FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

End-to-End Reliability without Unicast Acknowledgements over Vehicular Networks

Lídia Cerqueira

FOR JURY EVALUATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Doutora Ana Aguiar

Second Supervisor: Dr. André Cardote

July 24, 2017

Resumo

O projeto *Future Cities* [1] transformou a cidade do Porto num laboratório à escala urbana. Até ao ano de 2015, três plataformas de teste foram desenvolvidas: *UrbanSense*, *BusNet* e *SenseMyCity*. Este projeto proporcionou as condições necessárias para que muitos investigadores, empresas e *startups* pudessem testar as suas tecnologias, produtos e serviços.

A *UrbanSense* é formada por 19 unidades de sensorização ambiental instaladas por toda a cidade. Por outro lado, a *BusNet* é uma rede ad-hoc veicular (VANET) formada por *On-Board Units* (OBUs), instaladas nos autocarros da STCP, e por 43 *Roadside Units* (RSUs). Esta VANET é atualmente operada pela Veniam, criada no seio do Instituto de Telecomunicações, da Universidade do Porto e da Universidade de Aveiro.

A *BusNet* é, sem dúvida, uma boa solução para transportar os dados obtidos pela *UrbanSense* até uma base de dados. No entanto, a *BusNet* não suporta endereçamento unicast, pelo que não existe, atualmente, uma forma imediata de assegurar fiabilidade *end-to-end*. Consequentemente, resultados de primeiras experiências da *UrbanSense* evidenciaram taxas de entrega baixas.

O principal objetivo desta dissertação é explorar possibilidades que solucionem o problema referido, partindo do desenho de um protocolo ao nível da aplicação. Este deve garantir a fiabilidade na transmissão dos dados, sem que seja necessário um endereçamento *unicast*.

Os resultados obtidos provaram que o sistema proposto não possibilita a perda de mensagens. Se isso ocorrer, o único motivo serão as limitações de memória das DCUs. Para além disso, concluiu-se que a escolha do *retransmission timeout* (RTO) não deve ser baseado no *round trip time* (RTT). Em vez disso, o RTO deve depender do número de contactos estabelecidos desde o momento em que se dá a transmissão das mensagens.

[1] http://futurecities.up.pt/site/

Abstract

The Future Cities Project [2] has turned the city of Porto (Portugal) into an urban-scale living lab. Until 2015, three testbeds were developed: *UrbanSense*, *BusNet* and *SenseMyCity*. It allowed many researchers, companies and startups to test their technologies, products and services.

The *UrbanSense* testbed is formed by 19 environmental sensing units installed around the city. On the other hand, the *BusNet* is a vehicular ad-hoc network formed by On-Board Units (OBUs), installed in the STCP buses, and by 43 Roadside Units (RSUs). This VANET is currently operated by Veniam, an Instituto de Telecomunicações, Universidade do Porto and Universidade de Aveiro spin-off.

The *BusNet* is undeniably a great option to carry the data gathered by *UrbanSense* to a storage facility. However, as *BusNet* does not support unicast addressing, there is currently no strict way to provide end-to-end reliability. Therefore, results of first *UrbanSense* experiments have highlighted low delivery ratios.

The main goal of this thesis is to explore possibilities to address this problem, by designing an application level protocol that provides reliability to the data transfer without requiring unicast addressing.

Results proved the proposed system does not allow the bundles loss. If the bundles are lost, the only reason will be the DCUs storage limitations. Furthermore, it was concluded the design of the retransmission timeout (RTO) should not be based on the round trip time (RTT). Instead, it should depend on the the number of established contacts since the bundle transmission.

[2] http://futurecities.up.pt/site/

Acknowledgements

I would like to thank to my supervisor Prof.ª Dra. Ana Aguiar and to my second supervisor Eng. Dr. André Cardote for their guidance and valuable advice.

Additionally, I am thankful for the friendliness and good mood of all the Veniam family. In particular, I appreciate the help of João Gomes, Daniel Moura, Henrique Cabral, José Leite, João Azevedo, Diogo Lopes, Claudio Tereso and Joana Ribau, who had a more direct contribution to my project.

Similarly, I also had the pleasure of being supported by some of the IT members. I am particularly grateful for Yunior Rojo, Tiago Lourenço and Pedro Santos' help.

Finally, I want to express my gratitude to my family and friends who always have a word of encouragement during difficult times.

Lídia Cerqueira

Contents

List of Figures

List of Tables

Abbreviations

ACK	Acknowledgement
ARQ	Automatic Repeat reQuest
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DCU	Data Collecting Unit
DDS	Data Distribution Service
DNS	Domain Name System
DTLS	Datagram Transport Layer Security
DTN	Delay Tolerant Network
FCC	Federal Communication Commission
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
IPv6	Internet Protocol version 6
IoT	Internet of Things
MANET	Mobile Ad-hoc Network
MQTT	Message Queue Telemetry Transport
OBU	On-Board Unit
OSI	Open Systems Interconnection
PSID	Provide Service Identifier
QoS	Quality of Service
REST	Representational State Transfer
RSU	Road-Side Unit
RTO	Retransmission Timeout
RTT	Round-trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Universal Resource Identifier
VANET	Vehicular Ad-hoc Network
WAVE	Wireless Access in Vehicular Environments
WSMP	WAVE Short Message Protocol
WSN	Wireless Sensor Networks
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

1.1 Context

The Future Cities [1] was an interdisciplinary project developed to turn the city of Porto into a smart city. The project combined different areas of research such as mobility, urban planning and communication technologies.

The main goal was to provide startups and researchers a way to validate their technologies. To achieve this, different testbeds were developed and installed across the Porto region so that city-scale experiments could be performed. The city has now become a living lab, known as *Porto.LivingLAB*. Currently, it is composed by three main urban-scale testbeds: *BusNet*, *SenseMyCity* and *UrbanSense*. For the past years, they have been enabling many experiments in the fields of Intelligent Transportation Systems (ITS) and urban sensor networks.

The *UrbanSense* testbed is composed by 19 *Data Collecting Units* (DCUs) for environmental monitoring. Each unit includes a sensor board, a processing board and a control board. They are spread over the city, strategically placed at relevant locations, thus sampling the overall urban behaviour. Besides DCUs, the *UrbanSense* platform also comprises a backoffice, where the collected data is stored, and a communications backbone, responsible for transporting the data from the DCUs to the central database. This infrastructure consists of three optional communication backhauls: cellular, fixed WiFi or vehicular delay tolerant networking (DTN) [3].

The cellular is a high cost solution so it is only used as a last resource, where there is no other connectivity. The fixed WiFi alternative, provided by the municipality, does not cover all the DCUs' locations. For these reasons, the *UrbanSense* testbed takes advantage of the WiFi service, *BusNet*, available in 400 buses of the STCP fleet, the main bus fleet in the city of Porto.

The platform which provides the WiFi service is licensed by Veniam. The startup has equipped all the STCP buses and municipality vehicles with *OnBoard Units* (OBUs). These units work as WiFi hotspots and support vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication, thus establishing a vehicular ad-hoc network.

[1] http://futurecities.up.pt/site/

Therefore, when a vehicle gets closer, the DCUs can establish a WiFi connection with the OBU. Thus, the collected data can be forwarded to the OBU. From then on, the data is carried by the vehicular network to the backoffice in a delay tolerant way. However, for this mechanism to work, the DCUs have to be capable of keeping the samples locally stored until a communication opportunity arises. For this purpose, the DCUs have a local database [3].

A brief architecture of the system is illustrated in the following image [2]:

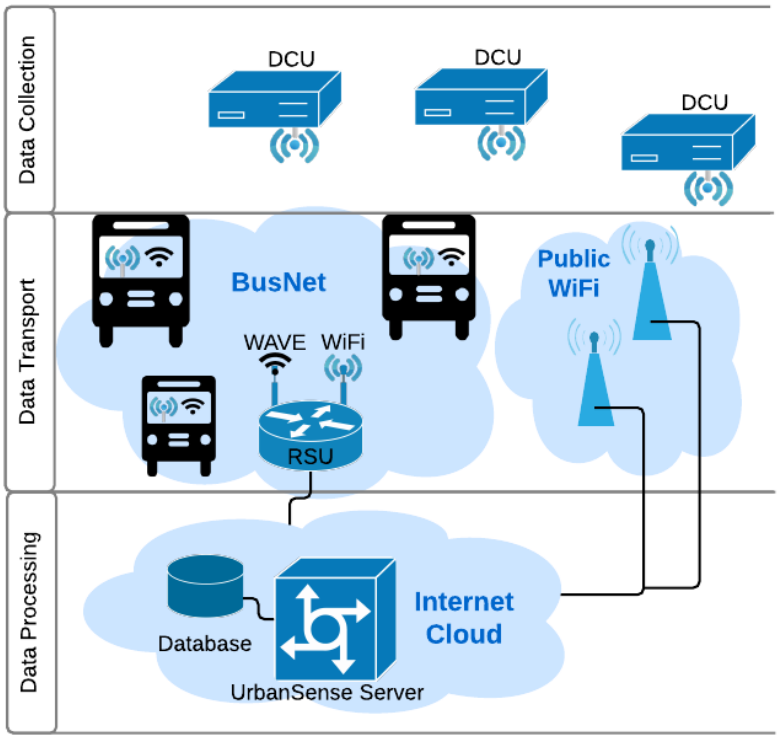


Figure 1.1: *UrbanSense* Architecture

1.2 Problem Definition

1.2.1 Motivation

For both cellular and fixed backhauls, it is possible to establish an end-to-end TCP/UDP socket to transport the data from the DCUs to the central database. Since the communication is synchronous, it is possible for the backoffice to acknowledge the DCUs that it is safe to erase the data from their local storage [3].

On the other hand, the data transport using the vehicular ad-hoc network brings additional challenges when it comes to implement end-to-end reliability.

When *BusNet* option is chosen as the communication backhaul, the DCUs are constantly waiting for an opportunity to connect with an OBU. As soon as a connection is established, the data is sent to the OBU using a Representational State Transfer (REST) API. After that, the transportation

[2] http://futurecities.up.pt/site/

is carried out by a store-and-forward mechanism, until the data is delivered to a *Road Side Unit* (RSU). From the RSU, it is carried via a fixed network to the backoffice.

This solution clearly does not provide end-to-end IP connectivity. Furthermore, within this VANET, the DCUs do not have a fixed IP, they receive a local IP address via DHCP when they connect to an OBU. For these reasons, an application-level unicast acknowledgement cannot be addressed from the backoffice to the DCUs.

1.2.2 Goals

The *UrbanSense* testbed has been using the Constrained Application Protocol (CoAP) REST API for the data transmission from the DCUs to the OBUs. However, low delivery ratios have been recorded in this first link between the DCUs and the OBUs.

Thus, the main goal proposed is to improve the reliability when the vehicular ad-hoc network is used. A solution to be exploited is to close the transportation loop with an application-level acknowledge message from the central database to the DCUs. This possible solution demands other specific challenges to be solved.

Firstly, as already mentioned, the DCUs have no fixed IP. For this reason, it will be necessary to find an alternative way to address the ACKs. Moreover, the Veniam's network has currently no way of carrying information, such as a list of ACKs, from the cloud to an end node, as it is the case of DCUs.

Finally, in every system which aims to provide reliability in data transmission, there is an important system variable that should be designed, the retransmission timeout (RTO). This variable should be carefully set because it has an important effect on the number of retransmissions, thus influencing the system efficiency.

1.2.3 Document Organization

This document is subdivided in different chapters:

- **Chapter 1**: presents an overview of the Future Cities project in order to provide a context for this dissertation. The main challenges and goals are also discussed;
- **Chapter 2**: addresses the most important concepts that will be useful throughout this dissertation. It also presents related work for each concept;
- **Chapter 3**: introduces the design of the end-to-end system. The protocol chosen to ensure end-to-end reliability is presented, as well as the modifications applied when taking the system in study into account;
- **Chapter 4**: exposes the implementation of each system component: DCU, OBU and receiver;
- **Chapter 5**: presents the results obtained in the experiments performed;

- **Chapter 6**: discusses the results and some possible improvements are pointed out as future work.

Chapter 2

Concepts and Literature Review

This chapter presents an overview of the most relevant concepts for the problem in question. In addition, the main differences and similarities with existing solutions for each challenge will also be introduced.

Firstly, the emerging field of the Vehicular Ad Hoc Networks will be addressed, just as their unique characteristics and particular challenges. Thereafter, the main basic concepts of the Delay Tolerant Network will be explored. Additionally, a brief discussion will look into the two main approaches to ensure reliability - *end-to-end* and *hop-by-hop* - over DTN. Similarly, some of the existing proposals for the estimation of the retransmission timeout will be referred, as the RTO is one of the variables to be designed when reliability must be provided. Finally, the most used REST APIs will be compared and evaluated.

2.1 Vehicular Networks

Nowadays, the presence of the internet in our daily lives is undeniable. The perfect scenario would be to provide internet access to the citizens in every single corner of our cities. This would require the installation of a huge amount of wireless routers across the cities in order to guarantee coverage everywhere. The vehicular networks have emerged as one of the solutions for this painstaking task.

The Vehicular ad hoc networks, VANETs, can be seen as a subclass of MANETs, mobile ad hoc networks, where each network node is a vehicle. By equipping the vehicles with wireless routers, these mobile nodes can increase, in fact, the coverage among the cities. Usually, within a VANET, there are vehicle-to-vehicle communications. However, vehicular networks are sometimes used in a hybrid architecture, which involves both vehicle-to-vehicle and vehicle-to-infrastructure communications. The former corresponds to the transmission of information between two OBUs and the later between an OBU and a RSU. The RSUs are the nodes that provide access to a fixed network.

Nevertheless, in more recent years, VANETs have been receiving an increasing attention from both research and industry, given the emergent market research towards autonomous vehicles. The characteristics of these networks make them the perfect ally for the self-driving cars industry. In

the first place, the mobility of the nodes is restricted to the existing roads, which means that if the city maps are known, the position of the vehicles can be predicted. Secondly, information of interest can be delivered depending on the vehicles' geographic position. Additionally, On-Board Units have neither power constraints, nor limited computational power.

However, the VANETs frequently face some challenges. For example, since the nodes of these networks are vehicles, a high mobility can be experienced, which means that the topology of these networks will be constantly changing. In some cases, that can also lead to the establishment of isolated clusters, which are a group of nodes that, due to the topology dynamic, were disconnected from the rest of the network. Moreover, the number of nodes can easily increase. For this reason, vehicular networks should be designed taking into account its scalability potential.

2.1.1 Data dissemination

One of the main decisions that a vehicular network designer has to take is the data dissemination algorithm used for spreading the information among all the participating nodes.

Prior to mention the data dissemination algorithms, it is worth mentioning that, even the most basic data transmission might require a multi-hop scheme. In fact, it demands the establishment of more than one single-hop transmission which means that there will be some vehicles between the source and destination that will perform the role of forwarders.

