verify the extent of such inconsistencies. The analysis of the dataset shows that data must be filtered before further processing, especially when it comes to large intervals between contacts. Figure 4.6 shows the cumulative probability function of the maximum delay for each bus. It shows that approximately 20% of buses have maximum delay greater than 7,200 s. This means that 20% of the buses travel more than two hours without stopping at a bus stop to serve passengers. While it is extremely rare for urban buses to remain two hours without stopping at a bus stop, it is expected that a sampling rate of 1 position/minute leaves out many contacts that last less than two minutes. Moreover, it is possible that some buses are out of service, but with their GPS equipment turned on. Our solution to this problem is to filter out buses that have large intervals between contacts, assuming that these buses are not fit for data collection.

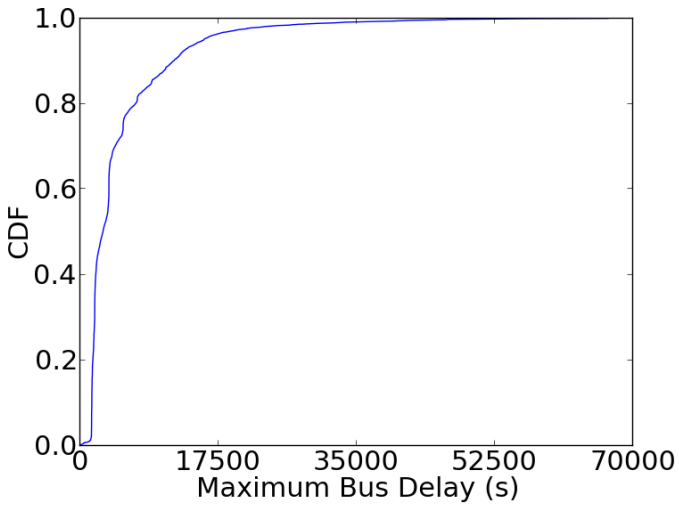


Figure 4.6: Cumulative distribution of maximum delay of each bus.

Three different filters are applied to the dataset, eliminating every bus b that, for some i, has $\delta_b(i) > 7{,}200$ s, $\delta_b(i) > 3{,}600$ s, or $\delta_b(i) > 1{,}800$ s. Figure 4.7a shows the distribution of all delays in the network before the filter. Figure 4.7b shows the distribution of the delays in the network after the filter of 7,200 s, Figure 4.7c shows the distribution of the delays in the network after the filter of 3,600 s, and Figure 4.7d shows the distribution of the delays in the network after a 1,800 s filter.

Figure 4.7 shows that, in every case, the filters set a threshold for the maximum delay. Table 4.2 shows the parameters of the original dataset and the filtered ones. These results show that, after all the filters, the remaining buses still make contact to approximately 99% of all the 6,272 bus stops somewhere in their paths. This indicates that there is no significant loss in spatial coverage. On the other hand,

Table 4.2: Dataset parameters.

Filter maximum delay (s)	**Number of Buses**	**Stops visited**	**Candidates in $\mathcal{I}$**
No filter	6,683	6,272	1,266
7,200	5,429	6,272	1,085
3,600	4,104	6,266	933
1,800	2,200	6,218	641

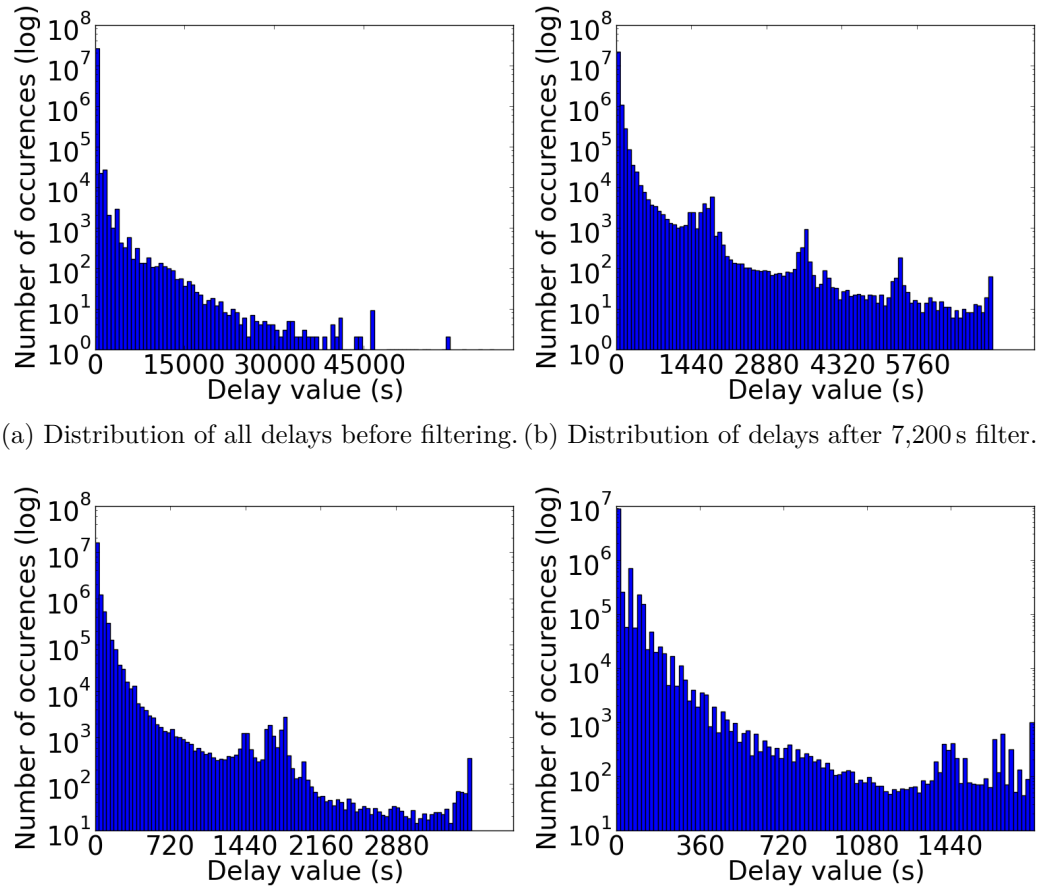
limiting the number of buses that are used for gathering data means that the area covered is less visited by sensing buses, making it less likely for a sensor to detect some event of interest [3, 21]. We discuss the coverage of the network in Chapter 5.

4.7.2 Algorithm results

We employ Algorithm 1 to remove Fog node candidates according to the maximum budget for each one of the datasets. Figure 4.8a shows the network delay Δ_{max} when executing the algorithm using the dataset obtained after the 7,200 s filter, for different numbers of removed candidates. Similarly, Figure 4.8b and Figure 4.8c show, respectively, the same results for the filters of 3,600 s and 1,800 s. We omit the values for fewer removals because there is no modification on them. The dots in these figures are inflection points, meaning that the delay is the same for a given range of removed candidates. These inflection points happen because the algorithm removes successive Fog node candidates without penalties to the network maximum delay until a certain threshold, limited by some bottleneck Fog node. When the bottleneck Fog node candidate is removed, the network maximum delay finally increases. Then, the cycle repeats for a new threshold.

The results show that our solution is capable of eliminating more than 83% of the Fog node candidates (i.e., 5205, 5200, and 5160 bus stops, for the filters of 7,200 s, 3,600 s, and 1,800 s, respectively) while having less than 10% delay increase (i.e., 7,920 s, 3,960 s, and 1980 s, for the filters of 7,200 s, 3,600 s, and 1,800 s, respectively). In Figure 4.8, we observe that the delay obtained with the removal of 83% of the bus stops is 7,200 s, 3,708 s, and 1,800 s, for the filters of 7,200 s, 3,600 s, and 1,800 s, respectively. In the case of the filter of 1,800 s, it is possible to use 20% (i.e., 1,244 bus stops) of the bus stops as Fog nodes and still have a network maximum delay of 30 minutes.

The results also show that the network maximum delay starts increasing faster when more Fog node candidates are removed. Furthermore, the analysis shows that different filters can provide different delay levels. In Section 3.1, we discuss that



(a) Distribution of all delays before filtering. (b) Distribution of delays after 7,200 s filter.

(c) Distribution of delays after 3,600 s filter. (d) Distribution of delays after 1,800 s filter.

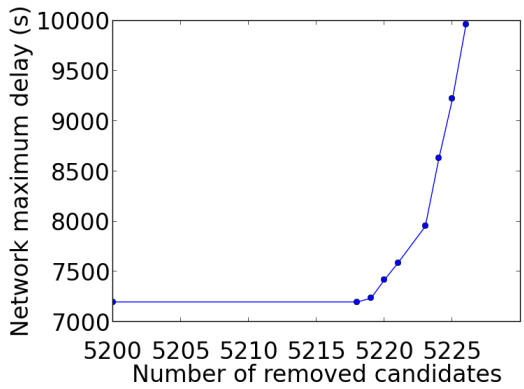
Figure 4.7: Distribution of delays before filtering and after 1,800 s, 3,600 s, and 7,200 s filters.

applications have different tolerances to delay. A possible scenario is that buses are divided into different delay levels, balancing the trade-off between delay constraints and the amount of data available. This option can also decide which buses and bus stops have the potential to serve which application. We use a similar idea in Section 5.2 to estimate the coverage of such a network.

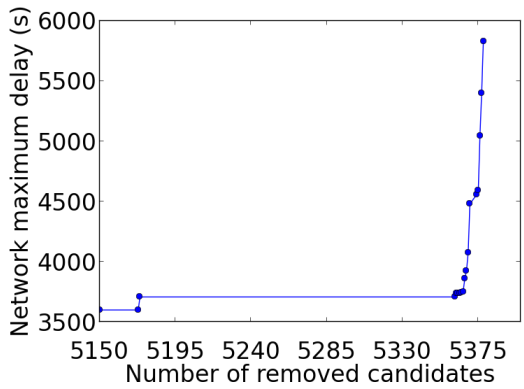
4.8 Reducing the problem cardinality

In Section 4.5.1, we present a MILP formulation for the Fog node placement problem. Since the p-center is NP-Hard [50], solving the problem for even a small amount of buses can be unpractical. In this section, we show that a suboptimal solution can be used to solve the Fog node placement problem. Then, we use the algorithm for Fog node placement presented in Section 4.6 to generate a suboptimal to solve the MILP formulation, finding an optimal solution.

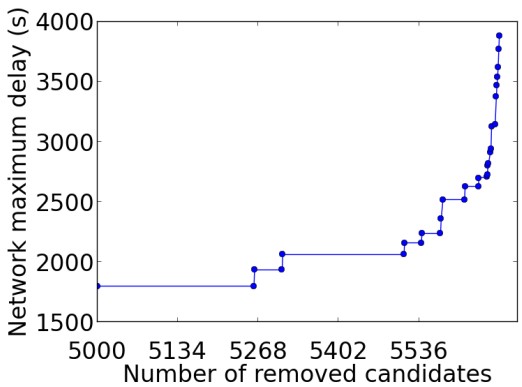
In the MILP formulation of the optimal Fog node placement problem, the number



(a) Network maximum delay ($\Delta_{\max}$) filtered for 7,200 s, as a function of the number of removed candidates.



(b) Network maximum delay ($\Delta_{\max}$) filtered for 3,600 s, as a function of the number of removed candidates.



(c) Network maximum delay ($\Delta_{\max}$) filtered for 1,800 s, as a function of the number of removed candidates.

Figure 4.8: Network maximum delay ($\Delta_{\max}$) for different filters, as a function of the number of removed candidates.

of variables x_s is the number of elements in the set $\mathcal{S}$. The number of variables y_{brs} and parameters d_{brs} is the same, and follows the expression $\sum_{b \in \mathcal{B}} \frac{(1+n_b)n_b}{2}$, where n_b is the number of bus stops that b encounters along its path. We present three strategies to reduce the number of variables y_{brs} and parameters d_{brs}:

- **Elimination by suboptimal solution**: As defined in Equation 4.13, $\Delta_{\max}$ is always the biggest delay d_{brs} suffered by any data gathered by the network (more formally, it is the biggest d_{brs} to which y_{brs} equals to one). We can thus reduce the solution space of the problem using any viable solution. Let $\Delta_{\max}$ be the solution of the problem and $\Delta^{\dagger}_{\max}$ be a viable solution. If $\Delta^{\dagger}_{\max}$ is viable, $\Delta_{\max}$ is smaller or equal to $\Delta^{\dagger}_{\max}$. Therefore, it is possible to eliminate from the original dataset all delays d_{brs} that are bigger than $\Delta^{\dagger}_{\max}$ and their respective y_{brs}. In Figure 4.9, the suboptimal solution $\Delta^{*}_{\max}$ equals to 3 and d_{xz} equals to 4. Therefore, we can use this strategy to eliminate d_{xz}.

- **Elimination by zero value**: In a given instant, a bus can make contact to many Fog nodes simultaneously. In this situation, we can have a sequence of bus stops with a delay equal to zero. When the problem is transformed into a p-center instance, this creates edges with weight d_{brs} equal to zero. Since there is no delay nor data gathered between these stops, we can eliminate d_{brs} and y_{brs} without losing the optimality of the solution. We use this logic to eliminate the variable d_{xy} and its corresponding y_{xy} from the example in Figure 4.9.

- **Elimination by guaranteed choice**: The Fog node placement problem has a set $s \in \mathcal{I}$ of bus stops that must be chosen. If the path of a bus $b \in \mathcal{B}$ contains s, and $s \in \mathcal{I}$, we can assume that any data gathered before s is delivered at most when b encounters s. This means that we can eliminate any variable related to an edge that starts before s and ends after s. In Figure 4.9, $\mathcal{I} = x$ and d_{wy} starts before and ends after x in the path of the bus. Therefore, we eliminate d_{wy} and its corresponding y_{wy} using this strategy.

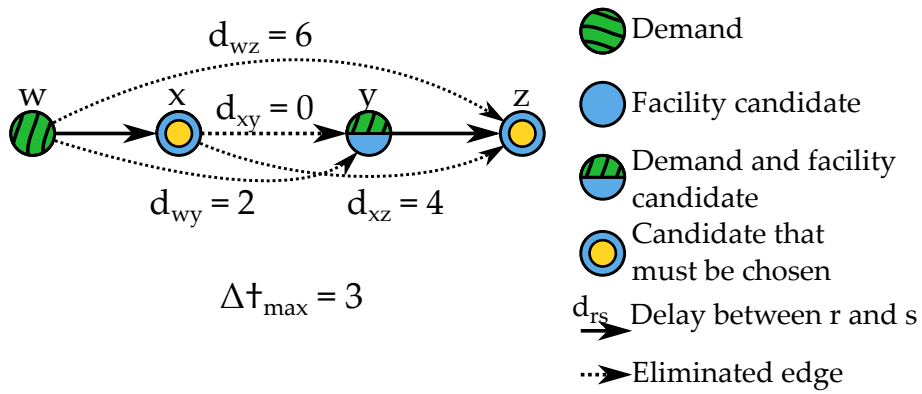
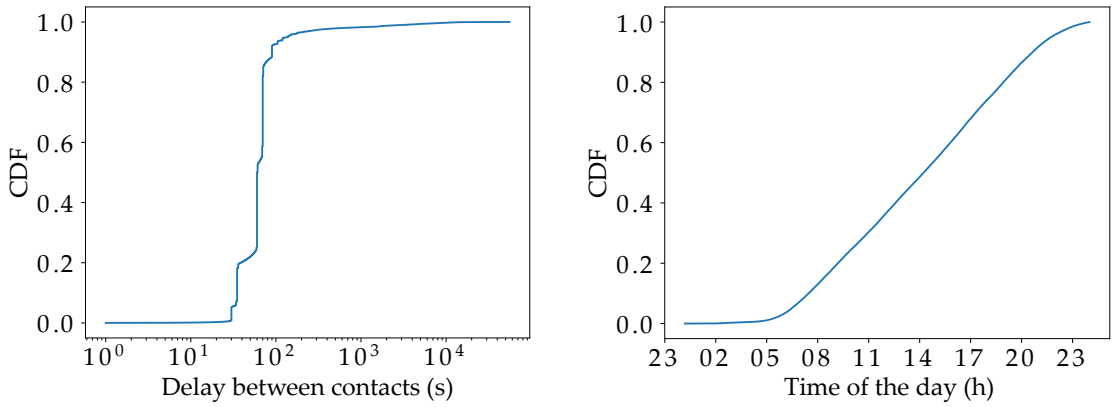


Figure 4.9: Examples of the edge elimination procedure.

To apply the first strategy, it is necessary to obtain a suboptimal solution. Hence, we use the solutions obtained by Algorithm 1 to reduce the problem cardinality.

4.9 Cardinality reduction applied to a real scenario

In Section 4.7, we apply the Fog Node Placement Algorithm to a dataset with real bus mobility data. We use the same API offered by FETRANSPOR [51, 52] to apply the cardinality reduction. We gather instantaneous positions of buses throughout a 24h interval. We start on November, 5th, 0:00 h and end on November, 6st, 0:00 h in the year of 2017. Also, because of the complexity of solving the MILP, we limit



(a) Cumulative distribution of delivery delays with no filter.

(b) Contacts cumulative distribution throughout the day.

Figure 4.10: Dataset attributes before filtering.

our analysis to the South Zone of the city of Rio de Janeiro. We consider a bus in the South Zone if it is inside the coordinates with latitude between -23.013874 and -22.918251, and longitude between -43.296139 and -43.149197 during all the analyzed time. As in Section 4.7, we consider that a bus is in contact with a bus stop if the distance between them is less than 300 m [53]. In the sequence, we detail the construction of the dataset and analyze it.

4.9.1 Dataset construction

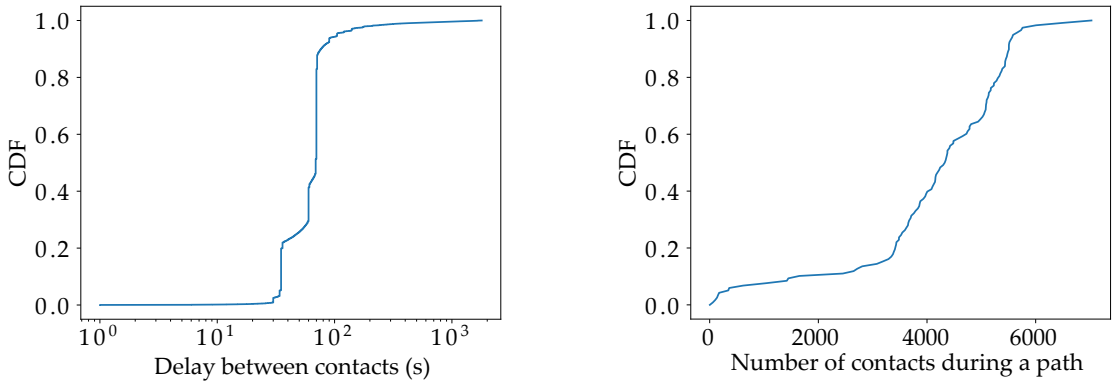
We analyze the delays of the dataset, to evaluate if it is well suited for our procedure. Figure 4.10a shows the cumulative distribution of the delays. We observe that more than 90% of the delays are smaller than 100 s. Figure 4.10b shows the cumulative distribution of contacts in the function of the time of the day. We note that there are fewer contacts at the end of the evening and the beginning of the morning. One possibility is that there are fewer buses serving passengers. For this reason, we consider only contacts happening between 8 h and 22 h.

We use the results from Zanella *et al.* [1] and consider that delays larger than 30 minutes are too long. We consider that a path starts with the first contact after a delay that is larger than 30 minutes, and it ends with the last contact before the second delay larger than 30 minutes. We can then build $\mathcal{I}$. We also eliminate from the dataset any buses that have more than two delays bigger than 30 minutes, considering that they are not suitable for sensing. Table 4.3 summarizes the attributes of the resulting data.

Figure 4.11a illustrates the cumulative distribution of delays after the filters. We eliminate the contacts before 8 h and after 22 h, and eliminate buses with delays bigger than 30 min in their paths. Figure 4.11b shows the cumulative distribution of

Table 4.3: Dataset attributes.