In the survey [4], the dissemination techniques are classified in:

- Geocast and Broadcast - the data is sent from the source to all of the nodes (or to all nodes located in a specific geographic position)
- Multicast - the data is sent from the source to specific group of nodes
- Disruption Tolerant Dissemination - the data is delivered to the destination with a delay

When it comes to broadcast, it most basic algorithm is flooding, in which every node relays a receiving packet to all of its neighbours. Obviously, this is an inefficient solution, particularly in dense networks, where a large amount of duplicates will be generated. Flooding in wireless ad hoc networks presents two alternatives that optimize the "blind flooding" algorithm by taking into consideration the information about the nodes' neighbourhood. In most of the cases, it is the right selection of a group of nodes responsible for relaying the packets that can accomplish better results and that can avoid the broadcast storm problem [5].

More efficient algorithms can be developed if the geographical position is considered - geocast. Multi-Hop Vehicular Broadcast (MHVB) [6] and Enhanced MHVB [7] propose a position-based which comprises two main algorithms: the congestion detection which first deals with wasteful data generated by congestion and then the Backfire algorithm that, based on the distance between each node and the sender one, defines which network participants will be in charge of relaying packets. Similarly, [8] introduce a prototype implementation of Content-Based AODV which uses, as input data of the algorithm, information collected in the vehicles such as their position,

velocity and trajectory. Another example is [9], it presents the Smart Broadcast (SB), in which the strategy is to minimize the time required to perform a single-hop transmission.

Within a vehicles network, there are a lot of situations, such as car accidents or adverse weather conditions, that can take advantage of a mechanism which enables a certain group of vehicles to be notified with some information of interest - multicast. The TMA technique, Trajectory-based Muti-Anycast [10], explores the possibility of having a central server who tracks each vehicle of the multicast group. As this server knows the position and trajectory of each element, it is this entity who is in charge of determining the path a message needs to follow. Currently, algorithms that were firstly designed for MANETs are applied to VANETs, such as [11], which introduces the Position-based multicast, PBM. This technique demands a location service responsible for translating destinations in positions. The path between the source and the group of recipients is chosen according to their position and their primary neighbours location. This technique has revealed to be inefficient in VANETs, particularly in dense networks with high mobility. Scalable Position-based multicast (SPBM) [12] was later developed and it accomplished a better performance by attributing different hierarchical levels to different geographical regions.

Lastly, the disruption tolerant dissemination, also known as delay tolerant networking, focuses on the challenges caused by the high nodes mobility such as the frequent disconnections and resulting network partitions. As this subject is of particular interest, it will be addressed in more detail later on.

2.1.2 Standards for vehicular communication

As mentioned previously, within a VANET, there are V2V and V2I communications - frequently referred to as vehicle-to-any (V2X) communication. In order to overcome the specific challenges of these networks and to guarantee an efficient communication between the participating nodes, a specific spectrum band was reserved by FCC and a set of standards were proposed by the IEEE.

2.1.2.1 DSRC - Dedicated Short-Range Communications

In 1999, the Federal Communication Commission (FCC) decided to allocate a specific band in the frequency spectrum for the vehicular environment. Thus, a band of 75MHz, around the 5.9GHz frequency, was reserved for V2I and V2V communications. This spectrum allocation is known as DSRC [13]. It can be freely used, no payment will be charged to the users.

The DSRC spectrum is divided into seven channels:

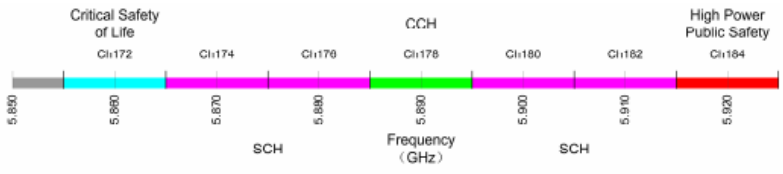


Figure 2.1: DSRC spectrum [1]

The centre channel, 178, is called the Control Channel (CCH) and it is only for safety-critical communications, transmitted in the WAVE short message (WSM) format. There are two channels, 172 and 184, which have a special purpose. The other four are Service Channels (SCH) can be used for common IP traffic. Both CCH and SCH allow the transmission of WSM. In contrast, IP datagrams should be delivered in the SCH band [1].

This allocation was done in the US. However, other continents and countries have also made some efforts in order to reserve spectrum regions for vehicular communications. The following schematic outlines the decisions made in Europe, North America and Japan:

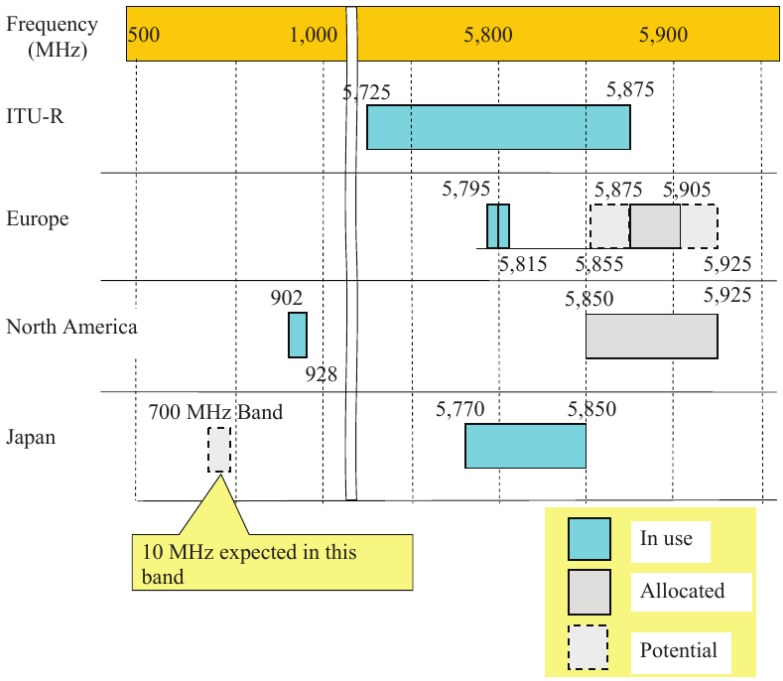


Figure 2.2: DSRC spectrum allocation in different countries [2]

2.1.2.2 WAVE standards

Wireless Access in Vehicular Networks (WAVE) standards include both IEEE 802.11p and 1609.x protocols. The target of the IEEE 802.11p are the lower levels: the PHY and MAC layers. On the other hand, the higher layers are defined by the IEEE 1609.x.

The following schematic presents the overall stack of these protocol families.

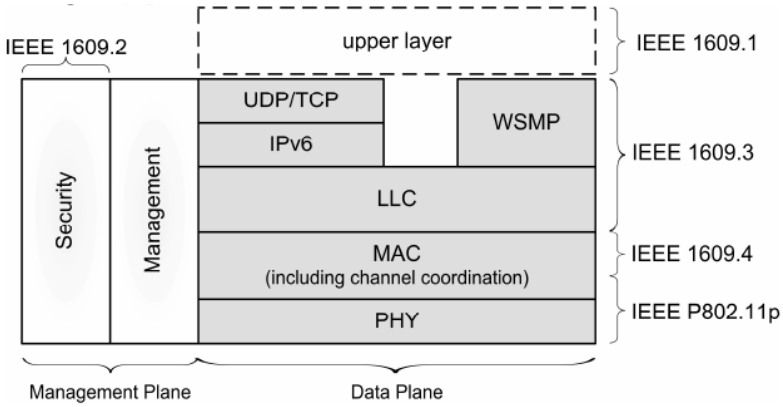


Figure 2.3: IEEE 802.11p/1609.x (WAVE) protocol suite [1]

As shown in the previous image, both IPv6 and WSMP are supported by the WAVE standards. The advantage is that it allows the coexistence of common messages and safety messages with higher priorities.

Relying on the spectrum allocation for DSRC, the IEEE has been defining the IEEE 1609.x standard, which are composed by a group of sub-standards. In January 20016, the IEEE Standards Association Standards Board approved [1]:

- IEEE 1609.2 - Security Services;
- IEEE 1609.3 - Networking Services;
- IEEE 1609.4 - Multi-channel Operations;
- IEEE 1609.12 - Identifier Allocation.

The IEEE 1609.2 defines secure message formats and processing. The IEEE 1609.3 specifies services placed at network and transport layers such as routing and addressing services. The IEEE 1609.4 presents some improvements to the IEEE 802.11 in order to support WAVE operations. The IEEE 1609.12 was later defined, it specifies procedures for the management of a Provide Service Identifier (PSID) [2].

The IEEE 802.11p is a result of some improvements applied to the 802.11a [13] in order to better respond to the challenges faced by the vehicular networks. In particular, it should be capable of dealing with high mobility patterns and the consequent frequent disconnections [14]. Similarly, the standard should also address relevant issues such as interference and multipath [15].

Therefore, there are some differences between IEEE 802.11a and IEEE 802.11p [15]. The main change applied was related to the channels bandwidth, the IEEE 802.11p has 10MHz channels because the the previous ones (20MHz channels) were not able to avoid inter-symbol interference. The 10MHz channels were proved to be a better option in [16]. Additionally, the receivers improved their performance regarding the adjacent channels rejection.

[1] https://www.standards.its.dot.gov/Factsheets/Factsheet/80
[2] https://standards.ieee.org/findstds/standard/1609.12-2016.html

In future versions of the IEEE 802.11p, the PHY and the MAC layer are not expected to suffer large modifications. Nevertheless, safety and network performance can still be improved by the application of more robust information dissemination algorithms [17].

2.2 Delay Tolerant Network

The Delay Tolerant Network concept was firstly applied as an approach to interplanetary internet. For interplanetary communications, there are some characteristics that must be taken into consideration. Namely, there are long paths with low data rates and high latencies. Additionally, an end-to-end path might not be available so these communications have round trip delays considerably high (they can take hours or even some days) [18].

Similarly, there are terrestrial networks, such as terrestrial mobile networks, which experience the same difficulties - due to nodes mobility, they usually have to deal with intermittent connectivity between the source and the destination entities. They are also frequently composed by end nodes that are resource limited, both in power and memory. Moreover, there might exist some disparities between the protocols used by the different networks. A mechanism to manage the interoperability between regions with different architectures is also required.

Given these similarities between the interplanetary and terrestrial networks, the DTN concept was extended to terrestrial networks. The broadly used TCP/IP protocol is not able to handle such challenges because it was designed under some assumptions that cannot be applied to the mentioned *challenged* networks. For example, it is expected a permanent connection between the end nodes, a short round trip delay, an existing bidirectional path for each communication and a low probability of packets loss [19].

In order to overcome the challenges related to the intermittent connections between the nodes, the DTN proposes a *store-and-forward* mechanism which requires storage capability in each node. As a result, if a connection is down, the source node can hold the data until the link comes back up. This method is often compared to the postal service - messages are retained in a storage node before being forwarded to the next one. This process is repeated until the message reaches the destination entity.

The DTN acts as an overlay network, thus enabling it to operate over the existing protocol stacks. Thanks to that, the data units, called *bundles*, can be forwarded over the network, even if it implies the bundle to cross different subnets (for example, crossing IP-based and non IP-based regions). The protocol which determines how the bundles are delivered in DTN is named *Bundle Protocol*.

2.2.1 Regions and DTN Gateways

Two nodes are said to be within the same region if they can exchange messages with no other intervening entity. For end nodes placed in dissimilar regions, their messages are forwarded through DTN gateways. They are placed at region boundaries and its role is to ensure the interoperability

between two neighbour regions. For instance, DTN gateways must know both protocol networks so they can translate a message from the source to the destination region [19].

2.2.2 Naming and Addressing

One requirement of every network is that nodes can be identified, thus enabling them to receive and send data. In DTN, the identifiers of the nodes are called *name tuples*. At first, these identifiers were compounded by three tuples - *region*, host and application - which could identify the service in interest besides the host itself. However, in subsequent improvements, the last two tuples were merged. The general structure of a *name tuple* is: {*Region name, Entity name*} [19] [20].

Consequently, every node and each DTN gateway *half* is assigned with one *name tuple*. These are variable length fields. The first is the region name and it should be a globally-unique identifier. The second one, the entity name, is only resolvable in the destination region so singularity is not required for this portion.

The *Region names* should be hierarchically structured, hence the size of the forwarding tables can be optimized. This portion is the one that is used to find the message path to the destination region. When the *bundle* is finally delivered to that region, the *Entity name* is locally resolvable. The *application* tuple referred in the early version is obtained by demultiplexing this field. If needed, it might also be translated to a specific-region address. This mechanism is usually defined as *late binding*. Since the *name tuple's* second portion is later interpreted, a DNS Internet naming approach would not be so appropriate. It would require an end-to-end name resolution, which is not so efficient given the long delays between each transaction [19].

However, the referred *name tuples* methodology only works efficiently for stationary networks. On the other hand, for mobile ad-hoc networks, for example, a different solution is demanded for naming each node. A more recent article [20] mentions other option for assigning nodes an identifier. The proposed mechanism uses URIs rather than *name tuples*. By doing so, it is possible to have multiple namespaces, which enables the nodes to have multiple identifiers.

2.2.3 Routing

The routing problem consists of determining the best path to forward a message. For conventional networks, this might be solved by selecting the path which requires the less number of hops, for example. However, in DTN, the most valuable criterion when assigning a path to a bundle is not the same. For instance, a shorter path with a previous intermittent connectivity may not be preferable when compared to a longer "always-on" link.

In *challenged* networks, the most likely situation is that an end-to-end path may never exist. For this reason, the current connectivity between the nodes, the available buffer space, the bandwidth capacity and message size are some of the aspects that have to be taken into account when assigning a path to a bundle. The ultimate goal of every routing decision should be to maximize the messages delivery ratio.

The routing may be either defined at the source node, *source routing*, or during its path until the destination endpoint, *per-hop* routing. For the first case, the path selection is done once so it is not modified after the message leaves the source node. This option is not the most efficient for the networks that frequently suffer from topology changes. Conversely, *per-hop* routing does use local information in each hop, which tends to achieve a better performance. One drawback of this approach is that it might lead to loops in the global path chosen.

Routing algorithms are also commonly divided into two methodologies: *proactive* and *reactive* routing. *Proactive routing* determines routes within a connected subgraph. It does not attend to traffic arrivals and it fails when responding to a request in which the destination node is not connected to that subgraph. Nevertheless, they can inform about the current connected nodes which is the key point for routing decisions. In contrast, *reactive routing* issues the routing computation when traffic has to be sent to a destination. Unknown endpoints are allowed but they require a route discovery protocol to be applied. Similarly, a *reactive routing* will fail if the endpoint is not connected to the subgraph [21].

Another useful feature that might facilitate the routing process is the *route stability*. It computes the ratio of the topology change. In other words, it measures the time interval during which the current connections will remain active. For networks characterized by low topology changes, it turns to be really efficient to use route caching thus avoiding same routes to be calculated more than once [21].

Routing has nearly become an independent field of study within the DTN subject. There are a lot of published articles that explore different routing algorithms.

In Zhao et al. approach [22], the control of nodes mobility is used for data transportation. *Message Ferries* are the entities in charge of collecting bundles and delivering them to the destination nodes, which are assumed to be stationary. The main focus of this work was to design different ferry routes and test their performance. In contrast to the conventional systems, the proposed solution requires the network to adapt itself in order to serve the application needs. There are several networks that might benefit from this solution. For example, in sensor networks, the *message ferries*, such as robots or vehicles, can save power resources to the sensors by collecting their data. Although this network seems to be particularly similar to the one proposed for this dissertation, the routes of the *message ferries*, buses in this case, cannot be assumed to be controllable once they have predefined paths to follow.

Daly and Haahr [23] explore a different solution which considers social phenomenons within the network. Each node is locally analysed and its ability to forward data to other network members is estimated. This feature, named as *centrality*, reflects the relative importance of a node and it allows to identify the *bridge nodes* of the network. Additionally, the *similarity* between nodes, which is related to the number of common neighbours between two network entities, is also calculated . The proposed forwarding algorithm, *SimBet Routing*, takes its routing decisions based on the *centrality* and *similarity* estimations. Therefore, when the destination entity is unknown for the source node, this algorithm chooses to forward the bundle to a node where the probability of finding a suitable bearer is higher.

2.2.4 Fragmentation

As previously mentioned, *challenged* networks are characterised by intermittent connectivity. For this reason, the bundles transmission should take advantage of every established connections, even if it does not allow the message to be fully transmitted.

Bundles fragmentation may occur when the connection between the nodes was lost during the transmission, *reactive fragmentation*, or when the receiver has no sufficient memory to store the entire message, *proactive fragmentation*. Obviously, this brings advantages when transmitting large messages because they may not fit in a single destination node. On the other hand, it brings additional complexity for the bundle's reception.

The reassembly of the fragments might be performed during their path over the network or at the destination. For handling their reconstruction, the fragments use a special header which indicates their offset relative to the original bundle. A common identifier is also used for enabling fragments originated by the same bundle to be grouped [20] [24].

2.2.5 Custody Transfer

The DTN also provides an optional mechanism which allows the improvement of the end-to-end reliability. By taking the custody of a message, a node promises to keep the bundle in its persistent memory until another node accepts the custody of that bundle.

To provide reliability hop-by-hop, these nodes, called *custodians*, wait for a *custody acknowledgment* message from the next node. At that moment, they can safely delete the bundle of their local memory. It is worth mentioning that the *custody acknowledgment* differs from an end-to-end acknowledgment message. It delegates to a *custodian* the responsibility of reliable delivery on behalf of the source endpoint [19] [20].

Depending on their current state, nodes might refuse the *custody transfer* of a message. This case may occur if the node has not enough available storage, for example. Additionally, some of the bundle's characteristics can influence that decision: its size, priority and lifetime.

Thus, a buffer space analysis has to be performed in each node. A poor local memory management may cause a *congestion* of the DTN routers. This might happen when a node becomes unexpectedly isolated from the others that are closer to the destination endpoint. If its memory is fulfilled with messages requiring a *custody transfer*, the node has no way to get free space unless a connection with another entity (which is accepting *custody transfers*) is reestablished. On the other hand, if there is a connected node which is waiting for messages without custody, its reception will not occur because the congested node cannot forward them. This phenomenon, called *head-of-line blocking*, is undesirable as it wastes opportunities to transmit bundles [24].

2.3 End-to-End vs Hop-by-Hop Reliability

The end-to-end argument is one of the most discussed fields in computer networking. It was firstly introduced in the 1980s and it has been subject to many reviews.

The argument helps system designers to decide where to implement functions in a communication network, either in the endpoints or in the network itself. It states that some functions can only be effectively implemented if they have information and resources of the application (placed at endpoint nodes). For this reason, those functions should be placed in the application on behalf of the low levels [25].

One of the study cases used to exemplify the end-to-end argument is the reliable transfer of a file between two hosts. A reasonable question that can be raised is where to check if the file is being correctly transferred. It can be done at lower levels, in the communication network, which ensures reliability in each hop along the file path. Alternatively, those tests can be performed only at the endpoints, by comparing the original checksum with the one calculated at the destination host. However, the first solution was proven to be insufficient once some of the possible failures can occur outside the communication network. In the article, a situation where the gateways introduced errors while copying the file's content between two buffers is described. Therefore, the end-to-end argument should be applied for this study case. The authors also defend that if functions should be placed at the endpoints, it might be a waste of resources to place them in the lower levels as well. Nevertheless, if reliability is also placed in the communication system, that would reduce the number of the end-to-end retransmissions, which are more time consuming [25].

Later on, Tim Moors published a critical review [26] of the first article . It starts to notice that the ends should be precisely identified. In the study case of a "careful file transfer", it is common to assume that the endpoint is the transport layer, such as TCP. This assumption is against the first article which argues that it should be the application to be responsible for checking the integrity of the operation, thus avoiding undetected errors while writing on the disk, for example. However, the author raises another issue related to trust. Since the integrity checks are costly at the application side and once the transport layers are robust nowadays, it is reasonable to trust the transport layer to do so.

The review also points out new arguments for and against the end-to-end argument. Namely, it refers that placing some functions at lower levels represents an extra cost for clients, even for those who do not benefit from the services. Furthermore, an end-to-end approach turns to be a more flexible implementation because no extra functionality implemented at lower levels will interfere with new services that might be provided at the endpoints. On the other hand, there are other functions, such as routing and congestion control, that are not suitable for an end-to-end method.

Obviously, this discussion is also relevant for the DTN subject. Both options, either to ensure reliability in each hop or at the end nodes, have their advantages and drawbacks.

The custody transfer ensures hop-by-hop reliability. By doing so, the source nodes, frequently resource-limited devices, are able to free their storage resources quicker, since an end-to-end acknowledge has to follow a longer path when compared to the path of a custody acknowledgement. With an end-to-end approach, the retransmission buffer space must be increased in order to avoid memory overflows. Furthermore, when data loss occurs, the instant to retransmit is delayed.

Moreover, that might not even be a reasonable solution if the time it takes to receive an end-to-end acknowledge exceeds the lifetime of the source nodes [24] [18] [19].

On the other hand, the end-to-end approach is usually addressed as a more robust mechanism to assure reliability than hop-by-hop. For example, the *Data MULEs* article [27] points out the possibility of the MULEs' failure (entities responsible for carrying the bundles). In these circumstances, only the end-to-end approach would ensure reliability. Anyhow, if hop-by-hop reliability is also implemented, the overall performance is increased.

2.4 Retransmission Timeout

One of the key points for a reliable end-to-end approach is to estimate the best time to retransmit a message, *retransmission timeout, RTO*. In reliable systems, the source entity sends a message and keeps it on its persistent storage. As soon as an ACK is received, the message can be removed from the local memory. On the other hand, if no ACK is received during RTO time units, the sender concludes that the message was lost and resends it. This challenge has been hardly studied for the TCP transport layer, which also requires an receipt confirmation. Dolev et al. [28] propose an online algorithm for selecting the RTO which maximizes the performance when compared to an offline algorithm, (*comparative analysis*). Ekstrom and Ludwig [29] introduce the *Peak-Hopper Algorithm* which decreases the time needed to detect a packet lost and the number of spurious timeouts. Kesselman and Mansour [30] suggest a new estimate of an optimal RTO which depends on the TCP window size.

In fact, the RTO timer demands for a cautious assessment. If it is set too short, the source might decide beforehand that the message was lost. This leads to worthless retransmissions, therefore increasing the channel overload. On the other hand, if the RTO timer is set too long, the loss of a message will be later noticed, so its retransmission will be delayed. For these reasons, a balance must be found when setting the RTO timer so that a maximum efficiency can be achieved.

In most cases, the RTO is set higher or equal to the the round-trip time (RTT), which includes the time for the message to reach the destination and the time for the ACK to be received by the sender. However, in delay tolerant networks, the RTO estimation is certainly more challenging due to the unforeseen links connectivity, unbalanced data rates and high latencies. Furthermore, it is worth mentioning that both message and ACK can be lost.

Wang et al. [31] suggest a new RTO estimation for deep-space communications, operating over DTN. In contrast to common proposals, the authors proved that an RTO lower than the RTT maximizes the goodput of the bundle protocol. The proposal was firstly defined by an analytical model and then it was proved experimentally. The model is illustrated for the transmission of a long message, such as a file. For the sake of simplicity, only some of the last steps will be addressed.

Firstly, the time to release a message from the source node can be approximated to:

$$T_{total} = RTT + (K-1) \times RTO \tag{2.1}$$

where RTT is the round-trip time, RTO is the retransmission timeout (to be determined) and K is the total number of retransmissions. The calculation of k is also demonstrated in the article and it depends on the channel bit-error-rate (BER).

Thus, the goodput can be estimated as follows:

$$\gamma = \frac{L_{message}}{T_{total}} = \frac{L_{message}}{RTT + (k-1) \times RTO} \tag{2.2}$$

As the equation suggests, a shorter RTO leads to a greater goodput. On the other hand, the number of retransmissions is increased, therefore overloading the channel. For deep-space communications this is not the main concern because the channels are usually devoted for specific data transmissions and their bandwidth is larger enough to handle those retransmissions.

For these reasons, the authors decided to explore the option of setting the RTO lower than the RTT. However, the range of the RTO values has a lower limit, which is equal to the time required to transmit the message. This ensures the RTO timer does not expire before the message actually reaches the destination.

The total number of transmitted bundles required to the complete reception of the original message is given by:

$$D_{total} = \left\lceil \frac{RTT}{RTO} \right\rceil \times N_{bundle} + (k-1) \times N_{bundle} - \sum_{i=1}^{k-1} i \times NB_{(k-1)th} \tag{2.3}$$

where NB is the number of bundles a file is divided for transmission and NB_{xth} is the total number of bundles in the xth transmission effort.

Finally, the goodput, normalized with regard to the amount of data transmitted, can be determined as:

$$\gamma = \frac{L_{file} \times N_{bundle}}{[RTT + (k-1) \times RTO] \times [(\left\lceil \frac{RTT}{RTO} \right\rceil + k - 1) \times N_{bundle} - \sum_{i=1}^{k-1} i \times NB_{(k-1)th}]} \tag{2.4}$$

Once it is difficult to obtain the RTO which maximizes the goodput performance from this equation, some tests were performed and the results were compared with the analytical model 2.4. The experiment results showed that the best performance is achieved when the RTO is approximately one-half of the RTT.

Lastly, it is worth to think about the application of the presented model in this dissertation. The research performed by Wang et al. was developed regarding deep-space communications, where the channels have an higher bandwidth available. Thus, an increased amount of data transmissions can be handled by them. However, in the VANET in question, that assumption cannot be strictly applied since the channels are not dedicated to a specific data transmission.

2.5 REST APIs

According to Cisco IBSG prediction in 2011, by the end of 2020, there will be 50 billion of devices connected to the Internet [3]. IoT devices will mainly transmit real-time data, collected by their sensors. The data exchanged are not expected to be heavy but the amount of messages will be large. Moreover, these devices are power-limited devices. As a result, networks formed by IoT devices require the development of communication protocols which take power-efficiency into consideration.

The Representational State Transfer (REST) approach has came up as a solution to this challenge. REST APIs allow the sensor devices to upload their data on demand. As they do not need to maintain a constantly open connection with the server, they can save power energy in the intervals during which they have nothing to transmit. The client and the server entities are independent and the connection between them are said to be stateless since no session data is shared between two different requests.

Usually, REST APIs provide the same set of methods of the HTTP protocol: POST, GET, PUT and DELETE. In fact, these methods fulfill the needs of any IoT application. The sensors are able to upload their samples and the devices can also receive some instructions to be applied to their actuators. Generally, the data is transmitted in XML or JSON formats and the devices are addressed using Universal Resource Indicators (URI) [32].

Many REST APIs have been developed for the last years. CoAP (Constrained Application Protocol) [4] and MQTT (Message Queue Telemetry Transport) [5] are two of the most known examples.

In short, MQTT defines a multi-client publish/subscribe messaging scheme, where the sensor nodes (clients) are allowed to upload their data and, at the same time, to subscribe a broker (MQTT server) in order to receive updates of interest. It was designed with the goal of saving both power and memory.

CoAP usually runs over UDP and its datagrams are more lightweight than the HTTP packets. As CoAP uses UDP, reliability is not guaranteed. However, there is an option available which allows to label messages as *confirmable*. If this option is set, the client should expect to receive an acknowledge message from the server. Just as MQTT, CoAP was developed in order to answer the needs of nodes with constrained resources [33].

Nevertheless, there are other alternatives to MQTT and CoAP, such as XMPP (Extensible Messaging and Presence Protocol), DDS (Data Distribution Service) and STOMP (Simple Text Oriented Messaging Protocol).

Given the increasing interest in IoT, REST APIs have been largely used by application developers. Many studies have been made either to evaluate or to improve the performance of these APIs.

[3] http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf
[4] http://coap.technology/
[5] http://mqtt.org/

Kovatsch et al. [34] argue that the key point to support power-limited devices is to keep a low duty cycle. This can be achieved by using the radio the minimum as possible. Thus, they propose a low-power CoAP implementation. On the other hand, Raza et al. [35] were more concerned with security and safety issues so they suggested a CoAP integration with a lighter version of the DTLS (Datagram Transport Layer Security). Castellani et al. [32] also took a look at possible improvements in the CoAP, this time by applying a conversion technique to the XML data sent by the sensor nodes. There are also some proposals that explore enhancements in the cloud side, as is the case of [36]. It presents a system architecture that is prepared to handle data from numerous connected devices so that the system efficiency is not affected when the number of participating nodes increases significantly.

As for MQTT, Hunkeler et al. [37] developed the MQTT-S, which is an extension of the original version. Its main target is the wireless sensor networks (WSN). A more recent proposal, MQTT-SN [38], introduces the concepts of *sleeping clients* that allow the messages to be buffered at the broker side until the client wakes up. As it happened with CoAP, the security was also an issue to work on. Singh et al. [39] analysed the integration of the elliptic curve cryptography in MQTT. Lee at al. [40] focused on the three QoS (Quality of Service) levels of MQTT and studied the correlation between the messages loss and delay in each level.

Thangavel et tal. [41] performed a comparison between these two REST APIs. They firstly developed a middleware that both supported CoAP and MQTT. After that, they performed tests in order to evaluate the performance of these protocols. They concluded that the network conditions can influence the results obtained. When the packet loss is lower, MQTT has a better performance regarding the delays. In contrast, for higher packet losses, CoAP has lower delays. Additionally, in order to ensure reliability, the CoAP consumes less bandwidth if the messages are small in their size and if the packet loss is lower.

2.6 Chapter Considerations

This chapter introduced the most relevant concepts and technologies for this dissertation.

Firstly, the concept of VANETs was presented, as well as the data dissemination techniques and the WAVE standards.

Secondly, the delay tolerant networking approach was analysed in more detail as it is used in Veniam's mesh networks.