Attribute	**Value (#)**
Buses	116
Bus stops	6.310
Bus stops with any contact	744
Total of contacts	485.208
Bus stops in $\mathcal{I}$	116



(a) Delay cumulative distribution after filtering.

(b) Contacts cumulative distribution after filtering.

Figure 4.11: Dataset attributes after filter contacts by time of day and by delays bigger than 30 minutes.

the number of contacts of each bus in the dataset. We then use this filtered dataset to choose the optimal Fog node placement.

4.9.2 Reduction of dataset cardinality

We apply the pre-processing defined in Section 4.8 to the dataset obtained in Section 4.9.1. Table 4.4 shows the suboptimal delivery delay $\Delta^{\dagger}_{\max}$ obtained by the Fog Node Placement Algorithm, for different values of n_{budget}. We can observe that after 120 Fog nodes, it is not possible for the algorithm to lower the biggest delay in the network. To obtain the results, we use a 2.93GHz Intel(R) Xeon(R) X5570, with 100 GB of RAM. The cardinality reduction can make a solution possible, with execution times lower or equal to 2.42 s.

We use the suboptimal results to apply the strategies described in Section 4.8 and then use the dataset with reduced cardinality as an input to the p-center problem modeled in Section 4.5.1.

Table 4.5 shows the number of variables d_{brs} and y_{brs} that are eliminated from the problem. The values of n_{budget} used to find optimal solutions are bigger than 120 Fog nodes. Therefore, we only need to find the suboptimal solution for 120 Fog

Table 4.4: Suboptimal results obtained with Algorithm 1.

n_{budget}	$\Delta^{\dagger}_{\max}$	**Execution time (s)**
116	2067	2.35
117	1917	2.37
118	1848	2.39
119	1801	2.40
120 or more	1794	2.42 (for 120)

nodes.

Table 4.5: Number of $d_{brs}y_{brs}$ restrictions in the problem.

Cardinality reduction	**Number of elements** $d_{brs}y_{brs}$
Before cardinality reduction	1.122.629.838
After cardinality reduction for 120 Fog nodes	393.157

4.9.3 Optimal results after cardinality reduction

We use the data to build the Fog node placement problem as a p-center instance, as described in Section 4.5.1. We use the values 200, 300, 400, 500, 600, and 700 for n_{budget}, since there are 744 stops with which there is any contact and 116 stops that have to receive a Fog node. We execute the problem using IBM ILOG CPLEX 12.5.1, in a processor 2.93 GHz Intel(R) Xeon(R), with 100 GB of RAM. Table 4.6 shows the obtained results.

Table 4.6: Optimal results of the Fog node placement problem with reduced cardinality.

n_{budget}	$\Delta_{\max}$ **(s)**	**Execution time (s)**
200	1786	1601
300	1786	1643
400	1786	1354
500	1786	1376
600	1786	1378
700	1786	1381

The instances with reduced cardinality converge to the optimal solution in 1643 s or less in all the observed cases. As a comparison, we try to solve the instances without the cardinality reduction. After 50 h of CPU time, the execution is still in the phase of construction of the data structures of the problem.

4.10 Remarks

Opportunistic communication can help bus-based sensing to offload data. Buses play the role of Sensing nodes, gathering data and delivering it to Fog nodes when a connection is available. The Fog nodes, coupled to bus stops, sends data to the Cloud node, which processes and serves data to the users. Nevertheless, the delays between the gathering time and the delivery time can render data useless to the applications.

In this chapter, we presented a model to estimate the delay of data collected by a bus-based mobile wireless sensor network. We also presented an optimization model to minimize the maximum delay when the number of Fog nodes is constrained by a certain budget. We have shown that this problem can be interpreted as an instance of the P-Center problem. We also proposed a heuristic to find good solutions when finding an optimal solution is impractical. Finally, we use the heuristic to reduce the cardinality of the optimal model, making it possible to solve bigger instances of the problem.

We applied the models and methods proposed to real GPS data from the bus fleet of the city of Rio de Janeiro. The results have shown that, for the city of Rio de Janeiro, it is possible to obtain a network using approximately 16% of the bus stops as Fog nodes and have a maximum delay of 30 minutes in data delivery. As shown in Table 3.1, this delay is compatible with the waste management application. Our results have also shown that the cardinality reduction made it possible to find an optimal solution to the Fog node placement problem to the buses and bus stops of the south zone of Rio de Janeiro in about 25 minutes.

Chapter 5

A Coverage Metric for Bus-based MWSNs

In this thesis, we study a scenario where buses carry nodes capable of sensing environmental data, as illustrated in Figure 2.1. This data is stored in the buses and transmitted to a Fog node through a wireless interface. Fog nodes, located at bus stops, receive data and send them to the Task Manager node through the Internet. One aspect that we must be careful about is that applications atop a bus-based mobile wireless sensor network may have specific requirements in terms of data completeness, timeliness [55], and granularity [1]. The completeness and granularity of spatial coverage are related to the knowledge of the complete area of the city and to what extent the measurements are spread through this city. Also, acquiring measurements about different areas can reduce noise and uncertainties about neighboring regions [56]. The timeliness is related to how up-to-date data is when it is finally ready for an application [55].

As discussed in Section 3.1, smart city applications have requirements in terms of the delivery delay and the frequency of data collection. As illustrated in Table 3.1, applications can tolerate different maximum delays. Additionally, applications can tolerate a minimum measurement frequency. This means that, for every application, in a given time interval, there should be a minimum number of measurements and the data generated by each measurement should be delivered to the application within a maximum delay. We denote the maximum delivery delay tolerated by an application as $D_{\max}$, and the minimum measurement frequency as $F_{\min}$.

In this chapter, we model the spatial coverage of a bus-based mobile wireless sensor network, relying on the fact that buses have fixed trajectories. We then propose a MILP formulation to maximize coverage while using only a subset of the total bus fleet as Sensing nodes. We use the problem to discover the maximum coverage obtained by different numbers of Sensing nodes [19, 21]. We then improve this metric to consider the $D_{\max}$ and the $F_{\min}$ of a target application. We apply this

Table 5.1: Notations used in the coverage model.

Notation	Description	Type
$\mathcal{G}$	Graph representing the road map of a city	Graph
$\mathcal{V}$	Vertices of the road map of a city	Set
$\mathcal{E}$	Street segments of the city	Set
$\mathcal{E}_s$	Street segments covered by the buses chosen in the problem output	Set
$\mathcal{E}_c$	The subset of $\mathcal{E}$ that is covered for the considered application	Set
$\mathcal{B}$	Urban buses that serve a city	Set
$\mathcal{B}_s$	Buses equipped with Sensing nodes, the problem output	Set
$\mathcal{N}_{(x_i,x_j)}$	Buses that can cover street segment (x_i, x_j)	Set
$\mathcal{T}_b$	Street segments that can be covered by bus b	Set
$\mathcal{P}$	The set containing all the paths of the buses	Set
$\mathcal{A}$	The applications served by the bus-based MWSN	Set
$\mathcal{H}$	The set containing the demands of the Maximum Covering Location Problem	Set
$\mathcal{J}$	The set containing the facility candidates of the Maximum Covering Location Problem	Set
a_h	The benefit of satisfying demand h in the Maximum Covering Location Problem	Parameter
(x_i, x_j)	The street segment that starts in vertex x_i and ends in vertex x_j	Parameter
$l_{(x_i,x_j)}$	The length of street segment (x_i, x_j)	Parameter
p	Total number of buses to be equipped with Sensing nodes	Parameter
$C_a^{\mathcal{B}}$	The coverage obtained by the buses in $\mathcal{B}$ to the application a	Parameter
$P_b[m]$	The mth edge visited by bus $b \in \mathcal{B}$ in its path P_b	Parameter
$D_{\max}$	The maximum tolerated delivery delay for the considered application	Parameter
$F_{\min}$	The minimum measurement frequency required by the considered application	Parameter
$v^T_{(x_i,x_j)}$	The number of visits received by street segment (x_i, x_j) on a given time interval	Parameter
$F_{(x_i,x_j)}$	The frequency at which the street segment (x_i, x_j) is visited	Parameter
K_a^b	The contribution of bus b to the coverage of application a	Parameter
P_b	The path $((x_i, x_j), (x_j, x_k), (x_k, x_l) \ldots)$ of bus b, in terms of edges in $\mathcal{E}$	Sequence
w_b	Binary value indicating if bus b is chosen to be equipped with a Sensing node	Variable
$z_{(x_i,x_j)}$	Binary value indicating if street segment (x_i, x_j) is covered	Variable
C	Total simple coverage of the city	Variable
C_c	Delay-aware coverage of the city for the considered application	Variable

metric to real mobility traces of buses from the city of Rio de Janeiro, considering the requirements of the applications of waste management, air quality monitoring, and noise monitoring. We present the results in the form of an abacus, relating coverage to different applications and their tolerances, in terms of delivery delay and measurement frequency. We also compare the coverage obtained to the scenario where bus stops serve as gateways to static sensors [22, 23]. Finally, we use this delay-aware coverage metric to rank the buses and verify if the same bus is equally important to different applications [24]. Understanding the importance of each bus can help to schedule the implementation and maintenance of the system.

5.1 A simple coverage model

Buses follow predictable paths that, all together, do not cover all of the streets of a city. As such, it is important to model the spatial coverage of buses to predict the coverage of a bus-based mobile sensor network. For the sake of readability, the notation we use in this chapter is summarized in Table 5.1.

5.1.1 The city map as a graph

We divide each street into non-overlapping parts named *street segments*. The street segment is used as an atomic unit of the measured region. A street segment must be a part of a street with a single entrance, a single exit, and a single possible path inside it. An example of a street segment is a portion of street between two consecutive corners. A similar definition can be found in the work by Ali and Dyo [12] and Opensense [16].

A common way of representing the road map of a city consists of modeling it as a directed graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$. The set $\mathcal{V}$ contains the vertices of $\mathcal{G}$. Each vertex $x_1, \ldots, x_{|\mathcal{V}|} \in \mathcal{V}$ represents an intersection, a curve, or another point of interest in the street topology. The set $\mathcal{E}$ contains the edges of $\mathcal{G}$. An edge $(x_i, x_j) \in \mathcal{E}$ exists if and only if it is possible to follow a street from x_i to x_j, not visiting any other vertex. A weight $l_{(x_i,x_j)}$ is associated to each edge (x_i, x_j), representing the distance between x_i and x_j. A street is a sequence of vertices since a street is an ordered collection of points of interest. A vertex can be part of more than one street when this vertex is an intersection between two or more streets. Every edge in $\mathcal{E}$ represents a portion of street with a single entrance, a single exit, and a single possible path inside it. Hence, we can conclude that each edge in $\mathcal{E}$ is a street segment.

We denote by $\mathcal{E}$ the set of all street segments in the city and by $\mathcal{B}$ the set of buses. The path of a bus is composed of street segments. If the path of a bus $b \in \mathcal{B}$ contains a street segment $(x_i, x_j) \in \mathcal{E}$, the section (x_i, x_j) can be covered by bus b. The sections covered by bus b compose the set $\mathcal{T}_b$. Although different buses may follow different paths, a path of a bus might overlap with the path of other buses. Therefore, a single street segment (x_i, x_j) might be covered by more than one bus. We denote by $\mathcal{N}_{(x_i,x_j)}$ the set of buses that cover the street segment (x_i, x_j).

It is also possible that some street segments remain uncovered because no buses cross them. We suppose that these street segments are covered by a complementary sensing system and are then out of the scope of our model.

5.1.2 Coverage as a function of street segments

We define as $\mathcal{B}_s \subset \mathcal{B}$ the subset of buses that are equipped with Sensing nodes. We can now define the set of covered street segments $\mathcal{E}_s$ as:

$$\mathcal{E}_s = \bigcup_{b \in \mathcal{B}_s} \mathcal{T}_b, \tag{5.1}$$

where $\mathcal{B}_s$ is the set of buses that carry a sensing device and $\mathcal{T}_b$ is the set of street segments covered by bus b. Figure 5.1 illustrates this situation. In this figure, bus b covers the set of street segments $T_b = \{(x_i, x_j), (x_j, x_k), (x_k, x_l), (x_l, x_m)\}$ and bus

c covers $T_c = \{(x_j, x_k), (x_k, x_l)\}$. Therefore, the set $\mathcal{E}_s$ of covered street segments is $T_b \cup T_c = \{(x_i, x_j), (x_j, x_k), (x_k, x_l), (x_l, x_m)\}$.

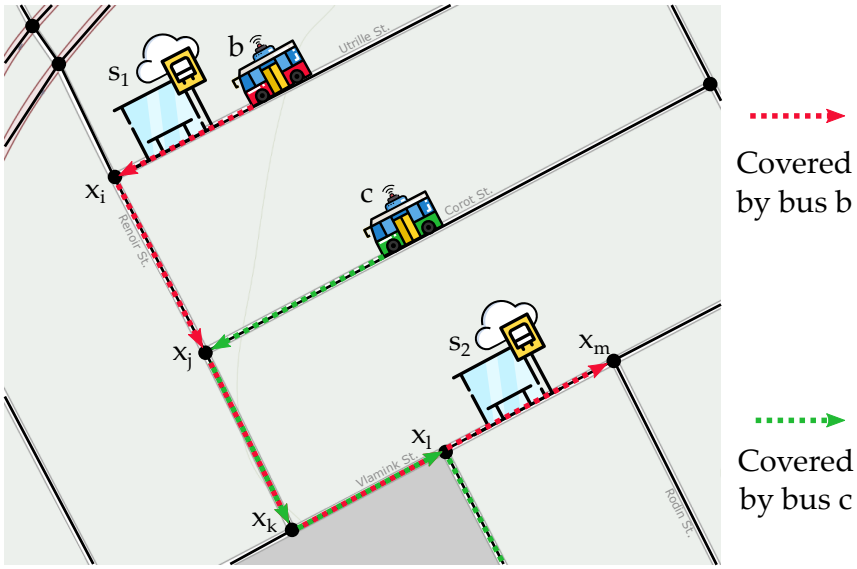


Figure 5.1: Coverage of street segments by buses equipped with sensors.

Street segments may have different lengths. We expect that sensors generate more data when the bus is passing by longer streets. For this reason, coverage C should also integrate the length of every section:

$$C = \sum_{(x_i,x_j)\in\mathcal{E}_s} l_{(x_i,x_j)}, \tag{5.2}$$

where $l_{(x_i,x_j)}$ is the length of section i.

For modeling purposes, it is possible to associate a variable $z_{(x_i,x_j)}$ to every street segment $(x_i, x_j) \in \mathcal{E}$, indicating whether the segment is covered:

$$z_{(x_i,x_j)} = \begin{cases} 1, & \text{if } (x_i, x_j) \in \mathcal{E}_s, \\ 0, & \text{otherwise.} \end{cases} \tag{5.3}$$

We can rewrite then Equation 5.2 as:

$$C = \sum_{(x_i,x_j)\in\mathcal{E}} l_{(x_i,x_j)} z_{(x_i,x_j)}. \tag{5.4}$$

If all buses in the city have embedded sensing devices, i.e., $\mathcal{B}_s = \mathcal{B}$, we obtain the maximum coverage. Nevertheless, in real life, this choice may produce prohibitive installation and maintenance costs. As a consequence, for a given budget, we need to limit the number of buses equipped with Sensing nodes. Hence, the problem is to choose which buses should be equipped with the available Sensing nodes.

Let $\mathcal{B}$ be the set of buses and p be the number of Sensing nodes, defined by the available budget. We need to select a subset of buses $\mathcal{B}_s \subset \mathcal{B}$ such that $|\mathcal{B}_s| = p$. Since buses may cover different sets of street segments, with different lengths, the choice of $\mathcal{B}_s$ may affect the coverage of the network. Let also $\mathcal{N}_{(x_i,x_j)}$ be the set of buses that cover (x_i, x_j) if equipped with sensing capabilities. Thus, section (x_i, x_j) is covered by bus b if $b \in \mathcal{N}_{(x_i,x_j)}$ and $b \in \mathcal{B}_s$. The problem, described next, is to select $\mathcal{B}_s \subset \mathcal{B}$, with cardinality p, to maximize the coverage of Equation 5.4.

5.1.3 Mixed-Integer Linear Programming formulation

We model the problem of choosing p buses as a Mixed-Integer Linear Programming (MILP) problem. We formulate the problem as a MILP because it is possible to either solve the problem optimally or, for too large instances, estimate the gap between a suboptimal solution and the best possible solution. In Section 5.1.4, we show that the problem is equivalent to the Maximal Covering Location Problem, a known problem in the literature. The MILP formulation is represented as follows:

$$\text{maximize:} \quad \sum_{(x_i,x_j)\in\mathcal{E}} l_{(x_i,x_j)} z_{(x_i,x_j)} \tag{5.5}$$

$$\text{subject to:} \quad \sum_{b\in\mathcal{N}_{(x_i,x_j)}} w_b \geq z_{(x_i,x_j)}, \quad \forall (x_i, x_j) \in \mathcal{E}; \tag{5.6}$$

$$\sum_{b\in\mathcal{B}} w_b = p; \tag{5.7}$$

$$w_b \in \{0, 1\}, \quad \forall b \in \mathcal{B}; \tag{5.8}$$

$$z_{(x_i,x_j)} \in \{0, 1\}, \quad \forall (x_i, x_j) \in \mathcal{E}. \tag{5.9}$$

The objective function of Equation 5.5 maximizes the spatial coverage defined in Equation 5.4. Equation 5.6 states that a street segment (x_i, x_j) is covered (i.e., $z(x_i, x_j) = 1$) if the solution has at least one of the buses that pass through (x_i, x_j). Since this is a maximization problem, there is no need to limit a minimum value for variables $z_{(x_i,x_j)}$, because these variables contribute positively for the objective function, assuming the maximum possible value. Equation 5.7 ensures that the number of chosen buses is p. Equations 5.8 and 5.9 define w_b and $z_{(x_i,x_j)}$ as binary variables. When the problem is solved, a variable $w_b = 1$ indicates that bus b is chosen to receive a Sensing node and $w_b = 0$ indicate that bus b should not carry a Sensing node. Respectively, $z_{(x_i,x_j)} = 1$ indicates that street segment (x_i, x_j) is covered by the current configuration, while $z_{(x_i,x_j)} = 0$ indicates that the street

segment (x_i, x_j) is not covered. The set $\mathcal{B}_s$, containing the buses that maximize coverage, is the set of buses b for which w_b equals to 1.

5.1.4 Maximal Covering Location Problem

We show that the problem of choosing a limited set of buses while maximizing coverage is equivalent to the Maximal Covering Location Problem (MCLP) [57]. In MCLP, there is a set $\mathcal{H}$ of demands distributed on space. For every demand $h \in \mathcal{H}$, there is a value a_h related to the benefit of satisfying demand h. Demands must be satisfied by a set $\mathcal{J}$ of facility candidates, also distributed on space. The MCLP defines that an installation j can satisfy a demand h if and only if the distance between h and j is less or equal than a given distance. In this sense, it is possible to determine a set $\mathcal{N}_h \subset \mathcal{J}$ of candidates that satisfy demand h. Additionally, there is a limit on the number of installations that can be built, denoted by p. The objective of MCLP is to choose a subset $\mathcal{J}_f \in \mathcal{J}$ to build installations, maximizing the benefit of the satisfied demands, such that $|\mathcal{J}_f|$ equals to p.

The transformation of our problem into an MCLP consists in considering street segments as demands and buses as installation candidates. The length $l_(x_i, x_j)$ of the street segment (x_i, x_j) is considered as the benefit a_h of satisfying demand h. The inverse transformation is also possible by transforming demands in street segments, installation candidates in buses, and benefit a_h of satisfying demand h into the length of street segment (x_i, x_j).

The bidirectional transformation proves the equivalence of both problems. The MCLP is an NP-Hard problem [57]. Since there is a bidirectional transformation between the bus coverage problem and the MCLP, our problem is also NP-Hard.