Since the main goal of this dissertation is to implement a reliable system, two design options were discussed. The first one is related with the methodologies to achieve reliability over DTN: *hop-by-hop* and *end-to-end*. The second one was the RTO setting.

Finally, since the system is indeed formed by sensor units which need to upload their data to the Veniam's network, the main REST APIs were presented.

Chapter 3

System Characterization

3.1 System Architecture

The following schematic illustrates the system architecture when the DTN backhaul was used:

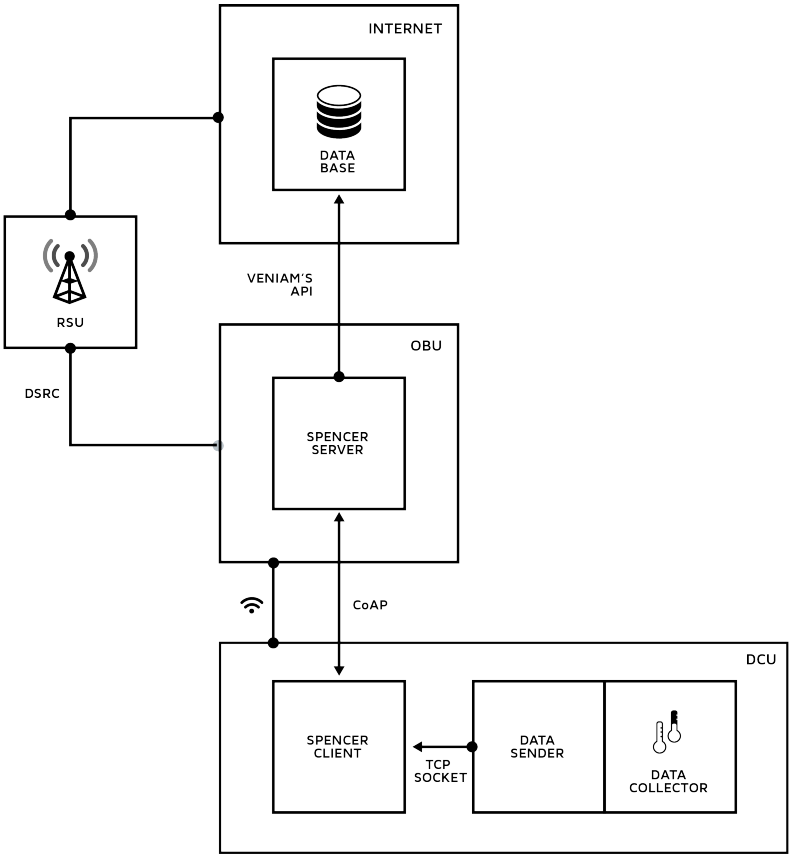


Figure 3.1: System Architecture

In short, the system operated as follows: the DCUs established a WiFi connection with the

OBUs that passed by. The protocol used to transmit the bundles from the DCUs to the OBUs was the CoAP.

It is worth mentioning that the path illustrated between the DCU and the RSU is the most simple version among the possible ones. For example, the first OBU could send the bundles to other OBUs, before they actually be delivered to the RSU.

3.2 System Implementation

As highlighted in the figure 3.1, the two main components of the system were the DCUs and the OBUs. The implementation of each one will now be addressed.

At the OBU side, there was a service, *spencer server*, that was waiting for the arrival of the bundles. As soon as they were received, an ACK was sent back to the DCU. From then on, the bundles were carried by the vehicular network.

At the DCU side, there were three services:

- *datacollector* - responsible for getting the samples from the sensors
- *datasender* - accountable for building and managing the bundles
- *spencer client* - in charge of sending the bundles to the *spencer server* running on the OBUs

The *datasender* and the *spencer client* established a TCP socket between them so that the former could send the bundles to the later. In the *datasender* there was a periodic task which tried to ping an OBU. As soon as a response was received, the bundles were started to be sent to the *spencer client*, who was acting as a bundle forwarder. Similarly, it was also responsible for receiving the ACKs sent from the *spencer server* and forwarding them to the *datasender*.

The entire management of the bundles was done by the *datasender*. The bundles were sent and preserved on local storage. If an ACK was not received from the *spencer server* after a given amount of time, they would be retransmitted. This process was repeated until a maximum number of retransmissions was reached. However, it is important to enhance that this was an “one hop ACK”, which means that, when the *datasender* removed the bundles from the local storage, it did not actually know if they were indeed stored on the database.

The number of bundles being sent simultaneously, the number of seconds to wait until a bundle retransmission and the maximum number of retransmissions were some of the system parameters that could be set by the user.

3.3 First Hop Analysis

From September 2015 to July 2016, while the bundles related to the sensors data were being sent, the DCUs recorded relevant information that allows the first link to be characterized. For each contact (connection establishment between a DCU and an OBU), it was recorded the start time of the contact, its duration, the number of packets sent and the number of ACKs received.

This data was analysed in detail because it can be used to perform an evaluation of the first hop (DCU->OBU). Although there is not enough information to characterize the full path, from the DCU to the database, the characterization of the first link can give useful insights regarding the impact of this link in the overall system performance.

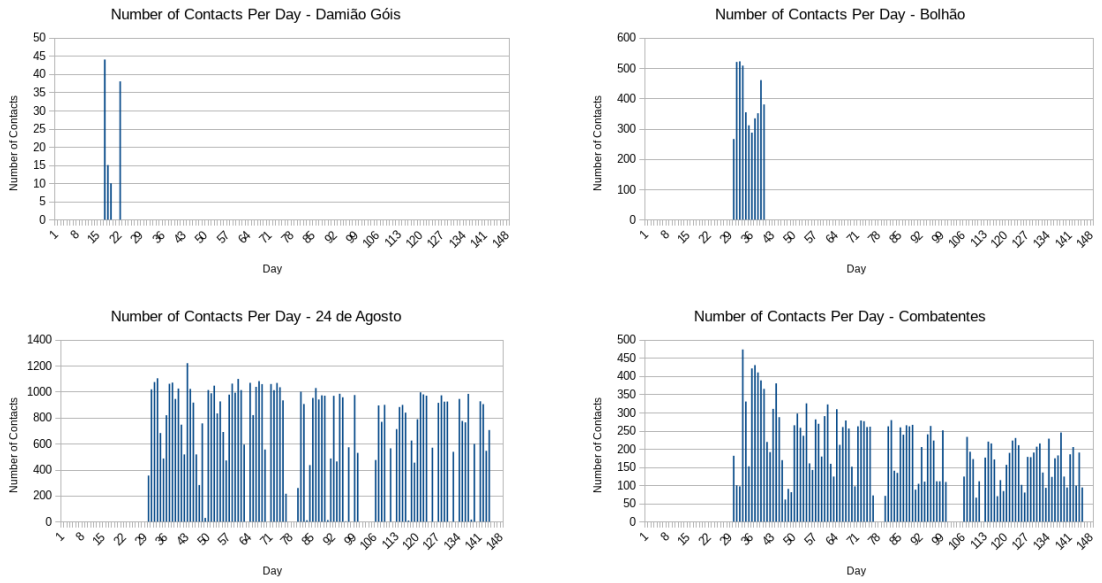
However, it is worth pointing out that this information, just like the sensors data, has arrived to the database via DTN. This means that there might be some contacts that were indeed established but for which there is not information about, due to the loss of the bundles that carry this type of data.

Even so, with the information available about each contact, it was possible to assess some of the first link key performance indicators such as the distribution of contacts, their duration and the delivery ratio along the day.

This analysis was performed for the DCUs which have this type of information available in the *UrbanSense* database: Damião Góis, Bolhão, 24 de Agosto, Combatentes, Campo Alegre, Casa da Música, Praça da Galiza, Praça da Liberdade and FEUP.

3.3.1 Number of Contacts per Day

In order to highlight the relevance of each DCU in future conclusions, the number of contacts per day, during the mentioned period, will be firstly presented. As it can be seen, Damião Góis's DCU is the least relevant since their contacts can be only characterized by the period of four days.



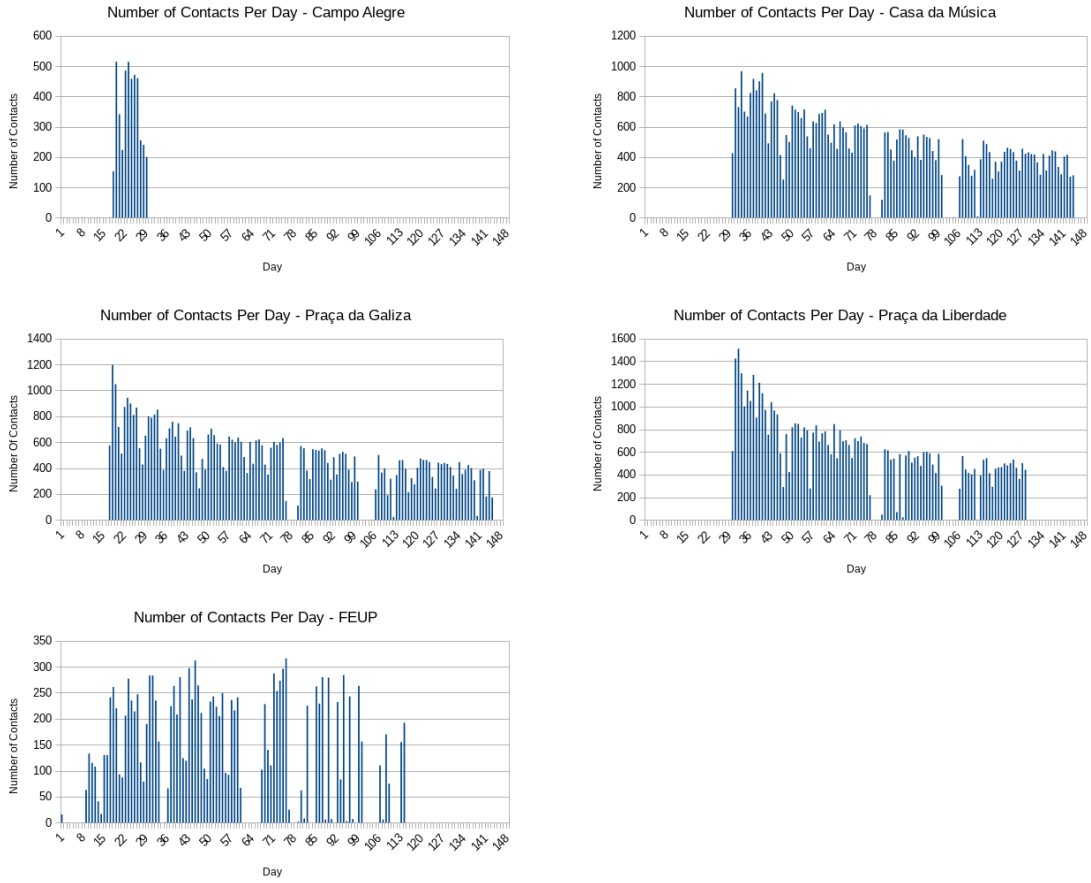


Figure 3.3: Number of Contacts Per Day in Each DCU

3.3.2 Distribution of Contacts per Hour

The following charts summarize the results obtained for the number of contacts established per hour. As expected, the amount of contacts is higher during the daytime, when there are more buses moving around.

The DCU placed at Damião Góis has no contacts established during the night hours because a STCP route does not exist at these hours. This is also the case for the DCUs located at Campo Alegre and Bolhão, where there is not a defined route at night-time. However, as it can be seen, the graphs obtained show a different result, which might be explained by the establishment of contacts with out-of-service buses and garbage collectors, which also carry OBUs.

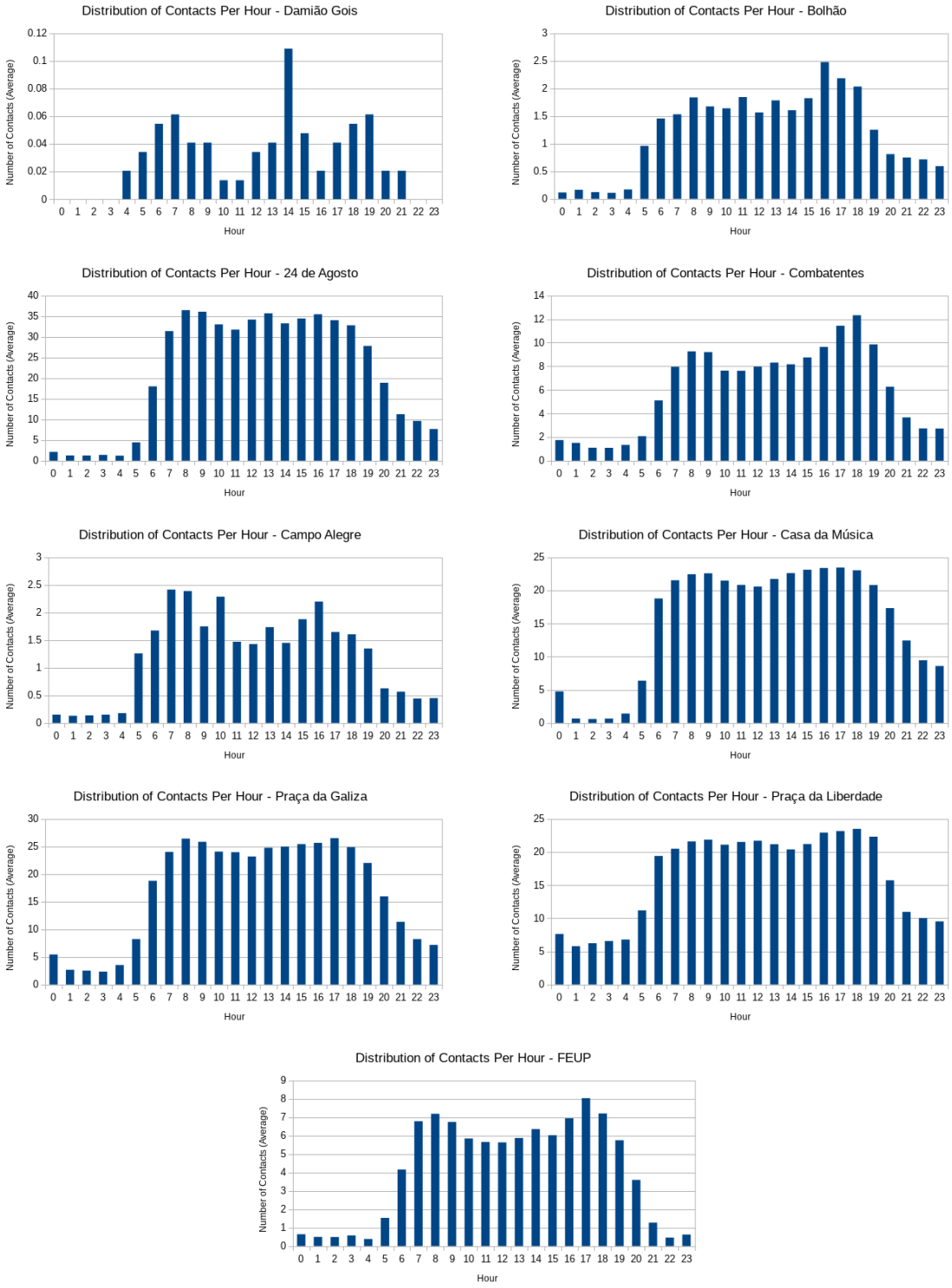
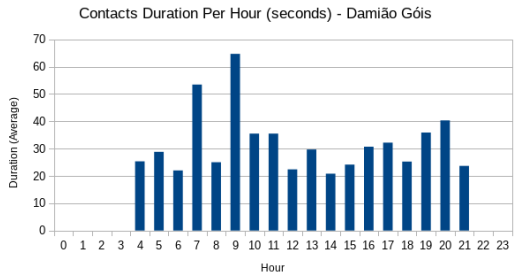


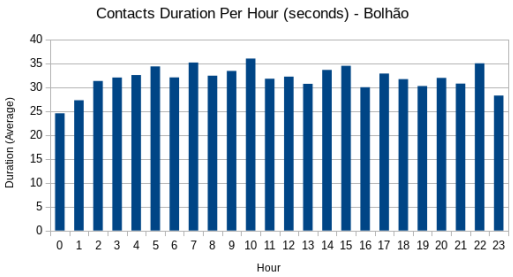
Figure 3.4: Number of Contacts Per Hour in Each DCU

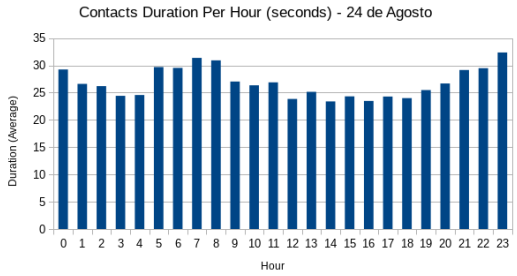
3.3.3 Contacts Duration

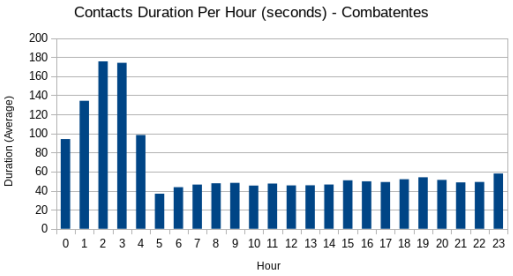
The next graphs are related to the contact duration. The contact's start time is the instant when the DCU connects to an OBU and receives a local IP via DHCP. A contact is considered to be finished when the connection with an OBU is lost (the DCU does not receive an answer when it pings the OBU).

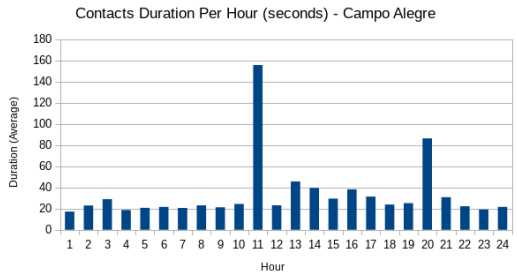
Since almost DCUs are located next to bus lines there are no evident reasons that may justify differences in the duration values at different hours. This behaviour is mainly verified in Bolhão, Combatentes, Campo Alegre and Praça da Liberdade. At the other locations there are some outliers that could be caused by periods during which there is a driver shift. Additionally, when a bus gets delayed (due to accidents, for example), it will find the following stops with a large amount of passengers which will greatly increase the contact duration. If this situation has frequently occurred, it can influence the average duration of the contacts at a given hour, as it is the case in Campo Alegre at 11am.

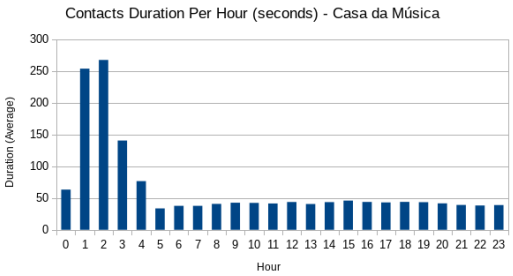












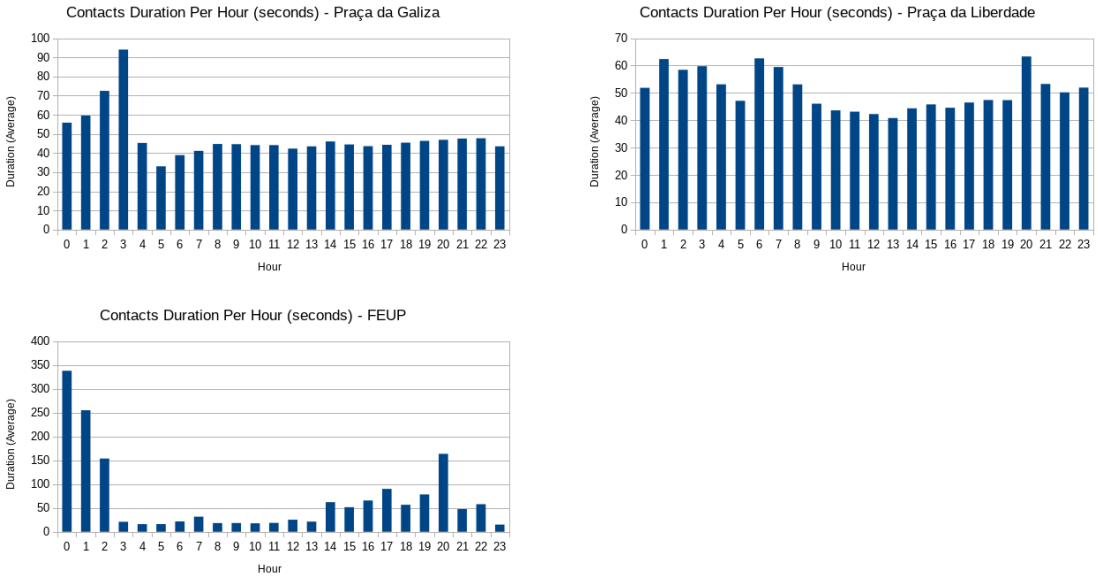
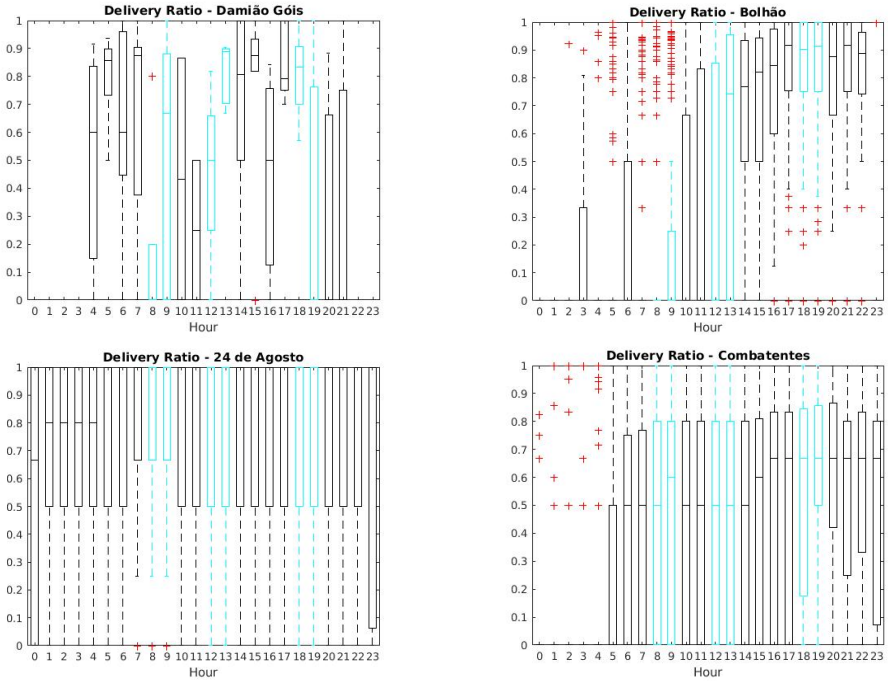


Figure 3.6: Contacts Duration Per Hour in Each DCU

3.3.4 Delivery Ratio

The last link characterization performed is related to the delivery ratio. This indicator was obtained by calculating, for each contact, the ratio between the number of received ACKs and the number of sent bundles.

Once this is the main topic of this dissertation, the results were analysed in more detail. They are presented in the form of boxplots. The hours in blue are considered as peak hours in the STCP case. As it can be concluded, the delivery ratio of the first link is not so high as desired.



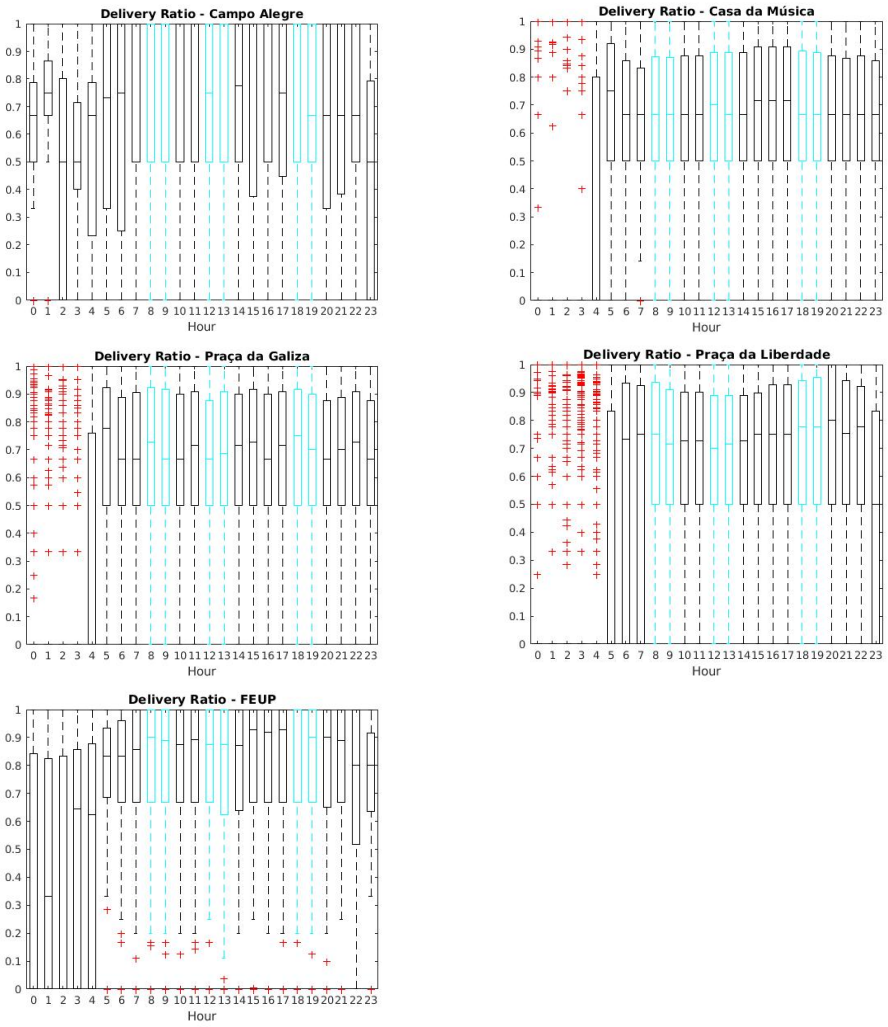


Figure 3.8: Boxplots of the Delivery Ratio Per Hour in Each DCU

Three possible causes might explain the low delivery ratios obtained: the contact duration, the quality of the link and the OBU's load. If the OBU's load is the main reason, the delivery ratio values calculated at peak time would be lower than the ones at off-peak hours. By looking at the boxplots, it does not seem to be the case, but with the available data it is possible to perform tests to either validate or not this hypothesis.

Firstly, a normality test - *Shapiro-Wilk* - was applied to the distributions of the delivery ratio at peak time and off-peak time. For the peak time case, for example, the null and the alternative hypothesis state the following:

- H_0*: The values of the delivery ratio at peak time are normally-distributed*
- H_a*: The values of the delivery ratio at peak time are not normally-distributed*

In every distribution, the null hypothesis was rejected, which validates the alternative hypothesis. For this reason, any subsequent test applied to this data should be a non-parametric test, which does not require the distributions to be normally-distributed.

Finally, in order to either validate or not the influence of the OBU's load on the delivery ratio, the *Mann-Whitney* test, a non-parametric test, was performed. It was used an one-tailed test with the alpha value of 5%. The considered hypothesis were:

- H_0: *The distribution of the delivery ratio of the two groups (at peak time and off-peak time) are equal*
- H_a: *The distribution of the delivery ratio at peak time is stochastically lower than the distribution of the delivery ratio at off-peak time*

The following table sums up the *Mann-Whitney* tests results in each DCU. They will be later discussed in the chapter considerations.

DCU	Result
Damião Góis Id	H_0 rejected
Bolhão	H_0 not rejected
24 de Agosto	H_0 not rejected
Combatentes	H_0 not rejected
Campo Alegre	H_0 not rejected
Casa da Música	H_0 not rejected
Praça da Galiza	H_0 not rejected
Praça da Liberdade	H_0 not rejected
FEUP	H_0 not rejected

Table 3.1: One-tailed *Mann-Whitney* test results

3.4 Chapter Considerations

This chapter analysed the data that was recorded by the DCUs between September 2015 and July 2016. With the information available it was possible to evaluate the performance of the first link (DCU->OBU) regarding the contact duration, the distribution of contacts per hour and the delivery ratio, for example.

Since the most relevant key performance indicator for this dissertation is the delivery ratio, three possible reasons were pointed out to justify the low delivery ratio.

The first one was the OBU load. By analysing the table 6.3, it can be seen that the null hypothesis was not rejected. This conclusion was obtained in every DCU, except in the one placed at Damião Góis. Nevertheless, as outlined previously in subsection 3.3.1, this DCU is not so representative as the others once its available data only refers to the period of four days. The non rejection of the null hypothesis means that there is not enough evidence to reject it and to accept the alternative hypothesis. This means the data could not prove that the peak time influences the delivery ratio of the first link. Therefore, it can be concluded that the OBU's load was not responsible for the low delivery ratio recorded in the first link.

As for the contact duration, it was verified that this indicator has a wide range, it can vary from a few seconds to more than three hundred seconds. In fact, there are contacts that are too short, which might have caused the loss of most of the sent bundles.

The other possible reason introduced was the link quality. This hypothesis could be tested if the same measures were applied in a different link, instead of the CoAP one.

Chapter 4

End-to-End System Design

As evidenced in the previous chapter, there is a lot of information generated by the DCUs that is being lost. This negative result asks for the design of a new system in which the bundles delivery is guaranteed.

As discussed in the section 2.3, the reliability can be ensured either by an end-to-end or by a hop-by-hop mechanism. The former was chosen in the design phase. This way, the bundles will unquestionably be delivered to the cloud, even in the worst case possible. In particular, the failure of an intermediate node (OBU). It is important to note that a hop-by-hop approach would not be sufficient in this situation, as it was discussed in the referred section.