5.1.5 Case study

We assess the behavior of our model by analyzing the street coverage in the city of Rio de Janeiro, Brazil. Our analysis relies on real data containing the positions of all buses in the city for a day-long period. Besides, we use this data to build the input for the MILP formulated in Section 5.1.3, which consists of:

- Set $\mathcal{B}$, containing all the buses in the city;
- Set $\mathcal{E}$, containing all the street segments in the city;
- Set $\mathcal{N}_{(x_i,x_j)}$, for every $(x_i, x_j) \in \mathcal{E}$, containing all the buses that can cover section (x_i, x_j);
- Parameters $l_{(x_i,x_j)}$, associated with the lengths of every $(x_i, x_j) \in \mathcal{E}$.

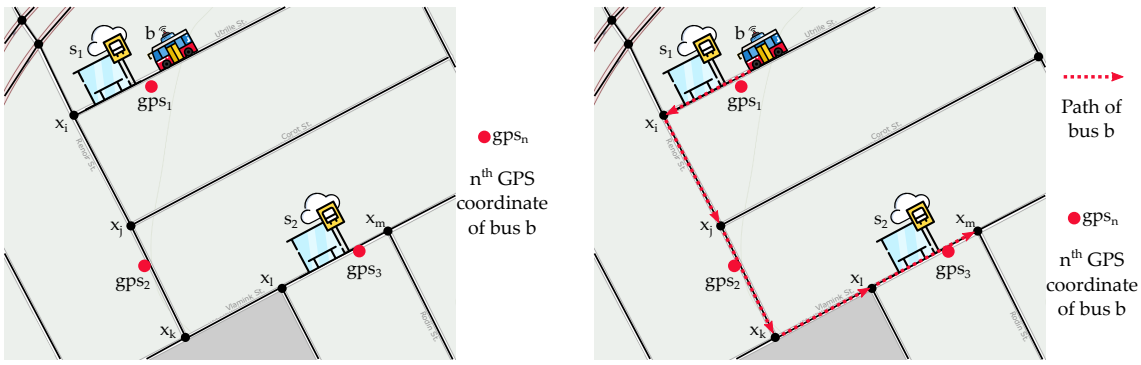
We obtain data from real mobility traces from buses. Then, we pre-process data by filtering it and detecting the street segments covered by each bus. We then use this data to build the MILP defined in Section 5.1.3. Finally, we solve the problem and obtain its solution. The next sections detail the whole procedure.

5.1.6 Obtaining data

In Section 4.7 we use an API offered by FETRANSPOR to build a dataset and perform a case study of our method. We use the same API to build another dataset and study the coverage problem. The API offers, for each bus in the fleet of Rio de Janeiro, a tuple containing the bus identification, its instantaneous GPS coordinates, and the timestamp of the acquisition of this information. The tuples are refreshed every minute.

We assume that buses follow fixed paths and that departure intervals might vary in the course of a day, but the same schedule is followed on all weekdays. We gather data from the API during 24 h, between May 10th, 0:00 h and May 11th, 0:00 h, in the year of 2017, a Wednesday. The coordinates of each bus are ordered in time, resulting in a sampling of their paths, for every minute. We refer to this dataset as the *gathered dataset*.

Bus traces have one GPS-coordinate sample per minute per bus. A street segment may be crossed by a bus in less than one minute, hence, the GPS sampling rate does not allow the detection of all the street segments in the path of each bus. Figure 5.2a illustrates an example of this situation, where gps_1, gps_2, and gps_3 are the GPS coordinates of bus b. These coordinates do not describe the path of b in terms of the street segments of the road map.



(a) GPS traces of bus b.

(b) Reconstruction of the path of b using GPS traces.

Figure 5.2: Route reconstruction from GPS traces.

It is necessary to obtain the path of a bus in terms of all the edges in $\mathcal{E}$ that are part of the path of the bus, i.e., the street segments of the path. For this reason, we

use the process flow illustrated in Figure 5.3. We detail each step in the sequence of this text.

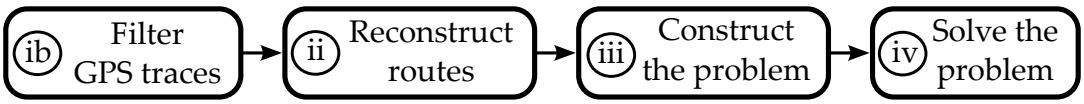


Figure 5.3: Processing flow used to study a real scenario.

The flow receives as input the GPS traces of each bus. In Step (*i*), we filter the gathered positions that are identified as noise. This procedure is necessary because GPS coordinates gathered from the buses are error-prone [58]. Using the GPS traces, we detect data points indicating that some buses are parked at bus garages, which generate data points even when they are not in service. Using this filter, the positions of the same bus path are ordered and, if the distance between two points is smaller than a certain threshold, we can filter the second position out. The first position is never removed and subsequent positions are compared only to points that are already inside the resulting dataset. We pick 10 m as the threshold because it is a typical error range of commercial GPS devices [58]. After this filtering phase, the buses have some positions removed from their paths, resulting in fewer positions as an input to the next procedure. In case a bus has all but one position removed from its path, the bus is considered as stopped and removed from the dataset.

In the course of a single minute, a bus can travel more than one street segment. This situation is illustrated in Figure 5.2a, with no GPS samples taken while the bus is in street segments (x_i, x_j) and (x_k, x_l). With this regard, simply mapping the gathered positions of a bus to the corresponding section may not reflect the complete coverage of this bus. In Step (*ii*), a routing algorithm reconstructs the most likely route followed by each bus, in terms of the street segments of the city. To perform these steps, we use the API Google Snap to Roads [59]. This API receives a list of ordered coordinates as the input, which is a sampling of the path traveled by a vehicle. It returns the most likely path followed by the vehicle according to the input coordinates, in the form of new coordinates, adjusted to the topology of the roads. In the case an input coordinate does not fall within a street segment, Google Snap to Roads approximates the input coordinates to the nearest street segment. Therefore, the only way GPS errors can affect the path estimation is if the input coordinates are closer to some street segment that is not the previous or the next segment in the path. Since GPS errors are about 10 m [58], it is reasonable to expect that the coordinates are precise enough for Google Snap to Roads to discover the street segment where the bus was originally traveling when its position was sampled. Google Snap to Roads also associates every new coordinate to a *place id*, which represents a Road Segment. Road Segments are data structures used by Google

Table 5.2: Attributes of the gathered and estimated datasets.

Attribute	**Value**	**Dataset**
Total gathered positions (#)	5,496,878	Gathered
Total buses in original set (#)	6,075	Gathered
Removed positions after filtering (#)	1,384,925	Gathered
Total positions after filtering (#)	4,111,953	Gathered
Removed buses after filtering (#)	328	Gathered
Total buses after filtering (#)	5,747	Gathered
Total positions after estimation (#)	52,250,671	Estimated
Total street segments (#)	95,992	Estimated
Sum of all street segment lengths (km)	5,655	Estimated
Total distance traveled (km)	1,005,327	Estimated

to define segments for their Road API [60]. The Road API is used to give drivers indications on routes and directions. Therefore, a Road Segment is a street segment of our model, since every point on a street is mapped into one, and just one, Road Segment. Figure 5.2 illustrates the processes through which samples of the path followed by bus b are transformed into a path and associated to street segments $(x_i, x_j), (x_j, x_k), (x_k, x_l), (x_l, x_m)$.

We refer to an adjusted coordinate associated to a road segment as an *adjusted position*. The dataset obtained after Step (*ii*) is called *estimated dataset*. Section 5.1.7 performs an analysis of the gathered and estimated datasets, detailing Step (*iii*) and Step (*iv*).

5.1.7 Data analysis

The objectives of gathering and processing real data from buses are to use it as the input to the problem formulated in Section 5.1.3. To understand the considered scenario, it is important to perform an analysis of the data in terms of its size, before and after Step (*ii*). Table 5.2 exhibits some attributes of the gathered and estimated datasets.

The gathered data contains 5,496,878 positions, obtained from 6,075 buses. The filter removes 1,384,925 positions and 328 buses (i.e., about 25% of positions and 5% of buses are removed); thus, we finally dispose of 4,111,953 positions and 5,747 buses. After these positions are applied to Google Snap to Roads, we obtain 52,250,671 estimated positions. The increase in the number of positions occurs because Google Snap to Roads estimates a smooth path for the vehicle. As a consequence, it may generate several positions between a pair of input coordinates. The estimated positions are distributed in 95,992 street segments. The sum of the estimated lengths

of all street segments is 5,655 km. According to the government of Rio de Janeiro, the city has a total of 10,577 km of streets. Hence, it is possible to estimate that the total fleet of buses is capable of covering 53 % of the streets.

In this thesis, the coverage obtained with a given number of buses is compared to the coverage of all the buses, using a relative coverage. We define the relative coverage as the coverage obtained divided by the coverage when all buses are equipped with Sensing nodes, expressed in percentages. In this way, it is possible to better evaluate the effect of the different budgets, in terms of Sensing nodes. Figure 5.4 illustrates the Complementary Cumulative Distribution Function (CCDF) of the estimated street segment length. We note that the majority of lengths lie between a few tens to a few thousands of meters. The street segments with the biggest lengths were checked manually, for consistency. The three biggest street segments range from 1,780 m to 2,146 m. Those correspond to freeways that cross the city.

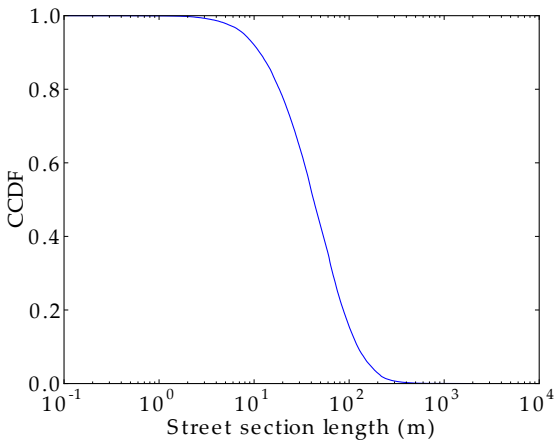


Figure 5.4: Distribution of estimated street segment lengths captured with Snap to Roads.

The Union of Bus Companies of Rio de Janeiro (i.e., *Rio Ônibus*) also makes available an estimate of the distance traveled by the fleet over the year [61]. We use the most recent records in relation to the dataset (Jan-Aug, 2016) to evaluate the obtained length estimations. According to the data, a bus travels an average of 218 km per day. In our estimation, the average bus travels 175 km on a day. Since we are performing a coverage analysis, it is safe to conclude that we make a pessimistic estimation. Therefore, it is possible to execute Step (*iii*) and construct the MILP with the dataset.

5.1.8 Experiment execution

After executing steps (*i*), (*ii*), and (*iii*), we can use data as the input to the coverage problem, completing Step (*iv*). The problem is solved using IBM CPLEX 12.5.1. This tool works finding solution candidates and upper limits to the solution, making

it possible to estimate the maximum gap between the best solution found and the best solution possible. Given the complexity of the problem, we configure the solver to return the best solution when the gap between the upper limit and the best solution found is within 2.0% of the upper limit. We choose a gap of 2.0% because it is the smallest gap we obtain for every tested value. We obtain this gap considering the best equipment available, an Intel Xeon E5-2650 with 264 GB of RAM. The values we use as the budget for the number of buses to be equipped with Sensing nodes (in the model, denoted by p) are 2, 4, 8, 16, 32, 64, 128, 256, 512, 1,024, 2,048, and 4,096. The total number of buses is 5,747.

5.1.9 Results

We show in Figure 5.5 the relative coverage for different budgets of Sensing nodes. As explained in Section 4.7.1, the relative coverage is calculated concerning the coverage when all buses are equipped with Sensing nodes. The horizontal axis is in logarithmic scale.

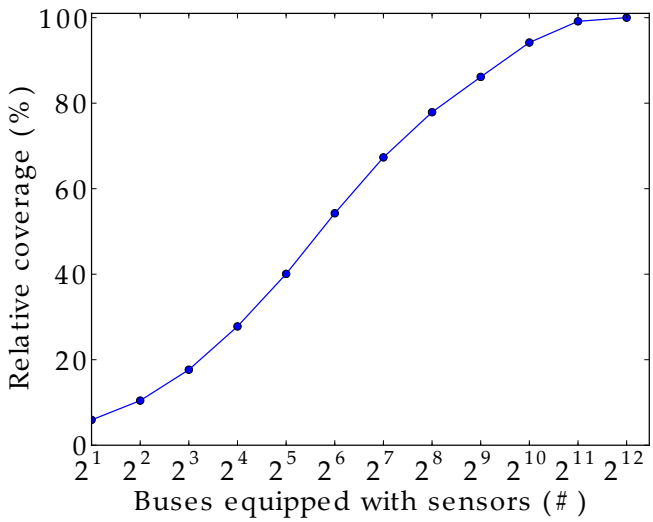
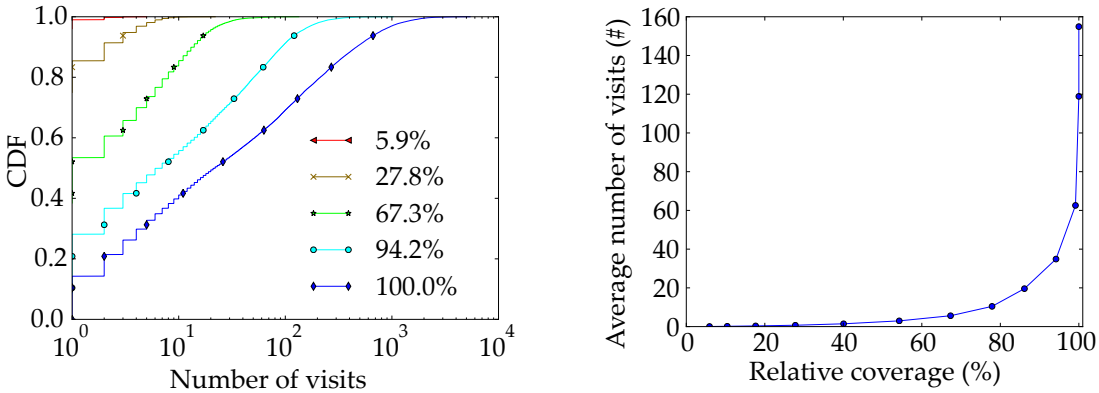


Figure 5.5: Relative coverage of the buses equipped with sensors.

We observe in the plot that 1,024 buses, or approximately 18% of the bus fleet, are able to cover at least 94% of the streets served by buses. This is equivalent to 5,060 km of streets. It is also worth noting that with 32 buses (2^5) we are able to cover about 40% of the total covered area. Hence, it is possible to use a so small subset of the buses to build a prototype of the complete service and yet cover 40% of the target area. This shows that it is possible to adopt an incremental deployment of such a sensing system.

The focus of the optimization problem is on maximizing the spatial coverage of the network. Nevertheless, mobile sensing poses a trade-off between spatial and



(a) CDF of the amount of times the same street segment is visited, for different coverage proportions.

(b) Average of times a street segment is visited, in function of the coverage proportion.

Figure 5.6: Visit number of street segments throughout a day.

time coverage [3]. Even though the area reached by each sensor grows with mobility, some areas are not sensed the whole time. To evaluate the time coverage, we count the number of times the same street segment is visited by any bus equipped with a Sensing node. The number of visits for a given street segment influences the likelihood of detecting some event of interest in this section or determines the freshness of sensed data. Additionally, a larger number of visits also improves the efficiency of algorithms for noise reduction and value prediction [40, 56].

To estimate the effects of the proposed optimization on the time coverage, we evaluate the number of times each street segment is visited in our dataset, for different values of the relative coverage. Figure 5.6a shows the CDF (Cumulative Distribution Function) of the number of visits of the same street segment for the relative coverage levels of 5.9%, 27.8%, 67.3%, 94.2%, and 100%. These relative coverage numbers correspond to the use of 2, 16, 128, 1,024, and 5,747 buses, respectively. The horizontal axis is on a logarithmic scale. It is possible to observe that when the relative coverage is 5.9%, the covered street segments get at most three visits, but when the relative coverage is 94.2%, more than 70% of the covered street segments are visited more than once.

Another way to visualize the information on time coverage is represented in Figure 5.6b. This figure shows the average number of visits received by each street segment in the course of a day as a function of all the relative coverage levels obtained in our experiments. We observe that the growth rate increases when relative coverage is around 50%. This result reinforces the idea that it is possible to incrementally deploy the sensing system. First, minimal service is established and few applications are supported, since some applications might need a greater number of visits per day; later, more equipment is integrated into the system, enabling new applications

and spreading the service to more areas of the city.

These results outline that a naive coverage metric is not able to account for the timeliness of data. In Section 3.1, we discuss the requirements of smart city applications in terms of the delivery delay and the frequency of data. In the next section, we refine the coverage metric proposed to account for these requirements.

5.2 A delay-aware coverage metric

Each bus in set $\mathcal{B}$ follows a fixed path through the street segments of the city. We take as an example the path P_b of bus $b \in \mathcal{B}$, illustrated in Figure 5.1. The path P_b can be represented as a sequence of the edges in the graph $\mathcal{G}$, $((x_i, x_j), (x_j, x_k), (x_k, x_l), (x_l, x_m) \ldots)$, where $P_b[n]$ is the nth edge reached by b and $(x_i, x_j), (x_j, x_k), (x_k, x_l), (x_l, x_m) \ldots \in \mathcal{E}$. While passing through a street segment, a bus gathers data and stores it until the bus reaches the next Fog node. When a connection is established, the bus delivers the data. The delivery delay of data gathered in street segment (x_i, x_j) is the time elapsed between the instant when the first data about this street segment is collected until the instant when data is delivered to a Fog node. We assume that vertex $x_l \in \mathcal{V}$ is in communication range of Fog node s_2. We also assume that x_l is the first vertex in communication range of a Fog node that b passes after passing by x_i, x_j, and x_k. The path of bus b is a sequence of vertices that includes the street segment (x_i, x_j). Later, when passing by vertex x_l, b delivers data to Fog node s_2. The time elapsed between the moment b reaches x_i until the moment b reaches x_l represents the delivery delay suffered by the data collected by b in (x_i, x_j). The bus b also collects data from the street segments (x_j, x_k) and (x_k, x_l), delivering this data when it reaches x_l. Similarly to the case of (x_i, x_j), the delay these pieces of data suffer is the time elapsed between the instant b reaches the first vertex of the street segment and the time b reaches x_l.

Several aspects might influence the delivery delay of data collected in a street segment (x_i, x_j). Among them, traffic conditions, the number of times a bus stops to serve passengers, and the distance between (x_i, x_j) and the gateway where data from (x_i, x_j) is delivered. As shown in Table 3.1, different applications can tolerate different data delivery delays. This means that, depending on the delivery delay, data collected on segment (x_i, x_j) may be useful or not for a given application. Since different applications may have different tolerances to delay, the same data may be useful to some applications and not for others.

As we show in Section 5.1.9, buses might sense a street segment (x_i, x_j) several times during a given interval of time T. This can happen either because (x_i, x_j) is in the path of more than one bus but also because some buses can pass over the same street segment several times during T. In this case, it is possible to say that there is

a certain measurement frequency of street segment (x_i, x_j). As shown in Table 3.1, applications might need a certain measurement frequency to provide a reasonable service to its users.

5.2.1 Constructing the covered set

A coverage metric must reflect application requirements in terms of delays and measurement frequency. To define the coverage for a given application, we use its maximum tolerated delay, $D_{\max}$, and its minimum tolerated measurement frequency, $F_{\min}$. We say that a bus $b \in \mathcal{B}$ has visited street segment $(x_i, x_j) \in \mathcal{E}$ if, after passing by (x_i, x_j) and collecting data, b is able to deliver the data before $D_{\max}$. We can define the visiting frequency $F_{(x_i,x_j)}$ of (x_i, x_j) as the number of times (x_i, x_j) has been visited by any bus in a given period T. A street segment (x_i, x_j) is *covered* if and only if its visiting frequency is greater than or equal to the minimum visiting frequency $F_{\min}$ required by the target application. It is possible, then, to define $\mathcal{E}_c$ as the subset of $\mathcal{E}$ containing all the covered street segments, which is evaluated in terms of $D_{\max}$ and $F_{\min}$. The set $\mathcal{E}_c$ and the set $\mathcal{E}_s$, defined in Section 5.1, are not the same. The set $\mathcal{E}_s$ contains the set of street segments that can be covered given that applications restrictions are satisfied with a single visit and data can be delivered with indefinite delays; $\mathcal{E}_c$ contains the street segments that can be covered given specific application requirements regarding measurement frequency and delivery delay.