Therefore, an end-to-end system will be implemented. In order to achieve reliability, there are some specific challenges that should be addressed during its design.

Firstly, the system will need to deal with the bundles transmission, retransmission and ACKs generation. These tasks are well defined by an existing sliding window mechanism - the *selective repeat* protoco. The choice of this protocol will become clear later on.

Additionally, for the proper system operation, the ACKs have to be delivered to the DCUs. This challenge, in fact, hides two problems that ask to be solved. Firstly it is necessary to figure out how the ACKs will be delivered and secondly, how they will be addressed. Each generated ACK will be handed over to a specific DCU. However, the DCUs have no fixed IP, they only receive an IP after connecting to an OBU. At that moment, they receive a local IP via DHCP but that IP is no longer valid as soon as the connection is lost. For this reason, no downlink unicast exists. The solution found was to assign an identifier to each DCU. This way, when an OBU receives a bundle from a DCU it has to discover to which DCU is communicating. After that, the OBU knows which ACKs has to deliver. As for the delivery itself, the OBUs will send the ACKs after receiving a bundle from a DCU. The ACKs will be effectively carried in the message-body of the *OK* message that will be sent back to the DCUs.

Lastly, there is another challenge that needs to be reminded. In any reliable system, the sender entity, in the absence of an ACK, will have to retransmit the bundle. However, if the retransmission is too soon, it might represent bandwidth waste. On the other hand, if the retransmission occurs too

late, it means that the retransmission was delayed. This variable should be carefully set because it has a straight impact on the number of retransmissions and therefore on the system efficiency.

To sum up, the main purpose of the system is to deliver end-to-end application-level acknowledgements. This way, the bundles will only be completely removed from the DCUs when those acknowledgements arrive.

4.1 Architecture of the Proposed System

The following schematic outlines the main architecture of the proposed system:

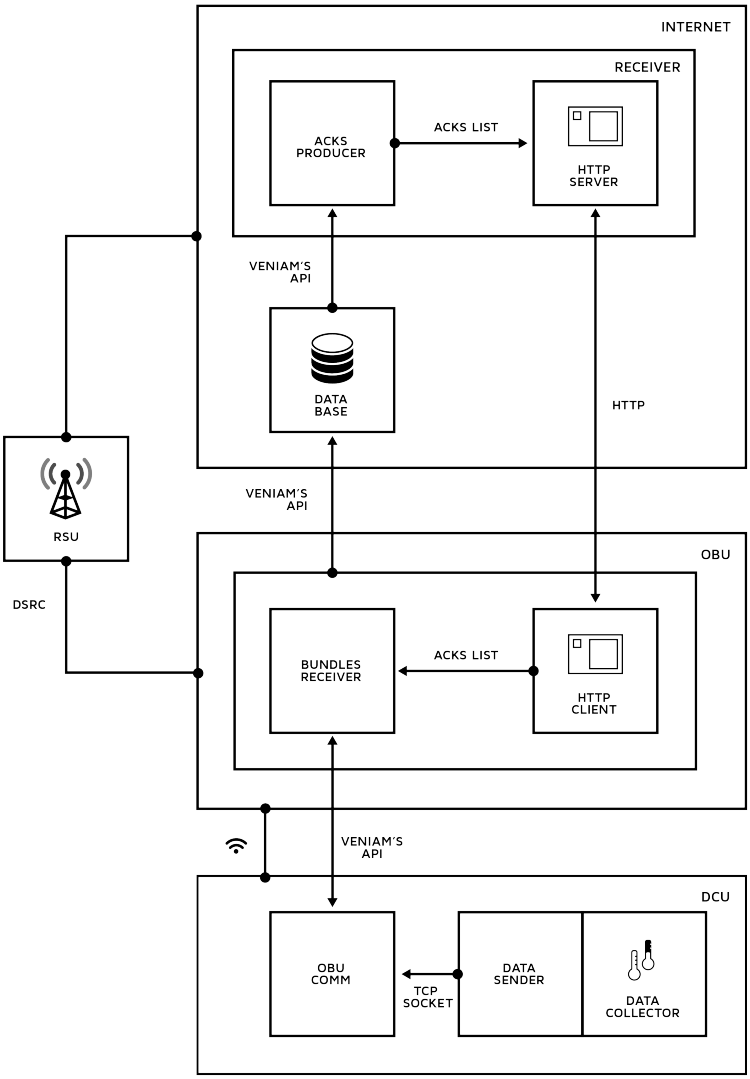


Figure 4.1: Architecture of the Proposed System

As highlighted, there are three main components: the DCU, the OBU and the receiver. The DCU and the OBU are familiar components since they were also presented in the subsection 3.1. As for the receiver, it is an endpoint that will run in a machine with a public IP address and it will be accountable for verifying which bundles have been effectively stored in the database.

In short, the proposed system operates as following: firstly, the *datasender* sends its messages to a service called *OBUComm*, which has the role of a forwarder, as it was the case of the *spencer client*. Thus, the bundles are effectively sent to the OBU by the *OBUComm* service. Afterwards, when the OBU finds an RSU, it offloads the bundles and they are stored in *UrbanSense*'s database. Later on, the receiver looks at the bundles that were recently stored in the database and it generates a list of ACKs accordingly. At the same time, the OBU periodically asks the receiver for the ACKs so that it can deliver them to the DCU in the following established contact.

The existence of the receiver could be questioned. In fact, it seems that a simpler solution would be to have the database itself generating the lists of ACKs. However, this solution is incompatible with the way the *Veniam*'s system is operating. Currently, Veniam provides an API for its clients so they can access to the data they have previously uploaded and that has reached the database. In this way, the existence of a receiver is justified. Using this API, this endpoint is able to find out which bundles have successfully arrived to the database and generate a list of ACKs accordingly.

4.2 Timelines of the System Components

In order to clarify the role of each component in the system, the timeline of each one is presented in the following images. Each one gives an overview of the main events occurring in each system component.

At the DCU side:

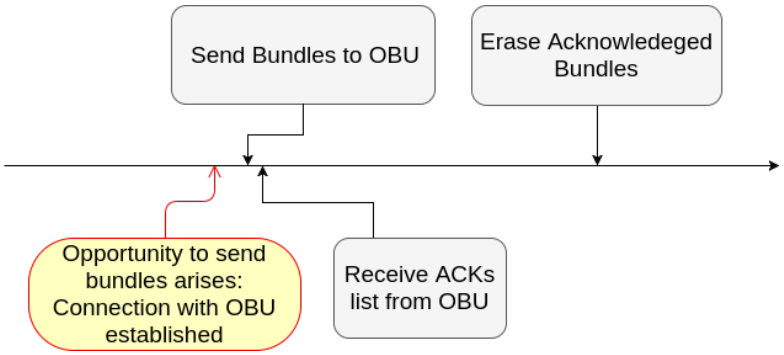


Figure 4.2: DUC Timeline

The DCU waits for an opportunity to send its bundles. As soon as a connection is established, the DCU sends its bundles and, as a response, it gets a list of ACKs. Finally, the DCU can remove from local storage the bundles that were acknowledged.

At the OBU side:

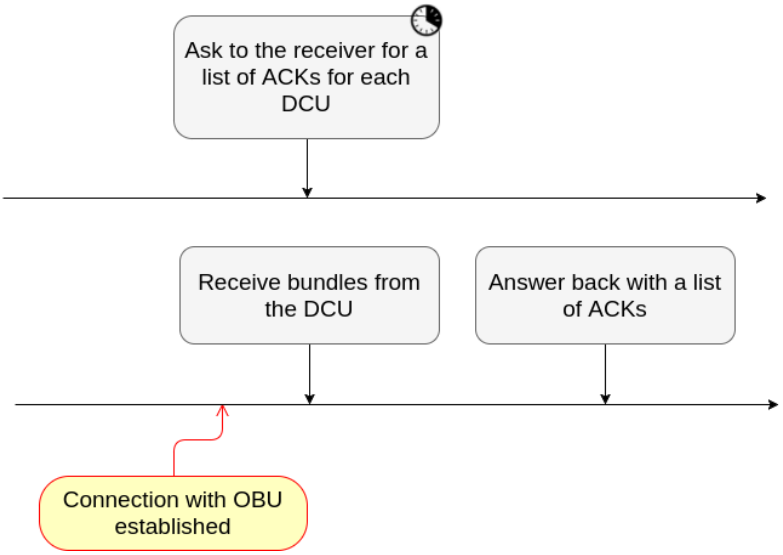


Figure 4.3: OBU Timeline

The OBU timeline was split in two in order to highlight the existence of concurrent events.

The first one is a periodic event and it is related to the communication with the receiver entity. The OBU needs to ask to the receiver for the list of ACKs that was generated for each DCU. This is a periodic event for the sake of simplicity. However, as this is a periodic request, it is done even when the OBU is not within the coverage of an RSU, which means the cellular backhaul is sometimes used. This solution is not free of cost so an improvement can be later applied. For example, perform the request only when the OBU is within the range of an RSU.

At the same time, if a DCU connects to an OBU, the later receives the bundles and sends back the list of ACKs it has previously received from the receiver.

At the receiver side:

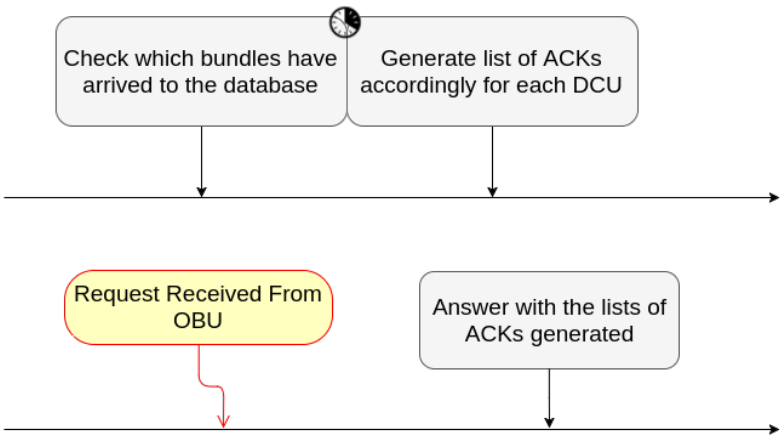


Figure 4.4: Database Timeline

Similarly, the receiver has two concurrent timelines.

The first one is a periodic event. It checks the database in order to find out which bundles have recently been stored. According to that, a list of ACKs is generated for each DCU.

The second one deals with the communication with the OBUs. As soon as a request arrives, the receiver answers with the lists of ACKs that were previously generated.

4.3 Choice of the *Sliding Window* Protocol

The bundles arriving from the OBU are inserted in the database. At this point, two options could be used: acknowledge those bundles either with *cumulative acknowledgements* or *selective acknowledgements*. This decision should take the channel characteristics into account. As analysed in the subsection 3.3.4, the first hop has low delivery ratio so it would be wrong to assume, for example, that if a given bundle has arrived to the database, all the previous ones have also reached the destination. For this reason, selective acknowledgments will be used, which means that every bundle will be independently acknowledged. Therefore, each ACK will be identified by the sequence number of the bundle to which it refers.

This design option is related to one of the sliding window protocols, the *selective repeat*. The other option, *go-back-n* protocol, uses *cumulative acknowledgements*, which is not a good solution for the system being studied. As discussed before, the use of *cumulative acknowledgements* would cause a lot of retransmissions, even of bundles that actually arrived to the database. The immediate disadvantage of this would be the bandwidth waste. On the other hand, the trade-off of the *selective repeat* approach requires extra memory resources at the sender entity, the *window buffers*. Additionally, it adds complexity on both sender and receiver endpoints.

4.3.1 *Selective Repeat* Protocol

The *selective repeat protocol* will be now explained in more detail, as well as some of the design options that differ from its standard implementation.

This is one type of the *sliding window* protocols. As the system uses application level acknowledgements, the protocol will be implemented in the application layer of the OSI model.

As its name suggests, the *sliding window* protocols define a window for the sender and the receiver endpoints. On the sender side, there is a window of sequence numbers that are allowed to be sent. Similarly, on the receiver side, there is a window for the messages that can be accepted as valid.

Here is how the protocol operates: the sender starts sending a number of bundles that can go from zero to window size minus one. It starts an individual timer for each bundle and it stores them in the window. Each element of the window is, in fact, a buffer that keeps a bundle that had already been sent but not acknowledged yet. The number of buffers is equal to the window size. Possibly, the bundles are likely to arrive to the receiver. This endpoint will send back to the sender an individual ACK for each bundle, whenever the received bundle fits in its window. At the time the sender receives the ACKs, it can finally remove the acknowledged bundles from the window,

"slide" its window and send new bundles. If, for some reason, the bundles or the ACKs are lost, the timer of the bundles will eventually expire and the sender will resend the bundles [42].

The protocol frequently uses NACKs, which are sent by the receiver to notify the sender that something went wrong with a given bundle. When receiving this message, the sender does not need to wait for the timer to expire, it can immediately resend the lost/damaged bundle [42]. However, the system in study relies on an opportunistic scenario, which means that the DCUs will take advantage of every contact to send new bundles and resend the ones that were not acknowledged. For this reason, the reception of a NACK would not change this scenario. Thus, the use of NACKs would bring a disadvantage to the system since it would represent a bandwidth waste.

Since the bundles will be independently acknowledged, it is important to carefully identify each one. Therefore, one of the design options of this protocol is to decide the number of bits that will be devoted to the sequence numbers. This field will be appended to the data itself. As a result, the size of the bundles is increased, which has an impact in the amount of bandwidth consumed to transmit each bundle.

Thus, if an N-bit field for representing the sequence numbers is defined, that means the maximum sequence number that can be sent is $MAX\_SEQ = 2^N - 1$. Since there are N bits available, it could be concluded that the window's size of both sender and receiver was MAX_SEQ. However, that is not possible. The following example will illustrate the reason behind that. The images were adapted from the animation available in [1].

In the image below there is a sender that, at instant t_0, starts sending its bundles to the receiver. The sender and the receiver window size is four and the MAX_SEQ is also four.