Algorithm 2 Algorithm to construct the subset $\mathcal{E}_c$

Data: $\mathcal{P} = \{P_{b1}, \ldots, P_{bn}\}$, $D_{\min}$, $F_{\max}$
1: **visits_counters** $\leftarrow$ **0**
2: **for** $P_{bi} \in \mathcal{P}$ **do** ▷ For the bus path of every bus
3: m $\leftarrow$ 0
4: **while** $m \leq |P_{bi}|$ **do** ▷ For each street segment along the path of b_i
5: $delivery\_delay \leftarrow get\_delivery\_delay(P_{bi}, m)$
6: **if** $delivery\_delay \leq D_{\max}$ **then**
7: **visits_counters**$[P_{bi}[m]] \leftarrow$ **visits_counters**$[P_{bi}[m]] + 1$ ▷ Count the number of visits for each section
8: m $\leftarrow$ m+1
9: $\mathcal{E}_c \leftarrow \emptyset$
10: **for** $section \in \mathcal{E}$ **do** ▷ Verify the measurement frequency for each section
11: $measurement\_frequency \leftarrow$ **visits_counters**$[section]/T$
12: **if** $measurement\_frequency \geq F_{\min}$ **then**
13: $\mathcal{E}_c \leftarrow \mathcal{E}_c \cup section$
14: **return** $\mathcal{E}_c$

Algorithm 2 formalizes the construction of $\mathcal{E}_c$. The algorithm receives as inputs the set $\mathcal{P}$, the limit $D_{\max}$, and the limit $F_{\min}$. The set $\mathcal{P}$ contains all the paths from the buses. Each path is defined in terms of the street segments of the city, as discussed in Section 5.1.1. The limits $D_{\max}$ and $F_{\min}$ define, respectively, the

maximum delivery delay and the minimum measurement frequency tolerated by the considered application. The array **visits_counters** is an array indexed by street segment. In the while loop starting at Line 4, the visits of each segment are accumulated in **visits_counters**. The function *get_delivery_delay* receives as input a path and an index m in the path. The function returns the delivery delay for the mth street segment of the path. The for loop starting in Line 10 adds to the subset $\mathcal{E}_c$ the segments that were visited with at least the minimum measurement frequency $F_{\min}$. The algorithm returns the subset $\mathcal{E}_c$, containing the street segments that are covered for the considered application. This set is the final output of the algorithm.

Given the reasoning above and the construction of $\mathcal{E}_c$, Equation 5.10 defines the coverage C_c of a city for a given application as the sum of the lengths of street segments that are visited within a minimum visiting frequency, normalized by the sum of the length of all the street segments of the city. In Equation 5.10, $\mathcal{E}_c$ is the set of covered street segments and L is the sum of street segments lengths.

$$C_c = \sum_{(x_i,x_j)\in\mathcal{E}_c} \frac{l_{(x_i,x_j)}}{L}. \tag{5.10}$$

It is important to note that the coverage metric proposed is one-dimensional in space. Since we assume a multi-purpose MWSN, we do not infer the sensing range of each sensor. Therefore, we assume the sensing range as the width of each street that a bus passes by. As a consequence, coverage is related to the length of streets covered by each bus.

5.2.2 Experimental analysis

To show the feasibility of our delay-aware coverage metric, we apply the metric to a hypothetical network, derived from real mobility traces of the buses of the city of Rio de Janeiro. To obtain the traces, we use the same API used in Section 4.7 and Section 5.1.6. Additionally, we use a similar procedure to the one followed in Section 5.1.6. We collect the GPS traces, filter them, and adjust them to the topology of the street segments. Nevertheless, this is not enough to obtain a delay-aware coverage of the city. We must also obtain the delivery delay of each measurement, accessing the times when each bus can make contact with a gateway and, therefore, can deliver data. Finally, we can compute the coverage of the city, for different application requirements.

Figure 5.7 illustrates the procedure we follow to compute the coverage of the city to different applications and build a coverage map. In Step (*ia*), we split the city map into street segments and in Step (*ib*) we filter the GPS traces. Then we follow

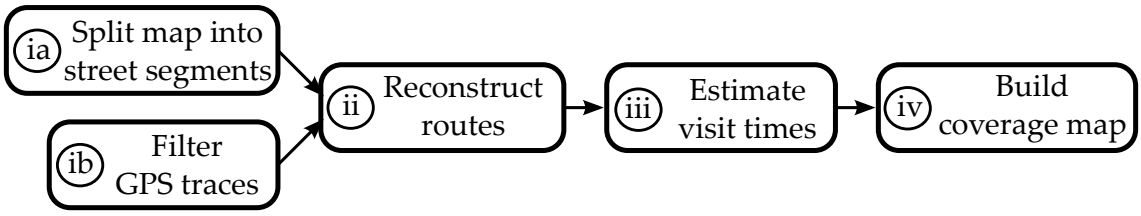


Figure 5.7: Procedure to construct the delay-aware coverage maps.

to Step (*ii*) and use the filtered GPS traces and the map to reconstruct the bus routes. These steps are similar to the steps followed in Section 5.1.6. In Step (*iii*) we estimate the visit times of each visited street segment. Then, in Step (*iv*), we can use the number of visits of each street segment to build a coverage map for different applications. In the sequence, we detail the steps of the procedure.

Data collection and processing

To divide the streets of Rio de Janeiro into street segments, we use a map provided by OpenStreetMap [62]. The area of Rio de Janeiro is selected from the map, using a square delimited by the coordinates (-23.07,-43.7) and (-22.78,-43.16). To select this region, we use the tool Osmium[1]. The map is a graph that uses the same model described in Section 5.1.1. Therefore, each edge of the graph is a street segment. The sum of all street segment lengths on the map is 13,852 km. This dataset is the output of Step (*ia*) in Figure 5.7.

To perform Step (*ib*) of the procedure, we employ the same API used in Section 5.1.6 and in Section 4.7. We collect 29,155,221 GPS positions of 5,706 buses during the week between November, 5th, 00:00, and November, 11th, 23:59, in the year of 2018. Since GPS coordinates are prone to errors, we eliminate inconsistent records. We consider that the records outside the square defined by the map and the records with timestamp out of the gathered period are inconsistent. Also, to reduce the cardinality of the dataset, we eliminate consecutive records of the same bus that differ from less than 10 meters. It is possible to discard these records because the difference between them is within the GPS error, i.e. 10 m [63]. After these filters, there are 19,979,537 records. Table 5.3 shows the number of active buses for each day of the week. A bus is considered active if there is at least one entry for this bus in the day. It is possible to note that the largest difference in the number of buses is between Wednesday and Sunday. This difference is of 216 buses, which represents only 4% of the buses. Based on this result, the dataset is not divided into weekdays and weekends.

We also analyze the effect of the time of day when the buses are operating. To do so, we compute the cumulative distance traveled by buses at each moment of the

[1] `https://osmcode.org`

Table 5.3: Number of active buses in the different weekdays.

Day of the week	**Number of buses**
Monday	5,540
Tuesday	5,546
Wednesday	5,573
Thursday	5,563
Friday	5,547
Saturday	5,435
Sunday	5,357

day. Figure 5.8 shows the result of the total distance traveled as a function of the time of the day. It is possible to observe that the distance traveled is close to zero between 0h and 4h. Therefore, we choose to eliminate this interval of the day of our analysis, trusting that the buses are not capable of collecting data during this interval.

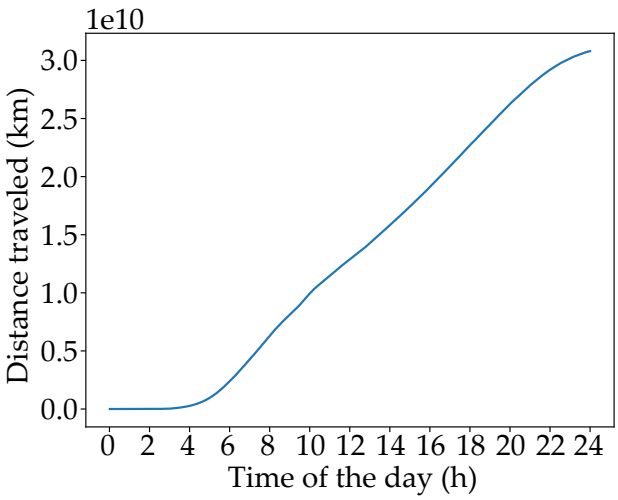


Figure 5.8: Cumulative distance traveled on each time of the day.

Bus traces have one GPS coordinate sample per minute per bus. As illustrated in Figure 5.2a, the GPS sampling rate does not allow the detection of all the street segments in the path of each bus. To complete Step (*ii*) of Figure 5.7, it is necessary to obtain the path of each bus in terms of all the edges in $\mathcal{G}$ that are part of the path of the bus, i.e., the street segments of the path. We use the matching function on Open Source Routing Machine (OSRM) [64]. This function returns the most likely route followed by a vehicle, from a sequence of GPS coordinates. OSRM returns a route as a sequence of vertices of the map. This way, we obtain the paths of the buses as a list of all the vertices where the buses pass by. With this list, we can derive

the street segments visited by each bus. Figure 5.2 illustrates the transformation of GPS samples into the bus path in terms of map vertices.

In Section 5.1.6 we follow a similar procedure to perform Step (*ii*), but in Section 5.1.6 we use the service Google Snap to Roads. We believe OSRM is more suited to the present procedure, for two main reasons. The first reason is that OSRM uses a public map, returning vertices of the map. This makes it easier to keep track of the street segments of the city. The second reason is that OSRM is an open software, allowing us to have more control over the data processing.

After obtaining the paths in terms of map vertices, we must also associate to every edge (xi, x_j) in a path P_b, the instant when b passes by this edge. It is expected that between two consecutive GPS positions, the route generated by OSRM consists of many vertices. Since OSRM identifies the vertices that correspond to a given GPS position in the matching, we associate the timestamp t of the GPS position to the corresponding vertex returned by OSRM. Then, we use interpolation to associate an instant to the other vertices, employing the time between vertices as weights. In other words, we assume the bus traveled at a constant speed between those two points and derive the arrival instant on each vertex.

To determine the time at which a bus can start delivering data, it is necessary to detect the instants when a bus is in contact with a gateway, completing Step (*iii*). Hence, we define that a bus can deliver data when it reaches a vertex that is within the communication range of a bus stop. We define that a node is within the communication range of a bus stop when it is at a distance of 10 m or less from this bus stop. Therefore, the bus can deliver data when it is within the communication range of at least one bus stop. After the processing, it is possible to build the set $\mathcal{E}_c$ and, finally, evaluate the coverage for different applications.

Coverage analysis

With the data obtained in Section 5.2.2, we perform Step (*iv*), building a coverage map of the network. Using different values for the maximum delays $D_{\max}$, we can build an abacus of the network coverage. Figure 5.9 shows the coverage of Rio de Janeiro as a function of $F_{\min}$, for different maximum delays $D_{\max}$ of 12 s, 120 s, 300 s, 600 s, 1,800 s, and 72,000 s. These values represent the applications in Table 3.1, but other values of maximum delay are added to represent applications that do not have a delay defined in the literature. The case where $D_{\max}$ is unrestricted (i.e., $72,000$ s) and $F_{\min}$ is minimum (i.e, one – 10^0) is equivalent to the coverage proposed in Section 5.1. To calculate $F_{(x_i,x_j)}$, we count the number of visits received by (x_i, x_j) and divide it by the period T. The period used to calculate the measurement frequency is of 20 h, since it is the period considered in the traces. If we observe the less restricted case, where $D_{\max} = 72,000$ s and $F_{\min} = 1$ measurement per day, we

can note that the coverage is 43.7% of the streets. It is also possible to observe that, for $F_{\min} = 1$ measurement per day, the coverage value for $D_{\max} = 120\,\text{s}$ more than doubles when compared to $D_{\max} = 12\,\text{s}$. In every observed case, a change in $D_{\max}$ or in $F_{\min}$ implies in a coverage change, as a consequence. These results demonstrate the importance of considering the delivery delay when estimating the coverage of such network.

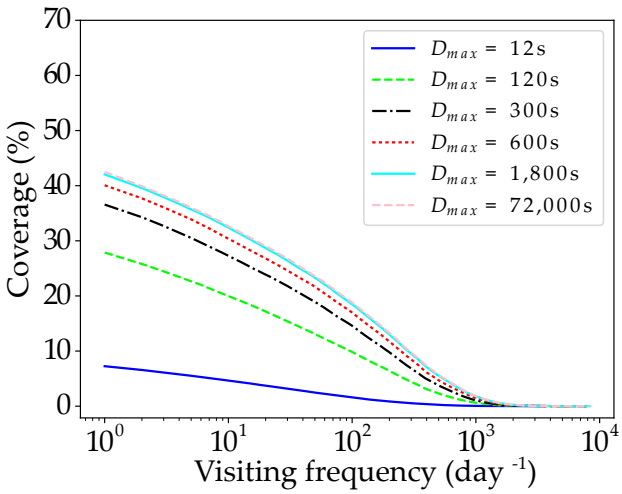


Figure 5.9: Abacus of the coverage of Rio de Janeiro in function of $F_{(x_i,x_j)}$, for different $D_{\max}$ over one week.

As shown in Table 3.1, a $D_{\min}$ of 300 s corresponds to the applications of air quality and noise monitoring, while a $D_{\min}$ of 1,800 s represents an application of waste management. Figure 5.10 illustrates the coverage for the central region of Rio de Janeiro, for the applications of waste management, air quality monitoring, and noise monitoring. The area illustrated in Figure 5.10 has 22.26 km^2, representing about 1.86% of the total area of Rio de Janeiro. The streets in blue are the coverage of the noise monitoring application. Since the air quality monitoring application is less restrictive than the noise monitoring, the coverage of this application is equal to the coverage of the noise monitoring application plus the street segments in green. The application of waste management is even less restrictive than the application of air quality monitoring. Its coverage is the coverage of air quality monitoring plus the street segments in red. The street segments in gray could not be covered.

Using the delay and frequency thresholds in Table 3.1, it is possible to define the coverage for different applications for smart cities. Table 5.4 shows the coverage for some of these applications. This table and Figure 5.10 show the differences in coverage obtained with different real-world requirements. We can observe in Table 5.4 a difference of up to 15.9 percentage points in the coverage of the analyzed applications. In other words, the coverage more than doubles when we compare the applications of noise monitoring and waste management.

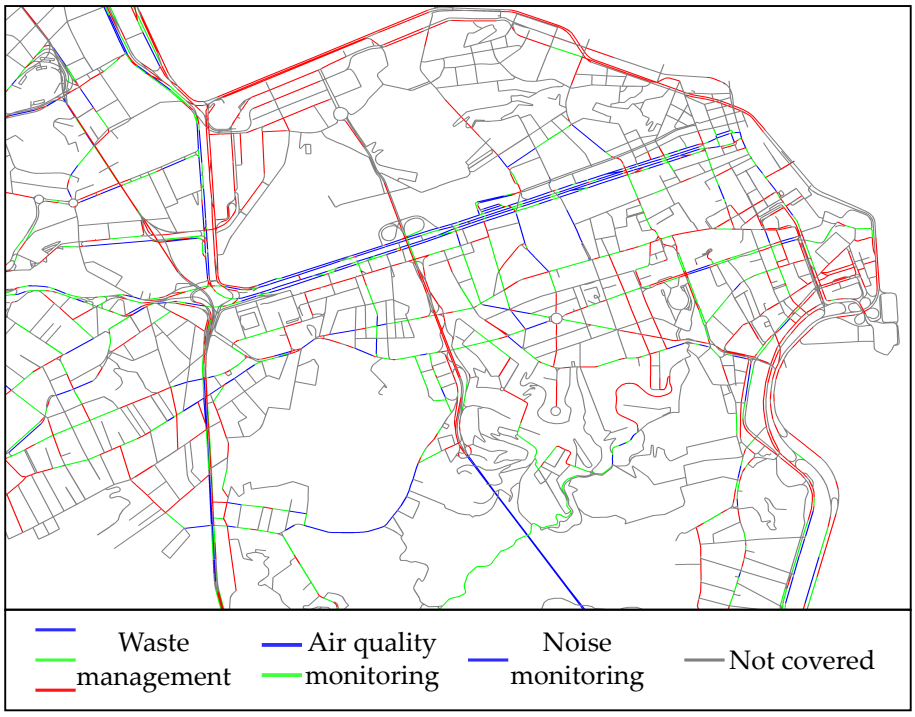


Figure 5.10: Coverage of the central region of Rio de Janeiro for different smart city applications.

Table 5.4: Coverage obtained by different Smart city applications.

Application	**Coverage (%)**
Waste management	27.9
Air quality monitoring	19.3
Noise monitoring	12.0

In this work, we rebuild bus routes from GPS traces to estimate the coverage of a bus-based network. An important remark is that Meegahapola *et al.* and Liu *et al.* show that it is possible to rebuild bus routes in Singapore and London, even when GPS traces are not available [65–67]. This means that, with some adjustments, the methods used in this thesis can be replicated to these cities. Since our coverage metric considers the delivery delay, it is expected that buses can achieve better coverage in cities with higher bus density and faster traffic.

5.2.3 Comparison with another coverage metric

There are many coverage metrics that do not consider the maximum delivery delay $D_{\max}$ and the minimum measurement frequency $F_{\min}$ tolerated by each application. The coverage metric proposed by Ali and Dyo is an example of this situation [12]. Their metric counts the number of streets that are visited at least once by any bus.

Therefore, in addition to not including $D_{\max}$ and $F_{\min}$, the length of street segments are also not considered. Even though it appears very similar to our proposal, not considering $D_{\max}$ and $F_{\min}$ creates a significant difference. To show the importance of considering $D_{\max}$ and $F_{\min}$, we apply the coverage metric by Ali and Dyo to the dataset obtained in Section 5.2.2. After estimating the coverage by the original model proposed by Ali and Dyo, we adapt their coverage metric to take into consideration $D_{\max}$ and $F_{\min}$.

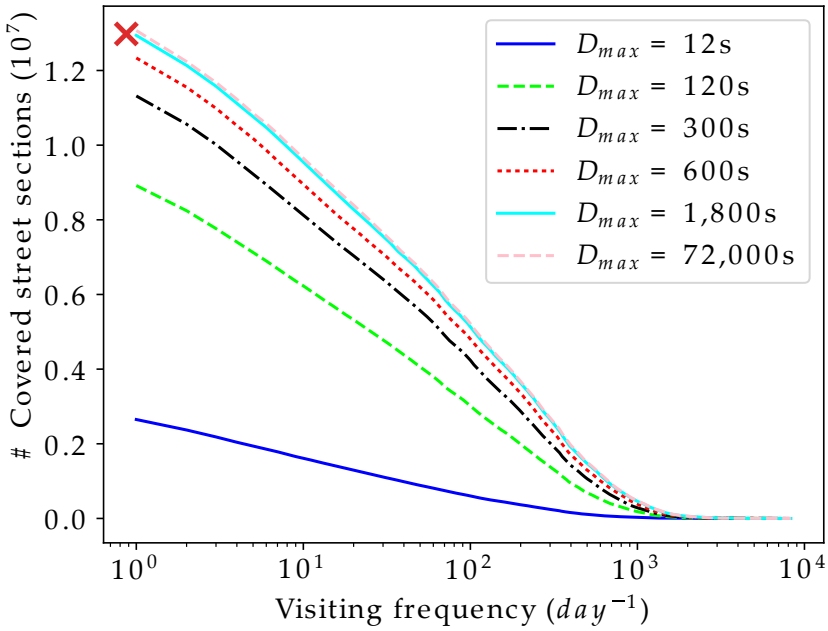


Figure 5.11: Abacus of a coverage metric disregarding street segment lenght for Rio de Janeiro in function of $F_{(x_i,x_j)}$, for different $D_{\max}$ over one week. The metric proposed by Ali and Dyo is marked with a red "×".