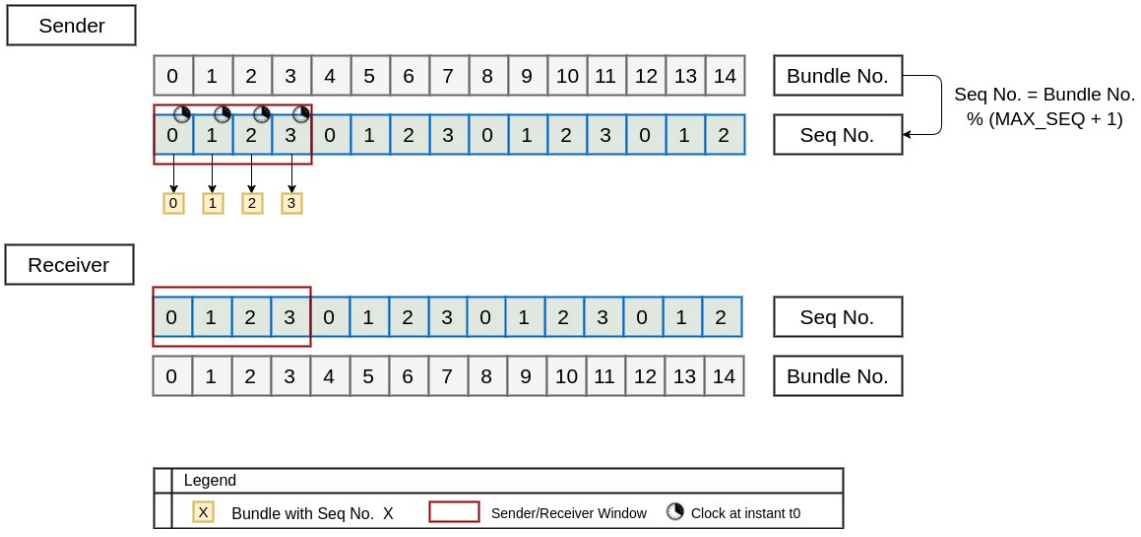


Figure 4.5: Incorrect Example of the *Sliding Window Protocol*, instant t_0

Later, at instant t_1, the receiver receives the bundles. Since the sequence numbers of these bundles fit in its window, the receiver generates the ACKs, identified with the sequence numbers

[1] CCS LABS, Johannes Kessler, 2012. URL: http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/

of the bundles to which they refer, and it sends them to the sender. However, the ACKs are lost on its path for some reason.

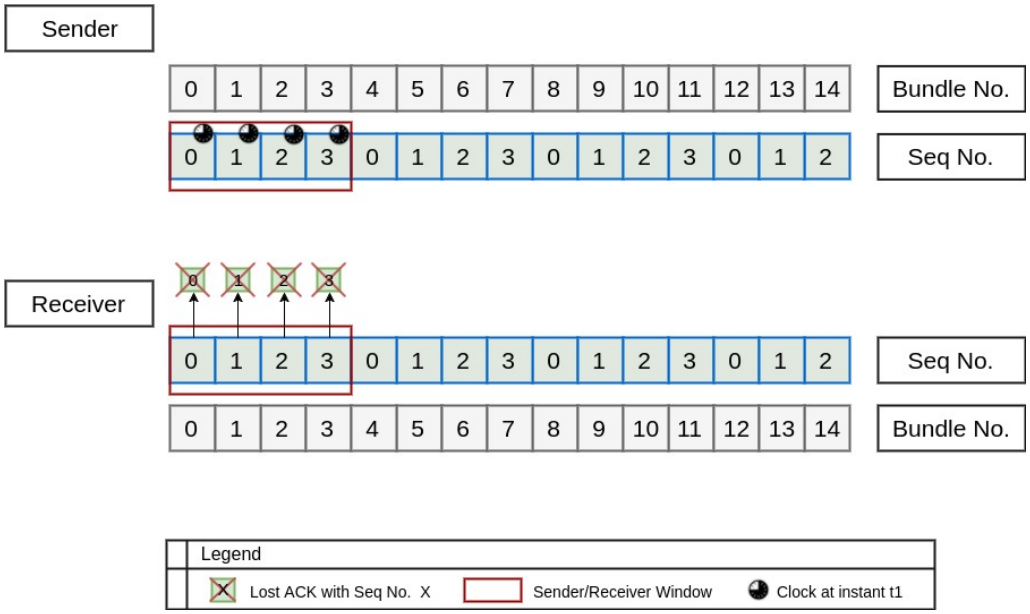


Figure 4.6: Incorrect Example of the *Sliding Window Protocol*, instant t_1

Since the receiver is unaware of the ACKs lost, it "slides" its window and starts waiting for the next bundles. As no ACKs were received on the sender side, the timers expire and the bundles are retransmitted at instant t_2.

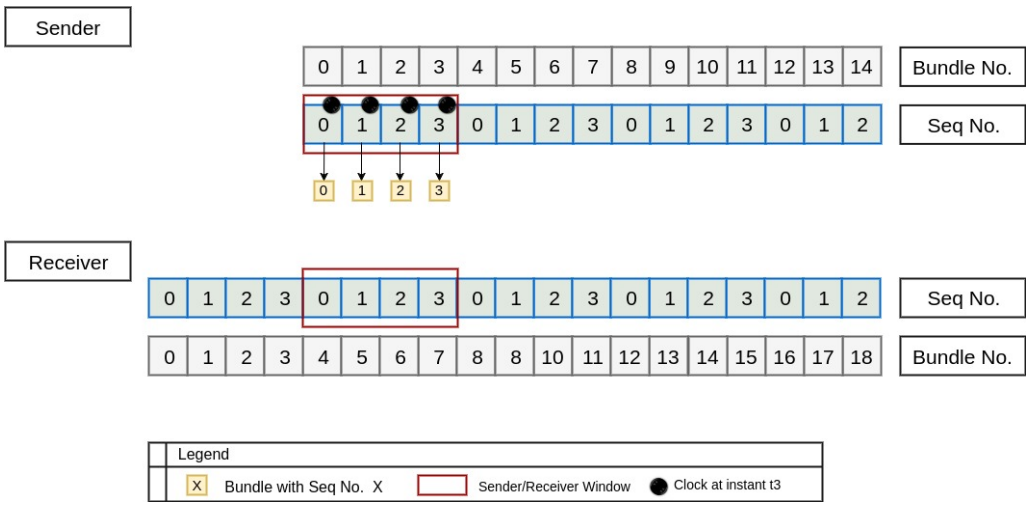


Figure 4.7: Incorrect Example of the *Sliding Window Protocol*, instant t_2

Afterwards, when the retransmitted bundles arrive, the receiver has no way of knowing that these bundles are duplicates. It will look at them as if they were new bundles.

For this reason, the range of a new window on the server side cannot contain sequence numbers that were acknowledged in the immediately preceding window. The way to guarantee this is to define $WINDOW\_SIZE = (MAX\_SEQ + 1)/2$.

4.3.2 N-bit field for sequence numbers

In order to compute the number of bits needed for representing the sequence numbers field, the window size will be firstly calculated. The maximum efficiency of the *sliding window protocol* can be achieved if the link utilization is at its maximum. To accomplish that, the number of sent bundles should be the maximum number of bundles that can fit inside the channel. This value is given by the bandwidth-delay product (BD), which is calculated as follows:

$$BD = B \times OWD \tag{4.1}$$

where B is the bandwidth in bits/sec and OWD is the one-way delay (or one-way transit time) [42].

However, in the system in question, the maximum performance will be accomplished if the entire duration of a contact is used to send bundles. It would not be reasonable for a DCU, during a contact with an OBU, to stop sending bundles because the number of the allowed outstanding bundles (window size) was reached. For this reason, the above equation should be replaced by:

$$BD = B \times Dur_{Max} \tag{4.2}$$

where B is the bandwidth in bundles/sec and Dur_{Max} is the maximum duration of the contacts in the DCU in question.

The bandwidth can be given by the throughput - number of acknowledges received divided by the duration of each contact. The data analysed in the chapter 3 can be used to calculate it. Since it is important to ensure that, even the best performance of the link is fully used, the maximum value of the throughput will be chosen for the calculations. These results are presented in the appendix A. As for the maximum duration of the contacts, it can be obtained from the graphs in the subsection 3.3.3. The exact values are listed in the appendix A as well.

The product of these two variables gives the maximum number of bundles that can be sent from the DCU to the OBU during each contact. In fact, this result reflects the window size. As discussed before, the maximum sequence number is given by:

$$MAX\_SEQ = WINDOW\_SIZE \times 2 - 1 \tag{4.3}$$

And finally, the number of bits needed for representing the sequence numbers is:

$$N = \log_2 (MAX\_SEQ) \tag{4.4}$$

As highlighted in the subsection 3.3.1, the DCU placed in Praça da Galiza is the one which has more recorded contacts. The calculation of the N-bit field for the sequence numbers will be exemplified for this DCU.

Thus, for this DCU, the bandwidth-delay (which will correspond to the window size) is:

$$BD = B \times Dur_{Max} = 2,11 \times 92 = 194 bundles$$

The maximum sequence number is:

$$MAX\_SEQ = WINDOW\_SIZE \times 2 - 1 = 194 \times 2 - 1 = 387$$

Finally, the number of bits for the N-bit field is:

$$N = \log_2 (MAX\_SEQ) = \log_2 (387) \simeq 8,57 \rightarrow 9 bits$$

which allows a maximum sequence number of:

$$MAX\_SEQ = 2^N - 1 = 2^9 - 1 = 511$$

The window should be recalculated in order to get its value as a power of two:

$$WINDOW\_SIZE = \frac{MAX\_SEQ + 1}{2} = \frac{512}{2} = 256$$

4.4 Fields of the Bundles

A sum up of the fields of the bundles will now be presented . As mentioned before, apart from the data itself, the bundles are also formed by other fields that enable the ACKs management and the bundles retransmission. They are the following:

- Data - carries the sensors samples
- Sequence Number - identifies each particular bundle
- DCU Id - identifies one DCU among the total of 19

The point of having the DCU Id field is that, although there is a sequence number which identifies a given bundle, there might be more than one outstanding bundle with the same sequence number. This happens with bundles that were generated in different DCUs. In order to solve this ambiguity, the bundles will also identify the DCU where they were generated.

In short, each bundle is a variable-length message, depending on the data it carries. As an example, the following table outlines the number of bits reserved for each field of the bundle in the DCU placed at Praça da Galiza:

Field	Number of Bits
DCU Id	5
Sequence Number	9
Data	Variable

Table 4.1: Bundle's Fields for the DCU placed at Praça da Galiza

4.5 Retransmission Timeout

The RTO is another system parameter that needs to be carefully assigned. In fact, in order to achieve the maximum system efficiency, it is desired that the number of retransmissions is the minimum possible, which could be achieved with a larger RTO. On the other hand, a larger RTO would cause the delay of the bundles loss detection. For these reasons, a balance should be found between the number of retransmissions and the time required to detect the loss of a bundle.

Different approaches for the RTO estimation could be applied to the system in study. Each one will be conceptually addressed, as well as their advantages and disadvantages.

4.5.1 Static RTO

The most simple approach would be to have a static RTO. The variable would be set offline and would not change during the system execution. Its value would be equal to the summation of the highest response times of each system component.

The implementation of this approach is simple but it does not shape according to the system conditions. Moreover, the delays added by the DTN are very difficult to predict. For this reason, it is most likely that this solution will have a higher number of retransmissions.

4.5.2 Dynamic RTO based on the RTT

A second alternative would be to start with a static RTO for the first bundles. From that point onwards, the RTTs calculated for the previously acknowledged bundles could be used to determine new estimations for the RTO.

This option has the disadvantage that the computed RTTs will include not only the time needed to effectively carry the bundle and the ACK, but also the time added by the delay tolerant network, which can be highly variable.

4.5.3 RTO based on the number of established contacts

Another RTO estimation could be based on the number of contacts. The main reasoning of this approach would be the following: if a bundle is transmitted in a given contact, it could be expected that, in the following contact, the next OBU would have the correspondent ACK available. If no ACK was received in that contact, it could be concluded that the bundle was lost. For that reason, the subsequent contact would correspond to the best moment for the retransmission.

Thus, the DCU starts sending its bundles in the first contact. In the second contact, it expects to receive the ACKs of the bundles sent in the first contact. However, in order to receive them, the DCU needs to retransmit at least one of them. Thus, the first window bundle is retransmitted during the second contact. Finally, the bundles that were not acknowledged will be retransmitted in the third contact.

In general, the bundles are sent in any contact. On the other hand, the retransmission of the bundles will only occur during odd contacts, with the exception of the first window bundle that is retransmitted in even contacts.

The implementation should also be prepared to deal with long contacts. During these contacts, the bundles are retransmitted according to an RTO, that could be either static or dynamic. As for too short contacts, caused by some instantaneous interruptions, they should not be interpreted as new contacts, because that miss-interpretation would trigger improper retransmissions and decrease the system efficiency.

This solution is likely to have a good efficiency in the DCUs placed at areas within the coverage of an RSU. In these situations, the OBU can immediately offload the bundles to the RSU. On the other hand, if messages are stored in cache in the OBUs, the efficiency will decrease because the assumption that the bundle sent in a contact will have its ACK available in the following contact might not be true.

4.5.4 RTO based on the number of contacts required to receive an ACK

Two improvements could be performed to the previous approach in order to make it more suitable for the DTN.

Firstly, the RTO estimation could be similarly performed based on the number of established contacts. However, rather than assuming the ACK is available on the second contact, it could be made an analysis on the already acknowledged bundles. For each one, it could be recorded the number of established contacts since the bundle transmission to the ACK reception. That information seems to provide a better estimate for the RTO.

Besides the RTO estimation, there is an alternative for the window size design that could lead to even better results. In the calculation of the bandwidth-delay, the duration of a single contact should not be used. Instead, it could be considered the duration of a contact multiplied by the expected number of contacts needed until the first ACK is received. This way, the DCU will be continuously sending bundles, without wasting established contacts. Moreover, the window size calculation could be performed during the system execution, which would lead to a window size that shapes according to the service provided by the vehicular network.

4.6 Chapter Considerations

This chapter outlined the main design options adopted in order to ensure end-to-end reliability.

To sum up, the main system components will be the DCU, the OBU and the receiver. The *selective repeat* protocol will be implemented at the application level.

Additionally, there will be some “control” information that will be appended to the sensors data. Thus, the bundles will be formed not only by the sensors data, but also by the bundle sequence number and the DCU ID.

Finally, the chapter addressed the possible RTOs estimation. Each one will be tested in the experiments presented in the results chapter.

Chapter 5

End-to-End System Implementation

This chapter provides details on the prototype implementation. Initially, the *OBUComm* module that replaces the *spencer client* on the DCU is detailed, followed by the receiver module on the cloud server, and finally the changes to the local API on the OBU.

5.1 OBUComm

On the proposed implementation it was decided to maintain the main architecture on the DCU side. As it can be seen in the following images, the *spencer client* was replaced by the *OBUComm*. They both act as bundle forwarders. However, in the proposed implementation, the *data sender* has less responsibilities since the management of the bundles (their retransmission and the ACKs reception) is now done by the *OBUComm*.