Figure 5.11 shows the coverage when we consider that street sections have equal importance, ignoring their lengths. This is coverage as estimated by Ali and Dyo. We also plot the coverage values when different restrictions on $D_{\max}$ and $F_{\min}$ are imposed, to show the impact of considering these dimensions. The coverage obtained by Ali and Dyo, signaled by an "×" symbol, is the unrestricted case. In our dataset, this means $D_{\max} = 72{,}000\,\mathrm{s}$ and $F_{\min} = 1$. It is possible to observe that the coverage obtained by Ali and Dyo overestimates the coverage to any application whose restrictions are tighter than $D_{\max} = 72{,}000\,\mathrm{s}$ and $F_{\min} = 1$. This result further stresses the importance of considering $D_{\max}$ and $F_{\min}$ when estimating coverage, even in the case where the coverage metric ignores the length of streets.

5.2.4 Comparison with the static case

The goal of using mobile sensors is to achieve better spatial coverage for certain applications. To quantify the coverage gain, we want to analyze the coverage obtained by leveraging bus mobility and compare it with the coverage obtained by a hypothetical static scenario. In the case of bus-based mobility, sensors cover each street segment from its beginning to its end. This is not the case for static sensors. Therefore, it is important to use a method capable of considering the coverage of fractions of street segments. In this static scenario, bus stops are Fog nodes and sensors are placed within their communication range. We consider as covered a fraction of street segment that is inside the communication range of at least one Fog node. The total coverage is the sum of the lengths of all covered fractions.

To evaluate the static coverage, we consider that it is possible to place static sensors anywhere in the communication range of Fog nodes. Therefore, any piece of street segment inside the communication coverage of a Fog node is also covered by sensors. To evaluate this coverage, we treat each Fog node as a circle of radius equal to the communication range. Figure 5.12 illustrates the coverage by static sensors placed in the communication range of Fog nodes. The union of the circles represents the area in the city where communication with at least one Fog node is possible. Formally, street segments can be represented as line segments, such that the union of these line segments is the total road map of the city. In this static coverage metric, the total sensing coverage is the intersection between the area where communication with Fog nodes is possible and the road map of the city. Our evaluations show a total static coverage of 1.7% of the total roads on the map. The static coverage obtained is equivalent to about 242 km of streets.

The static scenario assumes that static Sensing nodes are placed within the communication range of the Fog nodes. This means that the network has the same infrastructure of Fog nodes plus the Sensing nodes to achieve the sensing coverage of the network covered areas.

Figure 5.13 shows the coverage gain obtained by the bus-based mobility of sensors over the course of a week. We define the coverage gain as the coverage obtained by the bus-based MWSN divided by the static coverage. It is possible to note that the coverage gain for an application of waste management ($D_{\max} = 1{,}800\,\mathrm{s}$, $F_{\min} = 24$ per day – approx. $10^{1.4}$) is more than 16.4 times, while for the applications of air quality ($D_{\max} = 300\,\mathrm{s}$, $F_{\min} = 48$ per day – approx. $10^{1.7}$) and noise monitoring ($D_{\max} = 300\,\mathrm{s}$, $F_{\min} = 144$ per day – approx. $10^{2.2}$) are 11.6 and 7.6, respectively. Table 5.5 shows, for each $D_{\max}$ analyzed, the maximum $F_{\min}$ that could benefit from the mobility, when compared to the static scenario. In other words, it shows the tightest application requirements that could be satisfied by the mobile network and

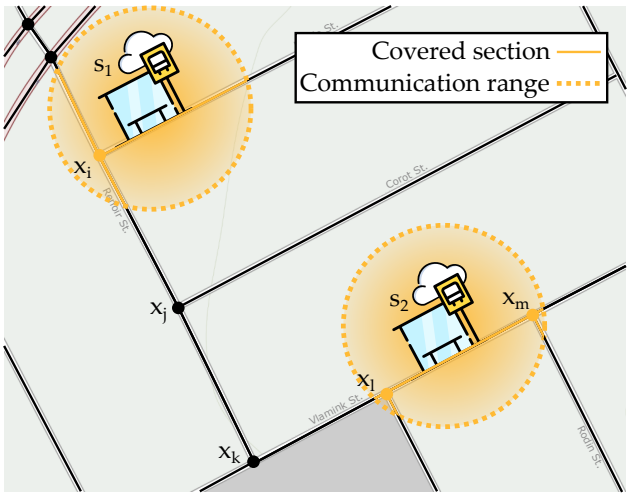


Figure 5.12: Example of covered streets by static sensors placed within communication range the gateways.

Table 5.5: Applications requirements limits to benefit from the mobile scenario.

D_{max} (s)	F_{min} (day^{-1})
12	105
120	715
300	960
600	1110
1,800	1203
72,000	1219

still improve coverage.

In this section, we have shown that the coverage of smart city applications can largely benefit from the mobility provided by buses. We have also shown that the coverage of a bus-based network is related to the requirements of each application. As a consequence, buses offer different coverage to different applications. In the next section, we study whether different buses contribute differently to the coverage of an MWSN.

5.3 Per-vehicle coverage analysis

Quantifying individual bus contributions can help to identify which buses are critical to the sensing tasks. This information allows operators to deploy a more efficient network by, for example, dismissing some buses from sensing tasks. Therefore, it is possible to lower the global costs in terms of sensing equipment, data transmission, and data mining. To help to achieve this goal, we propose a metric to quantify

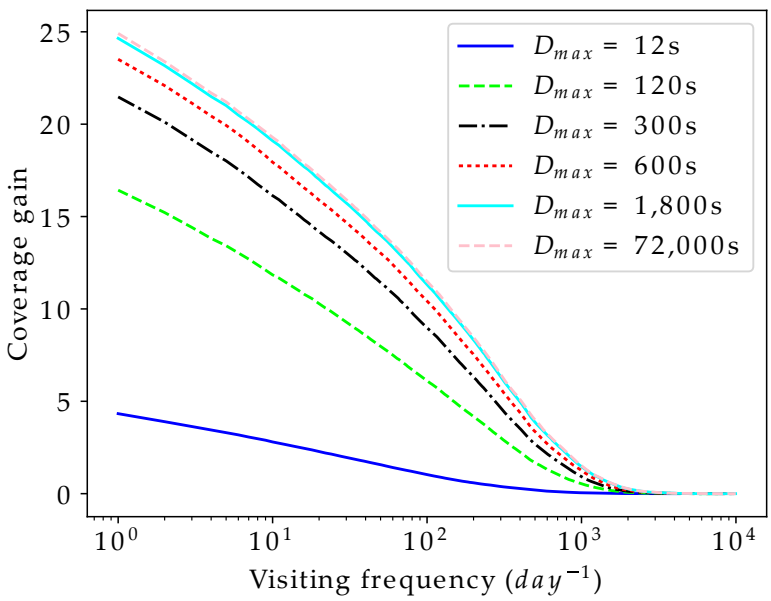


Figure 5.13: Coverage gain by buses of Rio de Janeiro in comparison to a static network, in function of $F_{(x_i,x_j)}$, for different $D_{\max}$ over one week.

such contribution, verifying if the coverage is more influenced by the contributing bus or the application. Again, we apply this metric to real GPS traces from the buses in the city of Rio de Janeiro to rank each bus based on its importance to the applications of waste management, air quality, and noise monitoring.

5.3.1 Coverage contribution metric

We denote $\mathcal{B}$ the set of buses of the city and $\mathcal{A}$ the set of applications served by the MWSN. Each bus $b \in \mathcal{B}$ may follow a different path or face different traffic conditions, while each application $a \in \mathcal{A}$ has its requirements. As already mentioned in the previous sections, the notations used in this chapter are in Table 5.1.

Using the coverage model defined in Section 5.2, we define the coverage contribution of $b \in \mathcal{B}$ for a single application $a \in \mathcal{A}$ as:

$$K_a^b = \frac{C_a^{\mathcal{B}} - C_a^{\{\mathcal{B}-b\}}}{C_a^{\mathcal{B}}}, \tag{5.11}$$

where K_a^b is the individual coverage contribution of bus b to the application a, $C_a^{\mathcal{B}}$ is the coverage obtained for application a using all the buses in $\mathcal{B}$, and $C_a^{\{\mathcal{B}-b\}}$ is the coverage obtained by the MWSN when b is removed from the network. The coverage metric presented in Equation 5.10 is $C_a^{\mathcal{B}}$, and $C_a^{\{\mathcal{B}-b\}}$ is the same coverage, obtained excluding b from sensing tasks. Note that the contribution metric is a real number between 0 and 1.

To decide which buses of the MWSN are the most or least important for a given

application, we rank buses according to their contributions. The effectiveness of an MWSN increases when it can serve several applications at the same time. In this case, the problem is that these applications have different requirements in terms of collected data. Therefore, we expect that the same bus has different contributions depending on the application. This means that some buses can be dismissed from the sensing task for some applications, but must remain in the network because they are required by some other applications.

5.3.2 Data-driven analysis

We use the same scenario to verify the variability of bus contributions with respect to different applications. We consider the same three applications as before, namely waste management, air quality monitoring, and noise monitoring.

To calculate the coverage of each application, we employ the same procedure described in Section 5.2.2. We also use the same map and the same API used in the previous sections to obtain bus mobility traces and to assess their contacts with Fog nodes. We collect data from November 1st to November 30th, 2018, obtaining GPS coordinates generated by 5,856 buses. Then, we proceed with steps (*ib*), (*ii*), (*iii*), and (*iv*) of Figure 5.7.

Coverage contribution

Using the data obtained in Step (*iii*), it is possible to evaluate the coverage obtained by all the buses or by a subset of them for each considered application. More specifically, we calculate $C_a^{\{\mathcal{B}-b\}}$, the coverage for an application a when bus b is not present, for every bus and every application for each day of our dataset. As a reference, during the evaluated period, the set of all buses can cover 50.0% of the city for an application with restrictions of $D_{\max} = 72{,}000\,\text{s}$ and $F_{\min} = 1$ (the least restrictive possible in our dataset). This represents a total of 6,929 km of streets.

Figure 5.14 illustrates the cumulative distribution of bus contributions for the applications of waste management, air quality, and noise monitoring, obtained using Equation 5.11 for each application, for each day. We then obtain the coverage average contribution over the days in the dataset. Figure 5.14 also illustrates the cumulative distribution of the average contribution of each bus over the three applications. The graph shows that the curve for the average contribution has much fewer buses with very low contributions than the curves for each application. It also shows that distributions depend on the applications, suggesting that applications are relevant to the contribution.

We also obtain the average speed of buses and the total coverage for each day. Considering the 30 days, the Pearson coefficient of the correlation between average

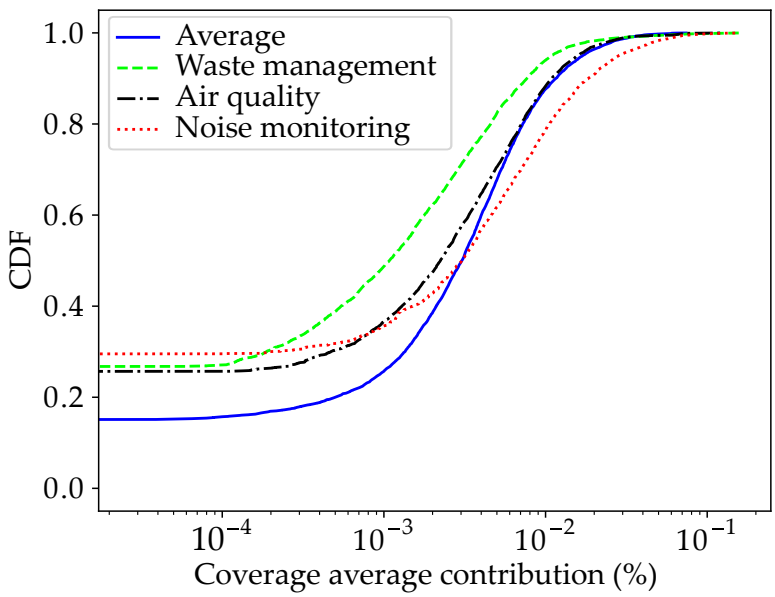


Figure 5.14: CDF of the bus contributions K_a^b from Rio de Janeiro.

speed and coverage is 0.834, 0.878, and 0.882 for the applications of waste management (i.e., the least restrictive one), air quality, and noise monitoring (i.e., the most restrictive one), respectively. A less restrictive application, with $F_{\min} = 1/\text{day}$ and $D_{\max} = 24\,h$ would have a Pearson coefficient of 0.788. The Pearson coefficient measures the correlation between two variables, assuming value 1 when they are perfectly correlated, 0 when they are uncorrelated and -1 when they are inversely correlated. This indicates that more restrictive applications receive more influence from the traffic conditions.

We know from Section 5.1.9 that city coverage is not spatially homogeneous. To measure this effect on the contributions of the buses, we divide buses into two groups: downtown buses, which visit downtown at least once, and suburban buses, which never visit downtown. We define the downtown as the rectangle inside the coordinates (-22.915651,-43.209736) and (-22.888928,-43.170168). Figure 5.15 shows, for each day, the average contribution of each group, for each considered application. It also shows, on the right axis, the ratio between suburban and downtown buses. We indicate Sundays and holidays with an "×" mark. We note that, when the number of downtown buses decreases, they are more important.

5.3.3 Contribution ranking

To quantify the impact of the application to the contribution, in the next experiment, we rank the buses for each application by ordering their coverage contributions. We also evaluate the average of their contributions for the considered applications. After that, we measure the ranking difference of the same bus, for all of the considered

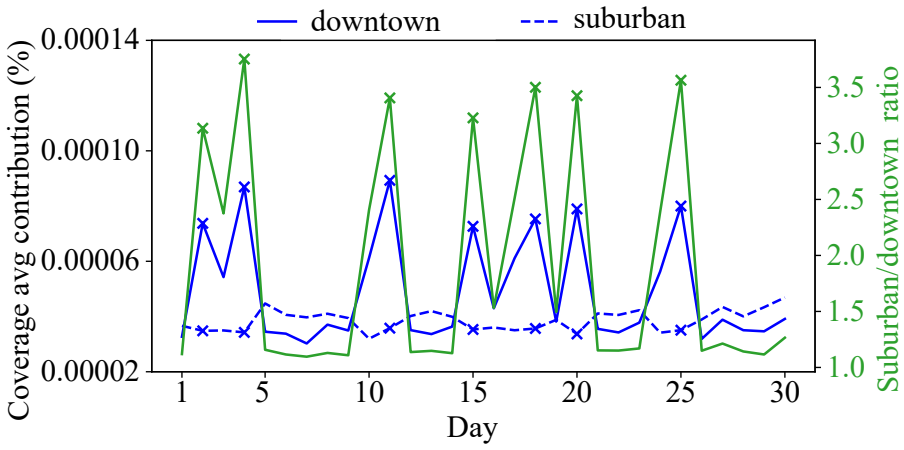


Figure 5.15: Average of the average coverage contributions K_a^b from Rio de Janeiro, for region and bus proportion between downtown and suburbs.

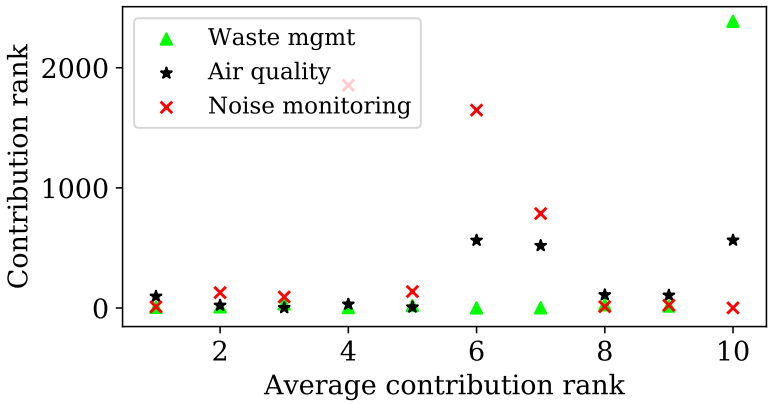


Figure 5.16: Bus contribution rank as a function of average contribution rank, for different applications, for the 10 largest average contributions.

applications and for their average contributions.

Figure 5.16 shows, on the horizontal axis, the rank of each bus when ranked by their average contribution, for the top 10 average contributors. In the vertical axis, Figure 5.16 shows the contribution rank for the analyzed applications. Therefore, every vertical section represents the ranks of the same bus. Buses represented more to the left have a higher average contribution. We note that the most relevant buses tend to have high contributions for every application, but this tendency does not hold for all of the 10 most relevant buses. The 10th bus in the average ranking is in $2,387^{th}$, 564th, and 1st places for waste management, air quality, and noise monitoring, respectively, out of 5,856 buses.

To study the whole dataset, Figure 5.17 extends the information in Figure 5.16 for all buses in the dataset. It is possible to observe that, except for the buses with the lowest average contribution, there is low stability of positions between rankings. This means that, in our dataset, there are not many buses that are important

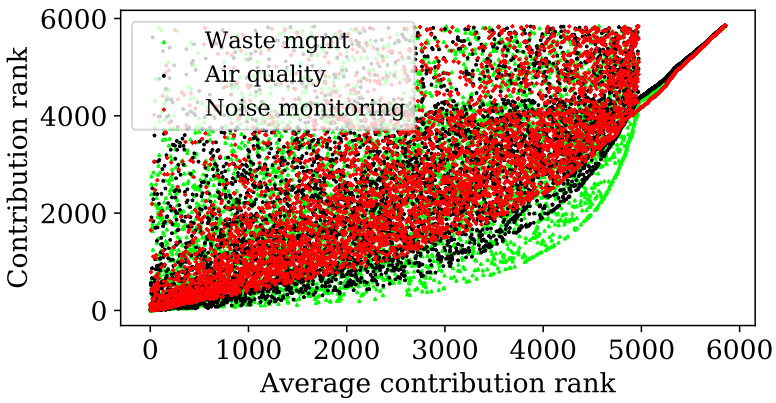


Figure 5.17: Bus contribution rank as a function of average contribution rank, for different applications.

Table 5.6: Kendall coefficient of bus contributions ranking.

Application	**Kendall coefficient**
Waste management × Air quality	0.21
Waste management × Noise monitoring	0.20
Air quality × Noise monitoring	0.17

to every application, but there is a significant number of buses that have a low contribution to all the applications. To quantify the stability between rankings, Table 5.6 shows the Kendall correlation coefficients of the contribution rankings of the considered applications. The Kendall coefficient is used to compare sequences. A Kendall coefficient of 1 means that sequences are perfectly correlated and thus independent of the application, while a coefficient of 0 means that rankings are uncorrelated. The small values of Kendall coefficients confirm that the applications have a strong influence over bus importance.

Figure 5.18 shows the cumulative distribution of the ranking difference of the same bus between its position in the contribution rankings of different applications. We observe that, in every case, more than 40% of the 5,856 buses change at least 1,000 places in the ranking. This means that the proposed contribution metric is highly related to the target application. Moreover, some changes in the application requirements can also change the way buses contribute to the coverage. The initial slope of the ranking difference between air quality and noise monitoring applications indicates that the contribution of buses for these applications is closer than the contributions for waste management. This trend is confirmed by Table 5.6. The proposed metric made it possible to discover and quantify the difference in each bus importance to the city coverage.

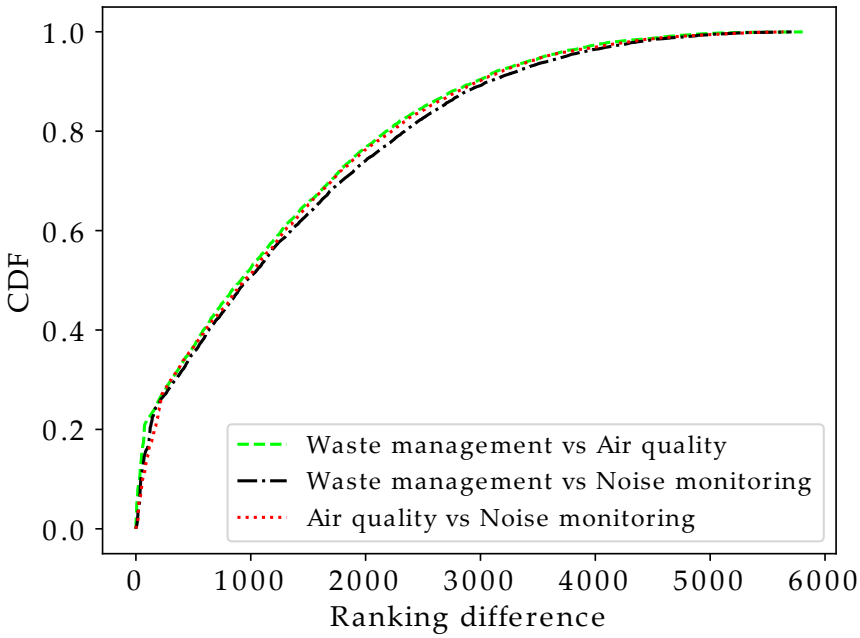


Figure 5.18: Contribution ranking difference of the same bus for an application compared to its contribution rank for another application.

5.4 Remarks

In this chapter, we have explored the coverage of a bus-based Mobile Wireless Sensor Network. First, we proposed a coverage metric that considers that a street segment is covered if any bus with a Sensing node passes by it. For this metric, we proposed a MILP to maximize coverage when the number of Sensing nodes is limited. We showed that only 32 buses can cover at least 40% of the total covered by all buses. We also showed that street segments are measured a different number of times. This last finding, together with our findings in Chapter 4, motivated us to propose another coverage metric.

We proposed a coverage metric that considers the maximum delivery delay and the minimum frequency of measurement of each street segment. We showed that coverage is related to each application. We also showed that buses contribute differently to the coverage of different applications.

Chapter 6

The SensingBus Prototype

We build a prototype for the SensingBus architecture, described in Chapter 3, to confirm its feasibility and scalability. The next sections explain in detail the different parts of the prototype. Additionally, the source code and documentation are available at `https://github.com/pedrocruz/sensing_bus`. The software is registered[1] and licensed under Apache License 2.0[2].

6.1 Sensing nodes

Sensing nodes employ an Arduino UNO as Controller, due to its low cost. The Wireless Interface is an ESP8266, a programmable micro-controller with a IEEE 802.11b/g/n interface. Table 6.1 lists the hardware used in the Sensing node prototype.

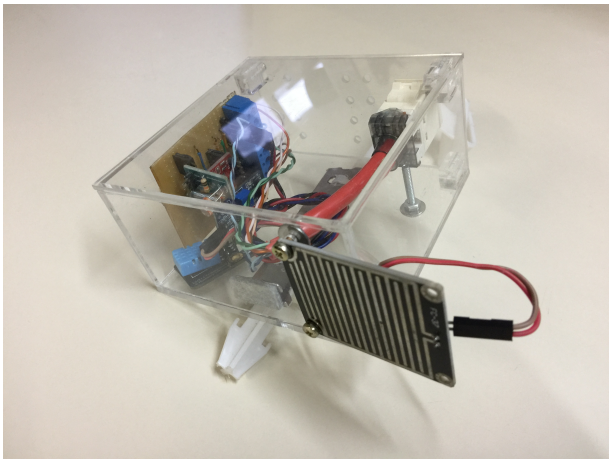
Physically, the Sensing node is composed of two parts: one is kept inside the bus and the other is installed outside the vehicle. This division is performed because some modules must be exposed to the environment, such as sensors and transmitters, while other components must be protected from harsh environmental conditions. For example, rain can harm some electronic equipment, but a rain sensor must be exposed to the rain to measure it. The part inside the vehicle contains the Controller, GPS Receiver, and Persistent Memory. The GPS Receiver and Persistent Memory share the same board. The external part contains the Wireless Interface, GPS Receiver antenna, and Sensor Bank.

The external part is protected by an acrylic case, measuring $7 \times 7 \times 4\,\text{cm}$, to be placed on the rooftop of the bus. The protective case has a transparent surface and some holes, allowing sensors to effectively sense the environment. Figure 6.1 shows the internal and external parts of the Sensing node prototype. We test the effect of the protective case and the effect of the mobility in Section 6.4.

[1] process number BR512019002626-8
[2] https://www.apache.org/licenses/LICENSE-2.0

Table 6.1: Equipment and software used in the SensingBus prototype.

Module	**Equipment**	**Manufacturer**
Controller	Arduino UNO R3	Arduino
GPS Receiver	GS-96U7	Guangzhou Xintu
Persistent Memory	GS-96U7	Guangzhou Xintu
Wireless Interface	ESP8266	Espressif
Sensor Bank	Humidity DHT11	DFRobot
	Temperature DHT11	DFRobot
	Light Intensity GL5528	GBK Robotics
	Rain Intensity GL5528	GBK Robotics
	Barometric Pressure BMP180	Sparkfun
	Temperature BMP180	Sparkfun



(a) External part of the Sensing node.

(b) Internal part of the Sensing node.

Figure 6.1: The external and internal parts of the Sensing node.

The following functionalities are currently implemented in the Sensing nodes:

- **Data gathering:** the sensors in the Sensor Bank gather data about the environment;
- **Data temporary storage:** the Persistent Memory stores data until a connection with a Fog node is established;
- **Data delivery:** the Wireless Interface data delivers data to a Fog node when a connection is available.

The Controller queries the sensors in the Sensor Bank and the GPS Receiver every second. Every query generates a total of 53 bytes, that are stored in the Persistent Memory.

Taking advantage of the programmable Wireless Interface, we implement a simple protocol for the communication between the Controller and the Wireless Inter-

face. In this protocol, the Controller asks periodically if there is a connection and the Wireless Interface answers. When the answer is positive, the Controller sends enough data to fill the buffer of the Wireless Interface, waits for data to be sent and initializes a new iteration, asking again if there is a connection. We implement the code for the Arduino UNO and the ESP8266[3].

As an implementation issue, the SRAM of Arduino Uno is limited. We take advantage of the fact that application-level messages have constant values per node, such as a header, indicating the sensors installed in the bank and the id of the sensor. We treat all the constants as strings and store them in the flash memory, avoiding keeping them in the SRAM. Another important implementation issue is related to the very little RAM of ESP8266. It is thus expected that the data stored in the Persistent Memory is several times bigger than the RAM available in Arduino and ESP8266. This means that Arduino must send several chunks of data to the ESP8266 until the data in the Persistent Memory is entirely copied. The ESP8266 must then flush every chunk to the Fog node before it can receive more data. In our first implementation on ESP8266, it would receive a chunk from the Controller, open a TCP (Transmission Control Protocol) connection, send the chunk of data and close it right after. Every chunk of data would open a new TCP connection. Even though this approach saves lines of code, every new connection creates new objects that are not discarded immediately when the connection closes, because of TCP's TIME_-WAIT. Since many connections would remain open, the memory of ESP8266 would be full with the pending connections, causing instability to ESP8266. To solve this issue, the connection status is kept and ESP8266 performs several requests over the same connection.

The Wireless Interface holds the SSID (Service Set Identification) and password of a WPA2-protected network created by the Fog nodes. The Wireless Interface searches periodically for a wireless network with a pre-configured SSID, and answers the queries from the Controller. After the Wireless Interface signals the Controller about a new connection, the Controller only sends application-specific data to the Wireless Interface. The Wireless Interface prepares the HTTP (Hypertext Transfer Protocol) headers and executes a POST method to the Fog node. We performed preliminary experiments and noticed data loss when the Controller would send more than 2 kB of data to the Wireless Interface. Therefore, every POST holds about 2 kB of sensed data. Each POST also holds 80 bytes of application header and about 133 bytes on HTTP headers. Also according to our preliminary experiments, preparing HTTP headers in the Wireless Interface is two times faster than preparing the HTTP headers inside the Controller. The codes used in this test are found in

[3] `https://github.com/pedrocruz/sensing_bus/tree/master/sensing`

the folder "sensing/tests" of the Github repository[4].

6.2 Fog nodes

Table 6.2 shows the equipment we use to build the Fog node. We chose the Raspberry Pi II Model B as Controller because of its computing power, size, costs, and available interfaces. We chose the WiPi as the Wireless Interface due to its costs and because of the number of simultaneous connections that it supports. Some other options with greater communication range were considered but discarded, because of the low number of simultaneous connections supported. The effect of simultaneous connections is further discussed in the Section 6.4.2.

The Wireless Interface of Fog nodes acts as an IEEE 802.11 access point protected with WPA2 (Wi-Fi Protected Access 2). The SSID and password are configured into the Controller. In the Controller, an HTTP server implemented in Python serves incoming requests and pre-process data before sending it to the Cloud node.

Table 6.2: Equipment and software used in the Fog node prototype.

Module	**Equipment**	**Manufacturer**
Controller	Raspberry Pi II B	Raspberry Pi Foundation
Wireless Interface	WiPi	Element14

Fog nodes have the following pre-processing functions currently implemented:

- **Error detection:** data is checked for inconsistencies and defective data is discarded;
- **Data concentration:** when several buses simultaneously connect to the fog node, their data is stored and sent periodically, reducing the number of connections to the Cloud level;
- **Data compression:** data is compressed before it is sent, reducing the traffic to the Internet.

For every request, the Fog node Controller checks data and discards inconsistent measurements. After that, data is accumulated in a concentration queue. At regular intervals, data is compressed and sent to the Cloud node. To send data, the Fog node acts as an HTTPS (Hypertext Transfer Protocol Secure) client, getting authorization by presenting its certificate to the Cloud node.

[4] `https://github.com/pedrocruz/sensing_bus/tree/master/sensing/tests`

6.3 Cloud node

The prototype of the Cloud node is implemented in software. Table 6.3 shows the software used to implement the Cloud node. An Apache (`http://httpd.apache.org/`) server runs on a virtual machine, instantiated on an IaaS cloud. The Apache server executes a Django (`https://www.djangoproject.com/`) application, that relies on MySQL (`https://www.mysql.com/`) to store data, and on Django REST Framework (`http://www.django-rest-framework.org/`) to create API endpoints. The Cloud node exposes an URL for data insertion. This URL is protected by Apache, which only authorizes clients if they present valid certificates, generated by a trusted Certificate Authority. Therefore, on SensingBus as a whole, data integrity and authorization between the Sensing level and the Fog level rely upon WPA2, whereas between Fog and Cloud levels, data integrity and authorization are assured by HTTPS. The Cloud node prototype also provides API endpoints for querying data. Data can be queried by date, time, location, and sensor type. Thus, users can use SensingBus to develop their own applications. Table 6.4 lists the endpoints of the API. The complete API documentation can be found together with the available code.

Table 6.3: Software used in the Cloud node prototype.

Module	**Software**	**Publisher**
IaaS Cloud	OpenStack	OpenStack Foundation
Server	Apache 2.4.18	Apache Software Foundation
Database	MySQL 5.7	Oracle
Web Interface	Django	Django Software Foundation
Web API	Django REST Framework	Tom Christie

The following functions are currently implemented in Cloud nodes:

- **Data insertion endpoints:** API endpoints are exposed for data insertion;
- **Data query endpoints:** API endpoints are exposed for data query. Data can be queried by date, time, location and sensor type.
- **Data visualization:** Data can be visualized on a map.

The communication between Sensing nodes and Fog nodes is performed using HTTP, the communication between Fog nodes and Cloud is performed using HTTPS, and users fetch data using a RESTful API. This ensures uniform interfaces, to deal with the heterogeneity of devices and applications.

Figure 6.2 illustrate the data visualization offered by the Cloud node.

Table 6.4: API endpoints offered by the prototype.

Name	Method	Description
/visualize	GET	Renders the visualization webpage
	POST	Returns map data, filtered by the parameters
measurements	GET	Lists all code measurements, filtered by the parameters
measurements/k	GET	Retrieve the measurement represented by k
	PUT	Update the measurement represented by k
	DELETE	Delete the measurement represented by k
/batch_sec	POST	Receives a batch of measurements using HTTPS
/zip_batch_sec	POST	Receives a batch of zipped measurements using HTTPS

6.4 Prototype analysis

In this section, we perform experiments with the prototype nodes. First, we evaluate the accuracy of data sensing in the presence of mobility. Then, we perform experiments stressing the Fog node prototype.

6.4.1 Sensing accuracy in the presence of mobility

We evaluate sensor measurements reliability under a real scenario. We build two Sensing node prototypes to perform this task: one plays the role of a mobile sensor and the other plays the role of a static sensor.

Two sessions of experiments are carried inside the campus of *Universidade Federal do Rio de Janeiro*, in Brazil. In the first one, the *Mobility Effect Experiment*, we compare the measurements obtained by a static and a moving Sensing node. The second session of experiments, named *Casing Effect Experiment*, takes into account measurement behavior regarding the case of the sensor bank.

Mobility Effect Experiment

We position the static node aside an internal street of the campus, as shown in Figure 6.3a. We attach the mobile node to the car window, as shown in Figure 6.3b. The car performs a circular trajectory close to the static node. The nodes are synchronized to acquire data simultaneously. This procedure allows the comparison between the sensed data collected under distinct mobility conditions. As a consequence, we can evaluate possible factors that may affect the accuracy of measurements during mobile platform circulation in metropolitan areas. This experiment is carried out on a sunny day between 10:00 AM and 12:00 AM.

The Mobility Effect Experiment reveals a relevant difference between static and mobile sensors measurements. Figure 6.4a shows the speed of the mobile node throughout the experiment. The plots in Figure 6.4 are aligned in the horizontal axis, meaning that the times are aligned between plots. Figure 6.4b shows the

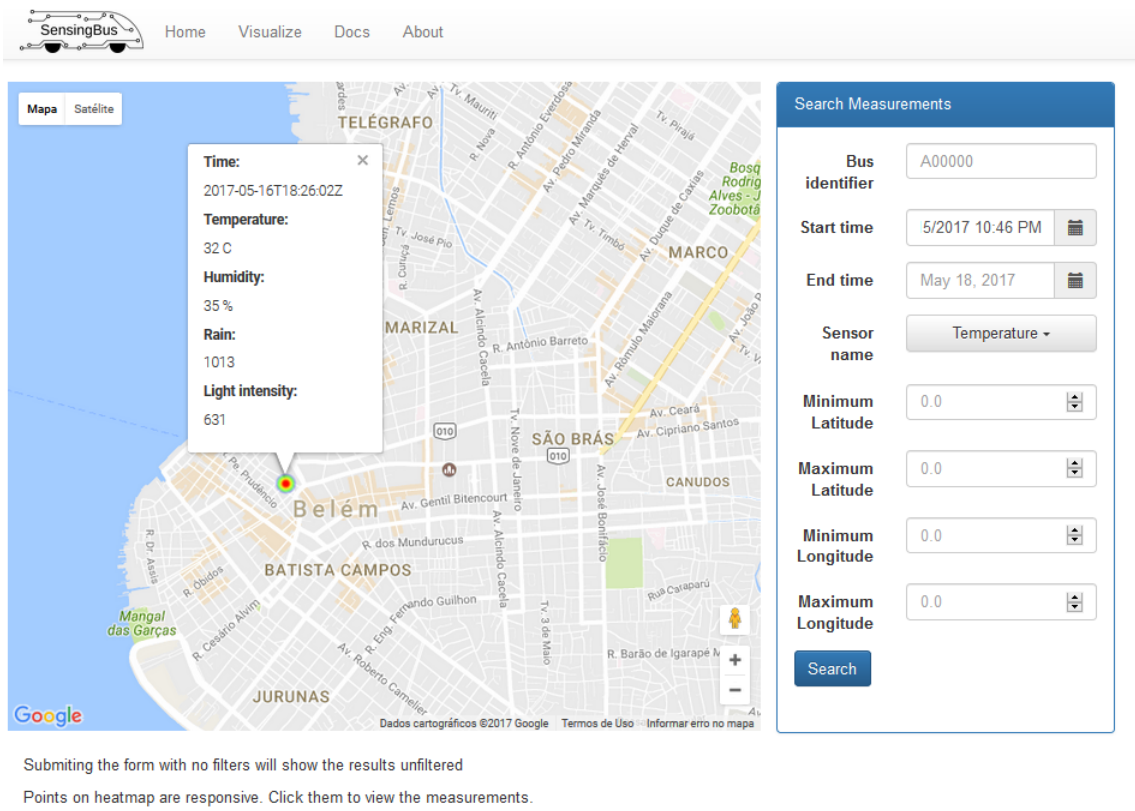
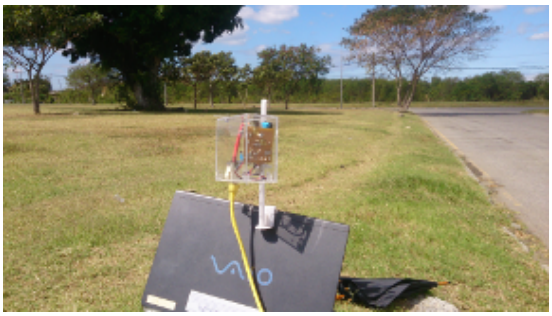


Figure 6.2: Data visualization offered by the Cloud node.



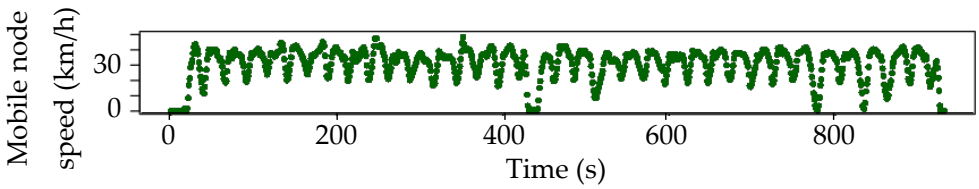
(a) Static Node.

(b) Mobile Node.

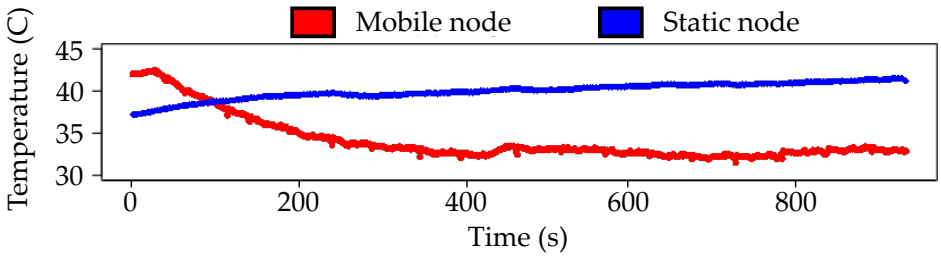
Figure 6.3: Positioning of the Sensing nodes for the mobility experiment.

temperature variation of both static and mobile nodes along the time at one sample per second rate. The blue points plot the air temperature in Celsius measured by the static node sensor, whereas the red points plot the mobile node sensor measurements. As one can observe, while the static node registers a temperature increase around 4 C, the mobile node registers a decrease around 10 C. A reverse behavior can be observed for the relative humidity, at smaller proportions. As Figure 6.4c shows, the static node value (blue dots) decreases 4%, whereas the mobile node value (red dots) increases 7%. Light intensity has no speed dependency and a comparison between the mobile and static values reveals high sensor directionality, as shown in Figure 6.4d. Although the barometric pressure shows fluctuations during the

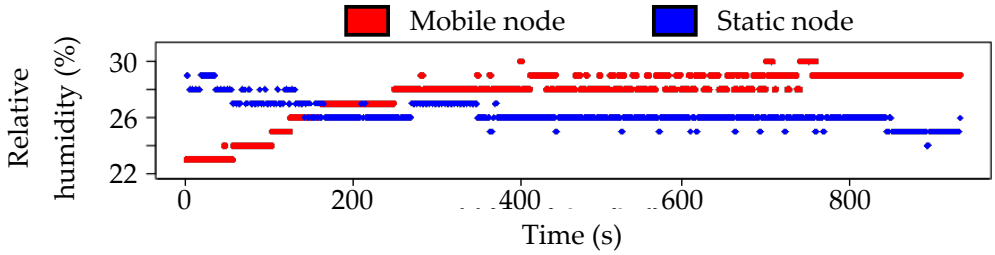
experiment, the speed dependency can be assigned to the Venturi effect caused by the airflow through the box bottom face holes [68], as shown in Figure 6.4e. This effect motivated us to perform our next experiment.



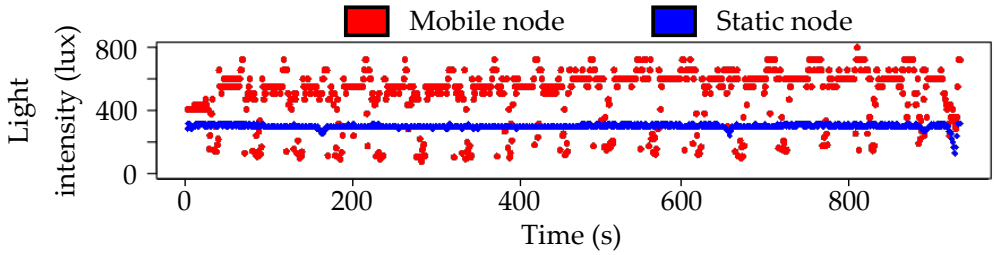
(a) Speed of the mobile node throughout the experiment.



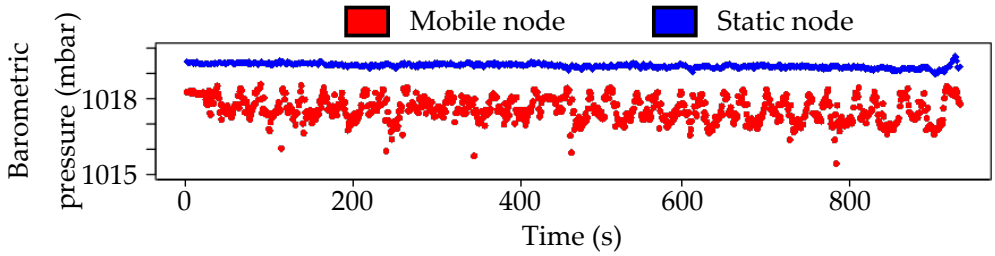
(b) Temperature observed in the experiment.



(c) Relative humidity observed in the experiment.



(d) Light intensity observed in the experiment.



(e) Barometric pressure observed in the experiment.

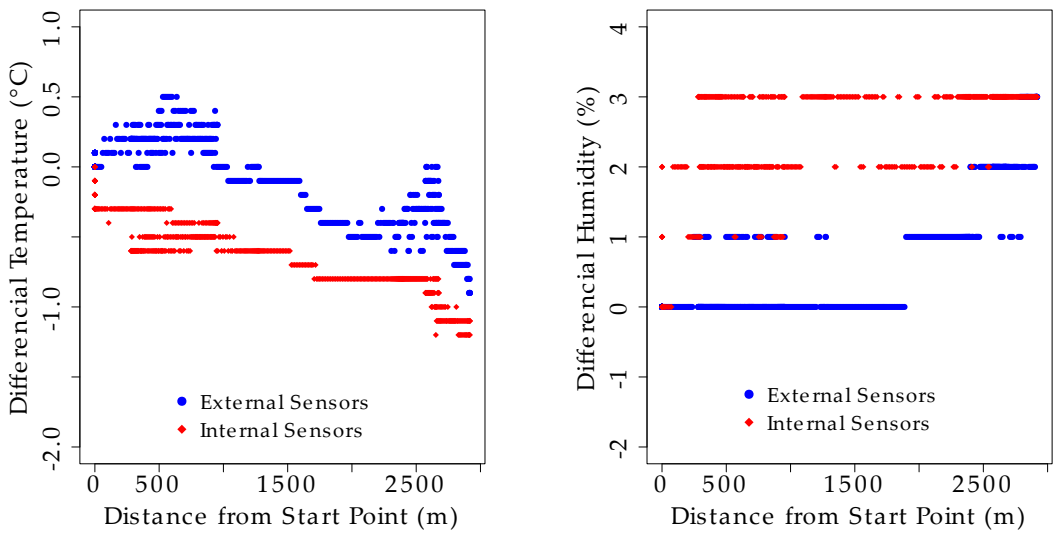
Figure 6.4: Comparison between static and mobile measurements.

Figure 6.5: Trajectory followed in Casing Effect Experiment (Source: Google Earth).

Casing Effect Experiment

The Casing Effect Experiment is performed after the analysis of the mobile effect that showed a temperature variation of up to ten degrees between the mobile and the static node measurements. As one can observe in Figure 6.3, the sensors are protected by an acrylic box, which has a number of holes on the bottom face to perform air inlet and outlet. This assembling is weatherproof, protecting the sensors against bad weather conditions. Nevertheless, we must check whether the data acquired is reliable, regardless of the mobility. To accomplish this task, we attach two Sensing node prototypes side by side on the car window. One of the prototypes with the sensors inside the box and the other with the sensors outside the box. The vehicle moves along the trajectory shown in Figure 6.5 to acquire campus environmental data. This experiment session was performed on a cloudy day between 3:00 PM and 4:00 PM, with no rain.

Figure 6.6 shows the differential temperature and relative humidity between the start point and points along the trajectory. Temperature variation, as shown in Figure 6.6a, during the vehicle trajectory, reveals that the small difference between internal assembled sensors (red points) and external assembled sensors (blue points) suggests a minor casing effect. The chosen trajectory (shown in Figure 6.5) has some local micro-climates not detected by internal sensors but detected by the external ones. These micro-climate areas can be easily identified by analyzing the heat map of Figure 6.7. Regarding relative humidity, the casing effect is more visible due to the



(a) Internal and external temperature.

(b) Internal and external humidity.

Figure 6.6: Comparison Between Internal and External Measurements.

low values of internal sensor measurements compared with the external ones. Similar to the temperature behavior, one can verify in Figure 6.6b that internal sensors show almost no fluctuations, whereas the external sensors show more fluctuations in relative humidity measurements.

6.4.2 Fog node performance

We conduct a number of experiments to test the scalability of our Fog node prototype. Our goal is to know whether the Fog node prototype can stand all the simultaneous contacts that it should in a real deployment. To achieve this goal, we must first evaluate the number of contacts a single Fog node has to serve simultaneously. To estimate this number, we perform an analysis of the datasets we analyzed in sections 5.1.6, 4.7, 5.2.2, and 5.3.2, containing the positions of all bus stops in Rio de Janeiro and the positions of all buses in Rio de Janeiro, refreshed every minute. We collect the positions of all buses throughout 8:00 AM and 10:00 PM of November 30th, 2016. The early and late hours of the day are discarded because a great number of buses is parked, creating distortions in our evaluation.

We assume that every bus carries a Sensing node and every bus stop has a Fog node installed. For every minute, the positions of buses and bus stops are compared. If the distance between a bus and a bus stop is shorter than a given threshold that represents the communication range, we consider that the Sensing node in the bus can send data to the Fog node on the bus stop. We define five different communication ranges: 1000, 500, 250, 125, and 72 meters. Using this dataset, it is possible to evaluate the amount of data received by every Fog node.

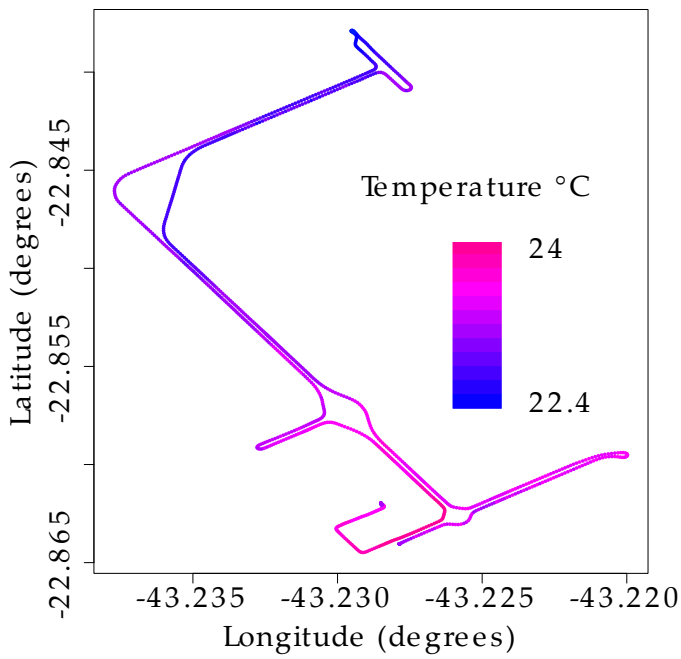
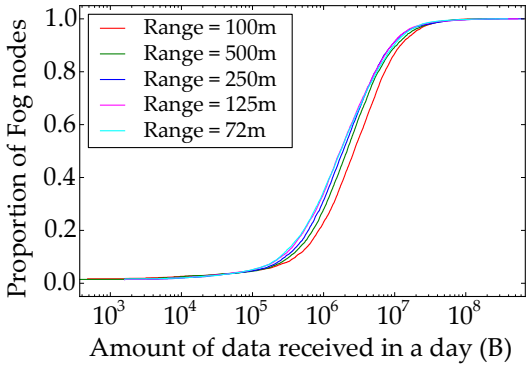


Figure 6.7: Temperature measured by the external sensor.

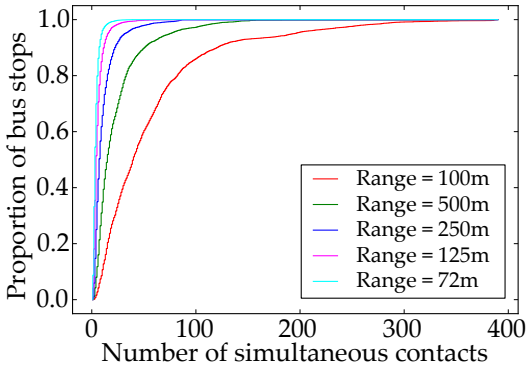
As mentioned in Section 6.1, the Sensing node generates 53 bytes per second, it is possible to estimate the data received by a Fog node on a single contact. We estimate the data received as the Sensing node data generation rate times the elapsed time since the last encounter of this Sensing node with a Fog node. Fig 6.8a illustrates the cumulative distribution of the amount of data received by each Fog node throughout a day. It is possible to note that, for all ranges, less than 40% of Fog nodes receive less than 1 MB (10^6) of data.

To test the scalability of the Fog node prototype, we evaluate the maximum number of Sensing nodes served simultaneously by each Fog node, during the day. Figure 6.8b shows the cumulative distribution of the maximum number of Sensing nodes served simultaneously by each Fog node, for different communication ranges. According to Rubinstein *et al.* [49], the range of IEEE 802.11g in traffic speed is inferior to 250 m. Since our prototype can use IEEE 802.11b/g/n, we adopt 250 m as a reference. In Figure 6.8b, it is possible to note that, for a communication range of 250 m, around 88% of Fog nodes serve less than 20 Sensing nodes simultaneously. Therefore, we adopt 20 as the maximum number of Sensing nodes to use in the stress test.

We use the same datasets to verify the intercontact times of buses with bus stops, considering a communication range of 250 m. We observe that, in more than 98% of the samples, a bus waits less than 10 minutes between contacts. Therefore, we configure every Sensing node in our stress test with data equivalent to 10 minutes of sensing. Given that the sensor bank in our prototype generates 53 bytes every



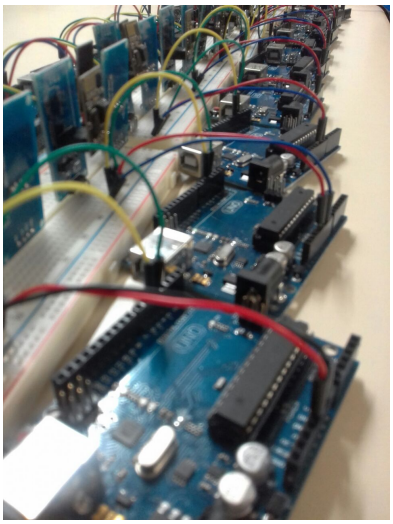
(a) Cumulative distribution of data received by each Fog node in a day, for different communication ranges.



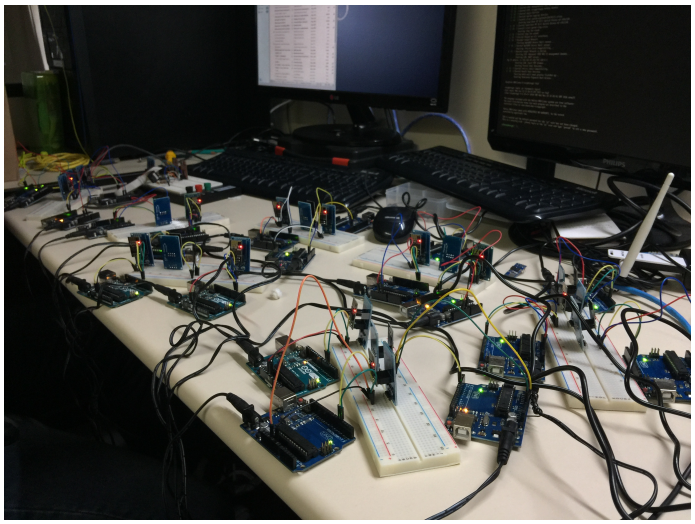
(b) Cumulative distribution of maximum number of Sensing nodes served by each Fog node, for different communication ranges

Figure 6.8: Analysis of the data receive by the Fog nodes of the prototype.

second, 10 minutes of sensing produces 31,800 bytes. This is equivalent to 15 HTTP POSTs from the Sensing node to the Fog node, since each POST holds about 2 kB in data and about 213 B in headers, as mentioned before.



(a) Arduinos and ESP8266's for the Fog node stress test.



(b) Full setup for the Fog node stress test.

Figure 6.9: Testbed to execute the Fog node stress test.

To simulate the simultaneous arrival of Sensing nodes in the communication range of a Fog node, we simultaneously turn on one, five, ten, fifteen, and twenty Sensing nodes within range of the Fog node. After Sensing nodes are on, we wait for all data to be received by the Fog node and transmitted to the Cloud node of the prototype. We call this the *Fog node stress test*. For each number of Sensing nodes, the test is repeated 30 times. Throughout the tests, we do not observe any instability in our nodes. The codes used in this experiment are available in the folder

"fog/stress_tests" of the repository[5]. Figure 6.9 shows the setup of the testbed for the Fog node stress test.

Figure 6.10 shows the results for the average memory, CPU usage, and throughput since the first POST arrives at the Fog node, until the last POST arrives at the Fog node, for all the participating Sensing nodes. Measurements are presented with 95% confidence interval.

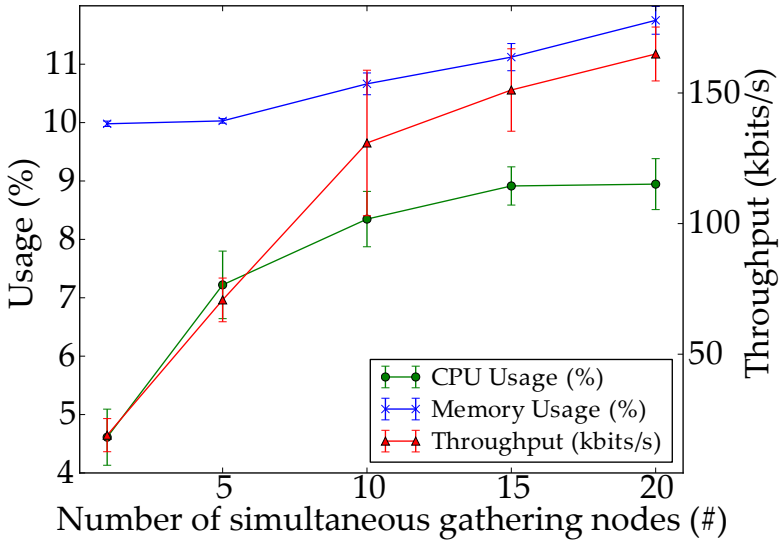


Figure 6.10: Results of the Fog node stress tests.

In Figure 6.10, one can observe that the Fog node can serve up to 20 Sensing nodes without exhausting its memory and CPU resources. It is worth noting that the average throughput growth is not linear. This means that the throughput obtained by each Sensing node drops as the number of Sensing nodes increases, as a consequence of competition for the transmission medium. In Chapter 4, we study the intercontact times between buses and bus stops. Considering the data generation rate of the sensor bank and the reasoning in Section 4.3, we can conclude that the throughput per Sensing node must be at least 3 kbps to deliver all data gathered in a single contact opportunity. Our experiments show that our prototype suits such requirements, even in the worst case, with 20 simultaneous Sensing nodes sharing the same Fog node.

6.5 Remarks

In this chapter, we presented a prototype for the nodes of each SensingBus level. We tested the accuracy of the data gathered by the Sensing node in the presence of mobility. In the experiments, we found that the casing plays a significant role when

[5] `https://github.com/pedrocruz/sensing_bus/tree/master/fog/stress_tests`

it comes to the Venturi effect, and can hinder measurements of barometric pressure. Regarding the Fog node prototype, we successfully tested its capacity to serve up to 20 simultaneous buses.

With the tests performed on the Sensing and the Fog node prototypes, we believe that they are suited for smart city sensing under the aspects of sensing accuracy, persistent memory size, and pre-processing requirements.

Chapter 7

Conclusions and Future Work

Smart cities need data to provide services to their citizens [1]. Mobile Wireless Sensor Networks (MWSNs) are an option to decrease the sensing cost of large areas, such as a city [3]. The mobility increases the region covered by each sensor, lowering the number of sensors and, consequently, the costs. Additionally, mobility enables opportunistic data delivery, creating a trade-off between networking costs and delivery delay [4]. Using the Internet of Things (IoT) paradigm [2, 5], it is possible to leverage the mobility of buses for sensing tasks. Therefore, buses become sensor nodes that travel through the city gathering data.

IoT devices are usually limited in terms of computational power. To compensate for the limited resources of IoT devices, and cope with the need for low latency by some services, a three-level architecture can be used [8]. In the first level, IoT devices gather data; in the second level, Fog nodes receive data and pre-process data; in the third level, a Cloud node processes, stores and serves data. This creates a scenario where buses equipped with Sensing nodes drive through the city, collect data, and deliver data to Fog nodes located in bus stops. The Fog nodes pre-process data and send it to a Cloud node, that serves data to third-party applications.

Sensors are carried around by buses and deliver data when they are in the communication range of a Fog node. This means that regions are not covered the whole time and sensed data is not delivered instantly. Therefore, there are different visiting frequency and delivery delay for different streets. Since smart city applications have different data needs [1, 45], each application can benefit from data gathered from different streets. Consequently, the coverage of the network is not the same for different applications.

To confront the challenges posed by bus-based urban sensing, this thesis presents three main contributions:

- A study of the data delivery delays, with a method to minimize delays for a constrained number of Fog nodes.

- A coverage metric, considering delay and sensing frequency requirements and a metric for bus relevance to their coverage contribution.
- A prototype for each node of SensingBus.

We applied these contributions to real datasets produced from the buses of the city of Rio de Janeiro. This procedure allowed us to have a few insights about a real-world bus-based MWSN.

The next sections examine each contribution in detail. We then conclude this thesis by presenting thoughts on future work.

7.1 Data delivery delays

When investigating the effects of bus-based sensing for delivery delays, we showed that the number and positioning of Fog nodes in the network pose great influence on the network delivery delay. We proposed a Mixed-Integer Linear Programming (MILP) problem based on the p-center problem to place a limited number of Fog nodes into the bus stops while minimizing the maximum delivery delay. We also proposed a heuristic to find suboptimal solutions [17, 18]. This heuristic is able to reduce the cardinality of the problem, making it possible to find optimal solutions for larger instances [19]. Our results showed that, for the city of Rio de Janeiro, it is possible to obtain an MWSN using approximately 16% of the bus stops as Fog nodes and have a maximum delay of 32 minutes in data delivery. It is worth noting that the maximum delay is 30 minutes when every bus stop in this network acts as a Fog node.

7.2 Coverage metric

We also analyzed the spatial coverage of an MWSN based on urban buses. We proposed a coverage model as a function of the street segments to be sensed, the set of buses equipped with Sensing nodes, and their paths in the city. We formulated a MILP problem to maximize the coverage when there is a budget limiting the number of buses that can be equipped with Sensing nodes. The results showed that with only 18% of the fleet of Rio de Janeiro we can cover at least 94% of the total area which the whole fleet could cover. Interestingly enough, we show that only 32 buses can cover at least 40% of the total covered by all buses [19, 21]. This indicates that it is possible to establish an initial service at a very low cost and incrementally deploy a system that serves the whole city. Finally, the results also show the relationship between the number of buses, spatial coverage, and the number of times that a

specific street segment is visited during the day. These last results inspired us to develop a more strict coverage metric.

We elaborated on the coverage metric to take into account the minimum visiting frequency and the maximum delivery delay tolerated by an application. We applied the metric and presented our results in the form of an abacus. Therefore, it is possible to obtain the coverage of the network, given a certain application that requires a minimum visiting frequency and maximum delivery delay. We also obtained the coverage of applications that have known minimum visiting frequencies and maximum delivery delays in the literature. We found that a waste management application can cover 27.9% of the city of Rio de Janeiro using the urban buses of the city. We also showed that, for this application, buses increase the coverage of the network up to 16.4 times, when compared to the scenario where sensors are static [22, 23].

Finally, we investigated the role of individual buses in the coverage of different applications in a city-wide MWSN. We proposed a contribution metric to individual buses that takes into account the coverage lost for a single application when a bus is not performing sensing tasks. Using our delay-aware coverage metric, we applied the contribution metric to the urban buses in the city of Rio de Janeiro, considering the applications of waste management, air quality, and noise monitoring. We also ranked the buses in terms of their city coverage contributions for each application and compared the rankings of each one. We show that the contribution of each bus is dependant on the targeted application. For instance, the Kendall coefficient between the rankings of the applications of air quality and noise monitoring is 0.17 (1.0 means that the rankings are equal). With the results, a developer can build a more efficient bus-based MWSN, deciding which buses are crucial to each application [24].

7.3 SensingBus prototype

As a final contribution, we developed a prototype for each node of the studied bus-based MWSN. We aimed to provide a proof of concept of the proposed SensingBus system. The prototype demonstrated the feasibility of SensingBus through the results obtained using a prototype, built using low-cost hardware. Those results analyze the scalability of the SensingBus prototype. We have shown that the effect of mobility is negligible to the sensors of temperature, humidity, and light intensity, but they might suffer significant effect from protective cases, necessary to implement the systems. Barometric pressure sensors also suffer from the Venturi effect, which can affect the accuracy of data [69]. We have also shown that the Fog node prototype can receive data from at least 20 buses, simultaneously. We note that 20 simultaneous buses is a reasonable number for a city like Rio de Janeiro [25, 26]. The

software developed for the prototype is registered and available for other developers.

7.4 Future work

As future work, we plan to use the Fog nodes to coordinate cooperation between buses, to eliminate, or at least reduce, the amount of duplicate data. This notion of duplicate data must take into account the spatiotemporal data requirements of targeted applications. Since a bus-based MWSN can serve different applications, it is likely that data gathered for different target applications has to be gathered with different granularity by the same bus. For instance, a Fog node can request a bus to lower the sampling rate of its temperature sensor, but increase the sampling rate of its rain sensor.

We believe it is relevant to consider existing sensing infrastructures. The coordination between existing static or mobile can improve the efficiency of services. For instance, some static network already collects a certain type of data from a certain region, buses going through this region can shut down the sensors in their sensor bank that gather the same type of data.

An important metric to consider in the future is the sensing delay, to discover how long it takes since an event happens until the moment it can be sensed. Some streets are visited with high frequency, but with non-uniform intervals. This can create problems for applications that need quick event detection. A study on sensing delay can identify the possibilities of bus-based MWSN for these applications.

Another important improvement is to consider the different business models of public transportation, to minimize the deployment costs of such a system. For instance, if the transport system is a public concession operated by private companies, it might be of public interest that the contract demands a certain amount of budget for city sensing. Alternatively, it might be more interesting to the city that sensing and transport contracts are completely separated.

A last idea is to take into account the instabilities in the transportation service and their implications for the sensing services. For instance, heavy traffic conditions or regular changes in the bus routes can have an impact on the sensing tasks. The reliability level of transport services is also related to the city where the service operates. This means that different cities might need different levels of countermeasures to undermine the instabilities in the sensing provided by bus-based MWSNs.

References

[1] ZANELLA, A., BUI, N., CASTELLANI, A., *et al.*, "Internet of things for smart cities," *IEEE Internet of Things Journal*, v. 1, n. 1, pp. 22–32, 2014.

[2] GUBBI, J., BUYYA, R., MARUSIC, S., *et al.*, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, v. 29, n. 7, pp. 1645–1660, 2013.

[3] LIU, B., BRASS, P., DOUSSE, O., *et al.*, "Mobility improves coverage of sensor networks." In: *6th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2005.

[4] EKICI, E., GU, Y., BOZDAG, D., "Mobility-Based Communication in Wireless Sensor Networks," *IEEE Communications Magazine*, v. 44, n. 7, pp. 56 – 62, 7 2006.

[5] SANCHEZ, L., MUÑOZ, L., GALACHE, J. A., *et al.*, "SmartSantander: IoT experimentation over a smart city testbed," *Computer Networks*, v. 61, pp. 217–238, 2014.

[6] XU, K., QU, Y., YANG, K., "A tutorial on the Internet of Things: From a heterogeneous network integration perspective," *IEEE Network*, v. 30, n. 2, pp. 102–108, 2016.

[7] BONOMI, F., MILITO, R., ZHU, J., *et al.*, "Fog computing and its role in the Internet of Things." In: *ACM Workshop on Mobile Cloud Computing (MCC)*, pp. 13–16, Aug. 2012.

[8] LI, W., SANTOS, I., DELICATO, F. C., *et al.*, "System modelling and performance evaluation of a three-tier Cloud of Things," *Future Generation Computer Systems*, 2016.

[9] WONG, J. L., JAFARI, R., POTKONJAK, M., "Gateway placement for latency and energy efficient data aggregation." In: *29th Annual IEEE International Conference on Local Computer Networks*, pp. 490–497. IEEE, 2004.

[10] UMER, T., AMJAD, M., AFZAL, M. K., *et al.*, "Hybrid rapid response routing approach for delay-sensitive data in hospital body area sensor network." In: *Proceedings of the 7th International Conference on Computing Communication and Networking Technologies*, p. 3. ACM, 2016.

[11] GHAFOOR, S., REHMANI, M. H., CHO, S., *et al.*, "An efficient trajectory design for mobile sink in a wireless sensor network," *Computers & Electrical Engineering*, v. 40, n. 7, pp. 2089–2100, 2014.

[12] ALI, J., DYO, V., "Coverage and mobile sensor placement for vehicles on predetermined routes: a greedy heuristic approach." In: *14th International Joint Conference on e-Business and Telecommunications (ICETE 2017)*. SCITEPRESS, 2017.

[13] GAO, Y., DONG, W., GUO, K., *et al.*, "Mosaic: A low-cost mobile sensing system for urban air quality monitoring." In: *35th Annual IEEE International Conference on Computer Communications (INFOCOM'2016)*, pp. 1–9. IEEE, 2016.

[14] ZHAO, D., MA, H., LIU, L., *et al.*, "On opportunistic coverage for urban sensing." In: *10th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS)*, pp. 231–239. IEEE, 2013.

[15] ZOYSA, K. D., KEPPITIYAGAMA, C., SENEVIRATNE, G. P., *et al.*, "A public transport system based sensor network for road surface condition monitoring." In: *Workshop on Networked Systems for Developing Regions*. NSDR, 2007.

[16] MARJOVI, A., ARFIRE, A., MARTINOLI, A., "High resolution air pollution maps in urban environments using mobile sensor networks." In: *2015 International Conference on Distributed Computing in Sensor Systems*, pp. 11 – 20. IEEE, 2015.

[17] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., "Um Algoritmo de Posicionamento de Pontos de Coleta para uma Rede de Sensores Baseada em ônibus Urbanos." In: *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'2017)*, 2017.

[18] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., "An algorithm for sink positioning in bus-assisted smart city sensing," *Future Generation Computer Systems*, pp. 761–769, Oct. 2017. Impact Factor: 5.768.

[19] CRUZ, P., COUTO, R. S., LUCENA, A., *et al.*, "Estratégias de pré-processamento para posicionamento de pontos de coleta em redes de sensores móveis." In: *50o Simpósio Brasileiro de Pesquisa Operacional (SBPO'2018)*, 2018.

[20] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., "Análise da Cobertura Espacial de uma Rede de Sensores Baseada em Ônibus Urbanos." In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC'2018)*, v. 36, 2018.

[21] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., *et al.*, "On the Coverage of Bus-Based Mobile Sensing," *Sensors (Basel, Switzerland)*, v. 18, n. 6, pp. 1–12, 2018. Impact Factor: 3.031.

[22] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., *et al.*, "Qu'attendre de la collecte de données dans une ville par des bus équipés de capteurs?" In: *Rencontres Francophones sur la Conception de Protocoles, l'Evaluation de Performance et l'Expérimentation des Réseaux de Communication*, v. 4, 2019.

[23] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., *et al.*, "A delay-aware coverage metric for bus-based sensor networks," *Computer Communications*, v. 156, pp. 192–200, 2020.

[24] CRUZ, P., COUTO, R. S., COSTA, L. H. M. K., *et al.*, "Per-Vehicle Coverage in a Bus-Based General-Purpose Sensor Network," *IEEE Wireless Communications Letters*, 2020.

[25] CRUZ, P., SILVA, F. F., PACHECO, R. G., *et al.*, "SensingBus: Using Bus Lines and Fog Computing for Smart Sensing the City," *IEEE Cloud Computing*, pp. 58–69, 2018. Impact Factor: 4.393.

[26] CRUZ, P., SILVA, F. F., PACHECO, R. G., *et al.*, "SensingBus: um Sistema de Sensoriamento Baseado em ônibus Urbanos." In: *XXXV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos*, 2017.

[27] AKYILDIZ, I. F., SU, W., SANKARASUBRAMANIAM, Y., *et al.*, "A survey on sensor networks," *IEEE Communications magazine*, v. 40, n. 8, pp. 102–114, 2002.

[28] DONG, W., GUAN, G., CHEN, Y., *et al.*, "Mosaic: Towards City Scale Sensing with Mobile Sensor Networks." In: *IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS'2015)*, pp. 29–36. IEEE, 2015.

[29] GUAN, G., CHEN, Y., GUO, K., *et al.*, "Low-cost urban air quality monitoring with Mosaic." In: *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 642–643. IEEE, 2016.

[30] ABERER, K., SATHE, S., CHAKRABORTY, D., *et al.*, "OpenSense: open community driven sensing of environment." In: *ACM SIGSPATIAL International Workshop on GeoStreaming*, pp. 39–42. ACM, 2010.

[31] APTE, J. S., MESSIER, K. P., GANI, S., *et al.*, "High-Resolution Air Pollution Mapping with Google Street View Cars: Exploiting Big Data," *Environmental Science & Technology*, v. 51, n. 12, pp. 6999–7008, Jun. 2017.

[32] VON FISCHER, J. C., COOLEY, D., CHAMBERLAIN, S., *et al.*, "Rapid, Vehicle-Based Identification of Location and Magnitude of Urban Natural Gas Pipeline Leaks," *Environmental Science & Technology*, v. 51, n. 7, pp. 4091–4099, Mar. 2017.

[33] ALSINA-PAGÈS, R. M., HERNANDEZ-JAYO, U., ALÍAS, F., *et al.*, "Design of a mobile low-cost sensor network using urban buses for real-time ubiquitous noise monitoring," *Sensors*, v. 17, n. 1, pp. 57, 2016.

[34] DIAS, D. S., COSTA, L. H. M. K., DE AMORIM, M. D., "Data offloading capacity in a megalopolis using taxis and buses as data carriers," *Vehicular communications*, v. 14, pp. 80–96, 2018.

[35] HU, Y., XUE, Y., LI, Q., *et al.*, "The sink node placement and performance implication in mobile sensor networks," *Mobile Networks and Applications*, v. 14, n. 2, pp. 230–240, 2009.

[36] LIANG, Q., FAN, Y., "An Optimal Sink Placement for High Coverage and Low Deployment Cost in Mobile Wireless Sensor Networks." In: *International Symposium on Intelligence Computation and Applications*, pp. 543–551. Springer, 2017.

[37] FU, Q., ZHANG, L., FENG, W., *et al.*, "Dawn: A density adaptive routing algorithm for vehicular delay tolerant sensor networks." In: *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1250–1257. IEEE, 2011.

[38] FENG, Y., GONG, H., FAN, M., *et al.*, "A distance-aware replica adaptive data gathering protocol for delay tolerant mobile sensor networks," *Sensors*, v. 11, n. 4, pp. 4104–4117, 2011.

[39] MERINO, J., CABALLERO, I., RIVAS, B., *et al.*, "A data quality in use model for big data," *Future Generation Computer Systems*, v. 63, pp. 123–130, 2016.

[40] ZHAO, D., MA, H., LIU, L., *et al.*, "Opportunistic coverage for urban vehicular sensing," *Computer Communications*, v. 60, pp. 71–85, 2015.

[41] ZHANG, M., YANG, P., TIAN, C., *et al.*, "Quality-aware sensing coverage in budget-constrained mobile crowdsensing networks," *IEEE Transactions on Vehicular Technology*, v. 65, n. 9, pp. 7698–7707, 2015.

[42] CHON, Y., LANE, N. D., KIM, Y., *et al.*, "Understanding the coverage and scalability of place-centric crowdsensing." In: *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pp. 3–12. ACM, 2013.

[43] MASUTANI, O., "A sensing coverage analysis of a route control method for vehicular crowd sensing." In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pp. 396–401. IEEE, 2015.

[44] ALBINO, V., BERARDI, U., DANGELICO, R. M., "Smart cities: Definitions, dimensions, performance, and initiatives," *Journal of Urban Technology*, v. 22, n. 1, pp. 3–21, 2015.

[45] SINAEEPOURFARD, A., GARCIA, J., MASIP-BRUIN, X., *et al.*, "Estimating Smart City sensors data generation." In: *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net'2016)*, pp. 1–8. IEEE, 2016.

[46] ZHOU, H., WANG, H., LI, X., *et al.*, "A survey on mobile data offloading technologies," *IEEE Access*, v. 6, pp. 5101–5111, 2018.

[47] BARON, B., SPATHIS, P., DE AMORIM, M. D., *et al.*, "Mobility as an Alternative Communication Channel: A Survey," *IEEE Communications Surveys & Tutorials*, v. 21, n. 1, pp. 289–314, 2018.

[48] DA SILVA, V. B., DA SILVA, F. O., CAMPISTA, M. E. M., *et al.*, "A trajectory-based approach to improve delivery in drive-thru internet scenarios." In: *2013 IEEE International Conference on Communications Workshops (ICC)*, pp. 489–494. IEEE, 2013.

[49] RUBINSTEIN, M. G., ABDESSLEM, F. B., DE AMORIM, M. D., *et al.*, "Measuring the capacity of in-car to in-car vehicular networks," *IEEE Communications Magazine*, v. 47, n. 11, 2009.

[50] KARIV, O., HAKIMI, S. L., "An algorithmic approach to network location problems. I: The p-centers," *SIAM Journal on Applied Mathematics*, v. 37, n. 3, pp. 513–538, 1979.

[51] IPLANRIO. "Descrição do Dataset Conjunto GPS ônibus." Available at `http://dadosabertos.rio.rj.gov.br/apitransporte/apresentacao/pdf/documentacao_gps.pdf` (Retrieved on June 12, 2018), Dec 2016.

[52] IPLANRIO. "Documentação de paradas das linhas de ônibus." Available at `http://dadosabertos.rio.rj.gov.br/apiTransporte/Apresentacao/csv/gtfs/onibus/paradas/gtfs_todas-linhas-paradas.csv` (Retrieved on November 10, 2016), Dec 2016.

[53] GOZÁLVEZ, J., SEPULCRE, M., BAUZA, R., "IEEE 802.11p vehicle to infrastructure communications in urban environments," *IEEE Communications Magazine*, v. 50, n. 5, 2012.

[54] DIAS, D., COSTA, L. H. M. K., "Análise da Capacidade de Dados de uma Rede de Ônibus Urbanos." In: *XXXIV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*. SBrT, 2016.

[55] PIPINO, L. L., LEE, Y. W., WANG, R. Y., "Data quality assessment," *Communications of the ACM*, v. 45, n. 4, pp. 211–218, 2002.

[56] STA, H. B., "Quality and the efficiency of data in smart-cities," *Future Generation Computer Systems*, 2016.

[57] CHURCH, R., VELLE, C. R., "The maximal covering location problem," *Papers in regional science*, v. 32, n. 1, pp. 101–118, 1974.

[58] KAPLAN, E. D., HEGARTY, C. J., "Understanding GPS - principles and applications second edition. ARTECH HOUSE," *Inc., MA*, pp. 153–173, 2006.

[59] GOOGLE. "Google Maps API Snap to Roads." Available at `https://developers.google.com/maps/documentation/roads/snap` (Retrieved on June 12, 2018), May 2018.

[60] GOOGLE. "Google Place IDs." Available at `https://developers.google.com/maps/documentation/roads/intro` (Retrieved on June 12, 2018), May 2018.

[61] RIOÔNIBUS. "Dados operacionais mensais do MUNICÍPIO DO RIO DE JANEIRO ano de 2016 (2º trimestre)." Available at `http://www.rioonibus.com/2016/12/07/planilhas/` (Retrieved on June 12, 2018), 2017.

[62] OPENSTREETMAP CONTRIBUTORS. "South America dump retrieved from `https://download.geofabrik.de/south-america.html` ." `https://www.openstreetmap.org`, 2017.

[63] ZOGG, J.-M., "GPS - Essentials of Satellite Navigation." Technical report, UBlox, 2009.

[64] LUXEN, D., VETTER, C., "Real-time routing with OpenStreetMap data." In: *19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pp. 513–516, 2011.

[65] MEEGAHAPOLA, L., KANDAPPU, T., JAYARAJAH, K., *et al.*, "BuScope: Fusing Individual & Aggregated Mobility Behavior for "Live" Smart City Services." In: *MobiSys*, pp. 41–53, 2019.

[66] MEEGAHAPOLA, L., ATHAIDE, N., JAYARAJAH, K., *et al.*, "Inferring Accurate Bus Trajectories from Noisy Estimated Arrival Time Records." In: *IEEE ITSC*, pp. 4517–4524, 2019.

[67] LIU, X., ZHOU, Y., RAU, A., "Smart card data-centric replication of the multimodal public transport system in Singapore," *J. Transp. Geo.*, v. 76, pp. 254–264, 2019.

[68] BAYLAR, A., OZKAN, F., UNSAL, M., "Effect of air inlet hole diameter of venturi tube on air injection rate," *KSCE Journal of Civil Engineering*, v. 14, n. 4, pp. 489–492, 2010. ISSN: 1976-3808. doi: 10.1007/s12205-010-0489-6. Available at: <`http://dx.doi.org/10.1007/s12205-010-0489-6`>.

[69] CRUZ, P., PINTO NETO, J. B., CAMPISTA, M. E. M., *et al.*, "On the Accuracy of Data Sensing In the Presence of Mobility." In: *7th International Conference on the Network of the Future (NoF'2016)*, Nov. 2016.