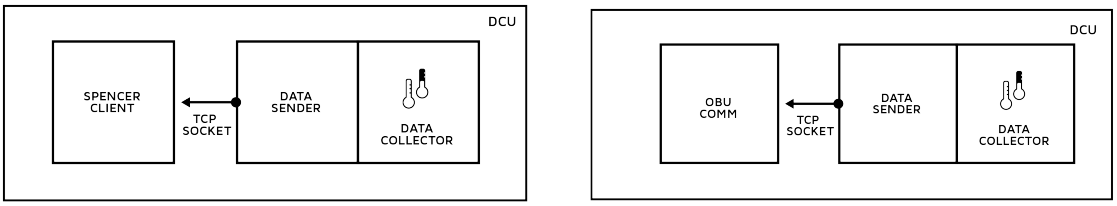


Figure 5.1: Previous and Current DCU components

Another option would be to include the *sliding window* protocol in the *datasender*. This way, the bundles could be directly transmitted to the OBU, without the need for a forwarder. However, by having the *OBUComm*, an independent entity responsible for applying the *sliding window protocol* and the bundles transmission, it is possible, for any service, either the *datasender* or another one, to have their data being reliably sent to the cloud. The only requirements to the use of the *OBUComm* service are:

- a TCP socket should be used to send the messages to this service;
- the message format should be json string.

The *OBUComm* has two different buffers. The first one is a queue and it stores the bundles which are waiting for an opportunity to be sent to an OBU. The second one is actually not one, but a set of buffers, which are responsible for saving the bundles that were already sent but not acknowledged yet. The number of buffers corresponds to the number of the allowed outstanding bundles. They will be further mentioned as *window buffers*.

This service runs three main threads. The following is an explanation of each one.

5.1.1 Connection State Thread

This thread is the simplest one. Each second, it tries to ping an OBU in order to conclude if the current state is connected or not connected. Therefore, to avoid pointless transmissions, the bundles will only be sent or retransmitted to the OBUs if the state is connected.

5.1.2 Receiver Thread of the Bundles

This thread establishes a TCP socket with the *datasender* and it waits for incoming messages. Each message received is immediately added to a queue, where it will be stored until there is a free position in the window.

In order to prevent the DCU to run out of memory, a maximum number of bundles that could wait in the queue was established. If the queue is full and a new message arrives from the *datasender*, this message will be discarded.

The queue dimension was performed by inspecting the amount of free storage of a DCU. Since the DCUs are running the same services, this probably does not significantly vary from DCU to DCU. It was then decided that only 5% of that space would be used for storing the queue elements. That result was divided by an estimation of each bundle size (section 4.4) and the maximum number of queue elements was obtained.

5.1.3 Window Management Thread

This thread is the most complex. It deals with the transmission/retransmission of the bundles, the reception of the ACKs and the subsequent window management.

The Window Management Thread runs a cycle which will be executing the following steps: firstly, if there are free positions in the window, the messages waiting for an opportunity to be sent are picked up from the queue. A sequence number and the DCU ID are appended to them. Each one is stored in a different position of the window and they are sent to the OBU through a POST request. Simultaneously, the timers of these window positions are initiated. If the reception is successful, a *201 OK* message is received. Besides, if the OBU has ACKs to deliver to the DCU, an ACKs list is received in the message-body of the *201 OK* message. Thus, the thread in question is also responsible for looking at the arrived ACKs list and managing the window accordingly.

In order to facilitate the implementation of the *selective repeat* protocol, there are some fields associated to each window position:

- *bundle* (string) - json string message picked up from the queue, with an appended DCU ID and a sequence number;
- *seqNr* (int) - equal to the sequence number that was appended to the field *bundle*;
- *acked* (boolean) - equal to true if the correspondent ACK has already been received; equal to false if not;
- *nRetransm* (int) - counter that calculates the number of times the bundle has already been retransmitted;
- *timeSent* (int) - epoch time when the bundle was transmitted or retransmitted.
- *timeFstTransm* (int) - epoch time when the bundle was transmitted for the first time. This variable will be used to calculate the time it was needed for the correspondent ACK to be received.

Thus, in each cycle, the *acked* element of each position will be checked. If the first elements of the window were acknowledged, their *bundle* field can be cleaned and the window can be shifted up.

An example will be presented in order to illustrate the window evolution according to the reception of a given list of ACKs. For the sake of simplicity, some of the above fields will not be represented. As for the *bundle* field, it will also be presented in an incomplete version. In this example, the $WINDOW\_SIZE = 4$ will be considered.

At instant t_0, the bundles with the sequence numbers $0, 1, 2, 3$ have already been sent. None of them has been acknowledged yet.

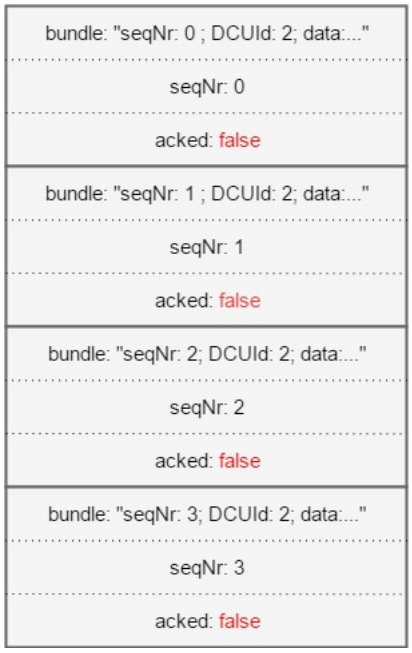


Figure 5.2: Example of the *OBUComm* window, instant t_0

Later on, the timer of the first bundle expires and the bundle is resent. In the message-body of the *201 Ok* message, the following list of ACKs is received: "acks_list: 1,2". At this moment, the DCU knows the bundles with the sequence numbers 1 and 2 were already stored in the database, thus the correspondent *acked* field can be set to true. However, the window positions will remain the same. The window can only be shifted up if the first element has already been acknowledged; in this case, the bundle with the sequence number 0. Thus, the window will remain as following:

Figure 5.3: Example of the *OBUComm* window, instant t_1

In the meanwhile, the bundle with the sequence number 3 is retransmitted and a new list of ACKs arrives: "acks_list: 0". The following two images present the evolution of the window that this received ACK will enable:

Figure 5.4: Example of the *OBUComm* window, instant t_2 and t_3

Finally, at instant t_3, if the queue is not empty, three new bundles can be picked up and added to the three free positions of the window.

To sum up all this thread's cycle, the following flowchart is presented:

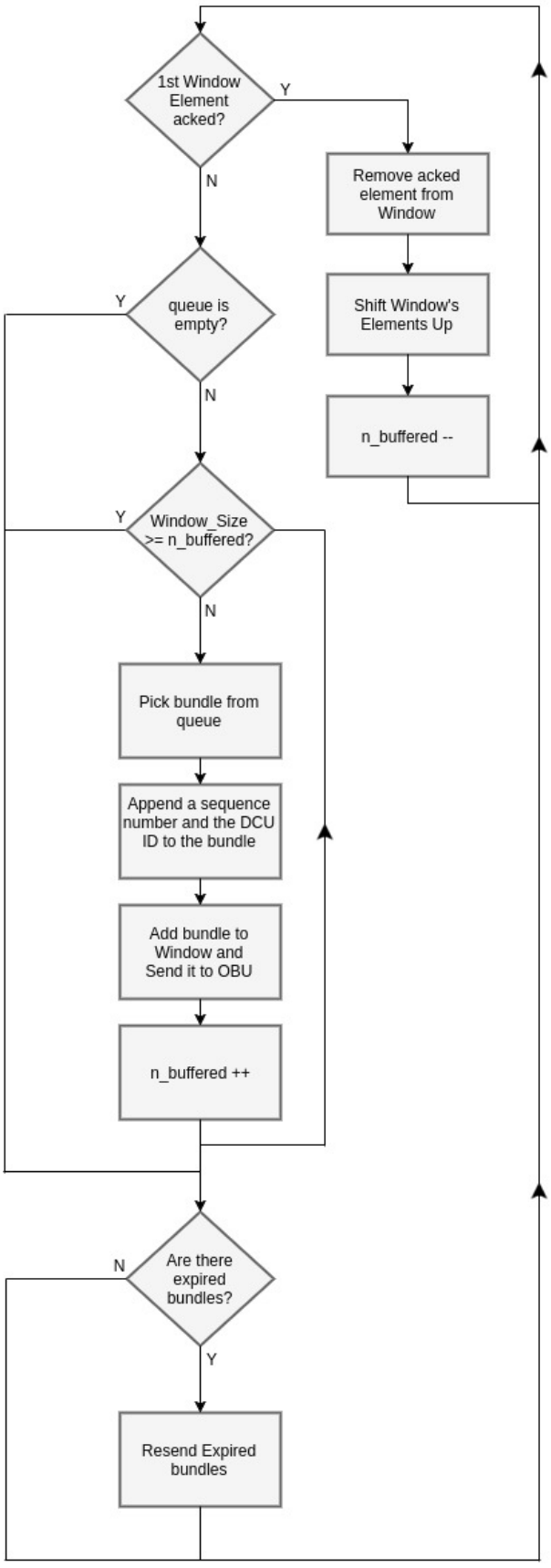


Figure 5.5: *OBUComm* Flowchart

5.2 Receiver

The receiver implementation is simpler. This entity has mainly two tasks. The first one is to generate the lists of ACKs for each DCU and the second one is to send those lists to the OBUs. These tasks will be implemented by two threads.

5.2.1 ACKs Producer Thread

This is a periodic task which looks at the messages stored in the database in order to generate an list of ACKs for each DCU of the system. As illustrated in the examples regarding the *selective repeat* protocol of the subsection 4.3.1, the receiver has a window for the messages that can be accepted as valid. Since the system is composed for more than one DCU, the receiver should have a window for each one. Thus, the receiver will have a data structure - *DCU_Windows* - to save the information related to the window of each DCU. Each element of this data structure is formed by:

- the *DCU ID* (int);
- the *window* itself (data structure);
- the *ACKs list* (list) - generated for the DCU in question.

On its turn, the *window* field is a data structure as well. Each of its elements contains the fields:

- *seqNr* (int) - sequence number that is expected to be received;
- *acked* (boolean) - it indicates if the bundle with the *seqNr* has already arrived or not.

In order to generate an ACKs list for each DCU, the receiver needs to know which bundles have been stored in the database. For that, it periodically makes a GET request (defined by the Veniam's API) with two query arguments - *tsinit* and *tsend* - that allow the specific range of time in interest to be specified. As a result, it gets all the messages stored in the database during the mentioned period. They are then filtered by the DCU ID so that an ACKs list can be generated for each DCU.

This process will be illustrated with an example. The following parameters will be considered: $WINDOW\_SIZE = 4$ and $MAX\_SEQ = 7$. The steps presented will be for the DCU with the $DCU\_ID = 2$. Obviously, the process will be the same for all the DCUs.

At instant t_0, the element of the *DCU_Windows* which refers to the DCU with the $DCU\_ID = 2$ is the following:

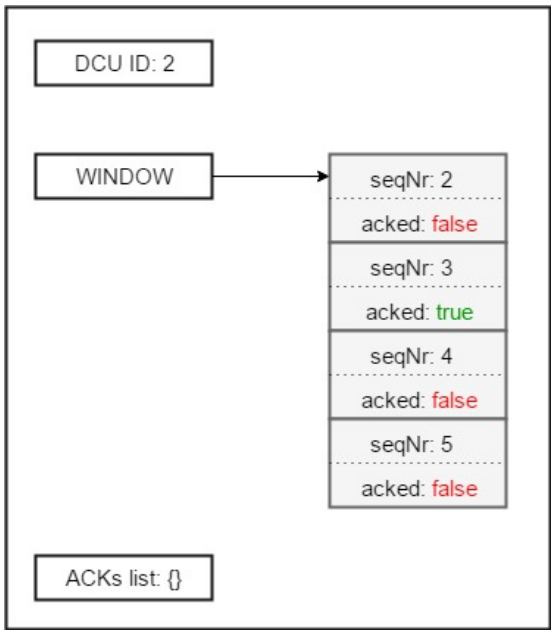


Figure 5.6: Example of one of the *DCU_Windows* elements, instant t_0

The *ACKs Producer Thread* then makes a GET request in order to discover which bundles have been stored in the database:

http://api.veniam.com/api/v2.5/local/urbanSense/events?tsinit=2017-06-02T10:19:54.000Z&tsend=2017-06-02T10:20:15.313Z

The response it receives is the following:


```
[{"_id":":test_device::2017-06-02T10:20:05.259Z","_data":"{\"
    temperature\":12,\"seqNr\":3,\"DCU_ID\":3}","_context":{"
    device_id":":test_device","sys_time":"2017-06-02T10
    :20:05.000Z","netrider_id":2297,"serial_id":"VENIAM_2297
    "}},{"_id":":test_device::2017-06-02T10:20:08.313Z","_data
    ":"{\"temperature\":31,\"seqNr\":0,\"DCU_ID\":3}","_context
    ":{"device_id":":test_device","sys_time":"2017-06-02T10
    :20:08.000Z","netrider_id":2297,"serial_id":"VENIAM_2297
    "}},{"_id":":test_device::2017-06-02T10:20:08.452Z","_data
    ":"{\"temperature\":5,\"seqNr\":2,\"DCU_ID\":2}","_context
    ":{"device_id":":test_device","sys_time":"2017-06-02T10
    :20:08.000Z","netrider_id":2297,"serial_id":"VENIAM_2297
    "}},{"_id":":test_device::2017-06-02T10:19:54.111Z","_data
    ":"{\"temperature\":15,\"seqNr\":3,\"DCU_ID\":2}","_context
    ":{"device_id":":test_device","sys_time":"2017-06-02T10
    :19:54.000Z","netrider_id":2297,"serial_id":"VENIAM_2297
    "}},{"_id":":test_device::2017-06-02T10:19:54.170Z","_data
    ":"{\"temperature\":30,\"seqNr\":4,\"DCU_ID\":2}","_context
    ":{"device_id":":test_device","sys_time":"2017-06-02T10
    :19:54.000Z","netrider_id":2297,"serial_id":"VENIAM_2297"}}]
```


As it can be seen, the response is a list of all the messages that were saved in the database, filtered by the period specified in the query arguments. It is worth mentioning that the only information sent from the DCU was the *_data* field. All the other fields were added by the service running on the OBU, implemented by Veniam, which is responsible for receiving the bundles.

After filtering the response by the *_data* field, the receiver gets the following bundles:


```
"{\"temperature\": 12, \"seqNr\":3, \"DCU_ID\":3}"
"{\"temperature\":31, \"seqNr\":0, \"DCU_ID\":3}"
"{\"temperature\":5, \"seqNr\":2, \"DCU_ID\":2}"
"{\"temperature\":15, \"seqNr\":3, \"DCU_ID\":2}"
"{\"temperature\":30, \"seqNr\":4, \"DCU_ID\":2}"
```


With the above information, for the DCU with $DCU\_ID = 2$, the receiver knows that the bundles with the sequence numbers 2, 3 and 4 have already been stored in the database. Taking the *window* of the figure 5.6 into account, the receiver will append the numbers 2 and 4 to the ACKs list. As it can be seen in the figure 5.6, the bundle with the sequence number 3 has the *acked* field equal to true, which means the above bundle is a duplicate. However, the receiver should also append the number 3 to the ACKs list because the *OBUComm* might have not received this ACK.

The following two images illustrate the *window* evolution after the analysis of the GET response. The *ACKs list* field might increase in subsequent requests. Its content is only cleaned when the list is sent to an OBU.

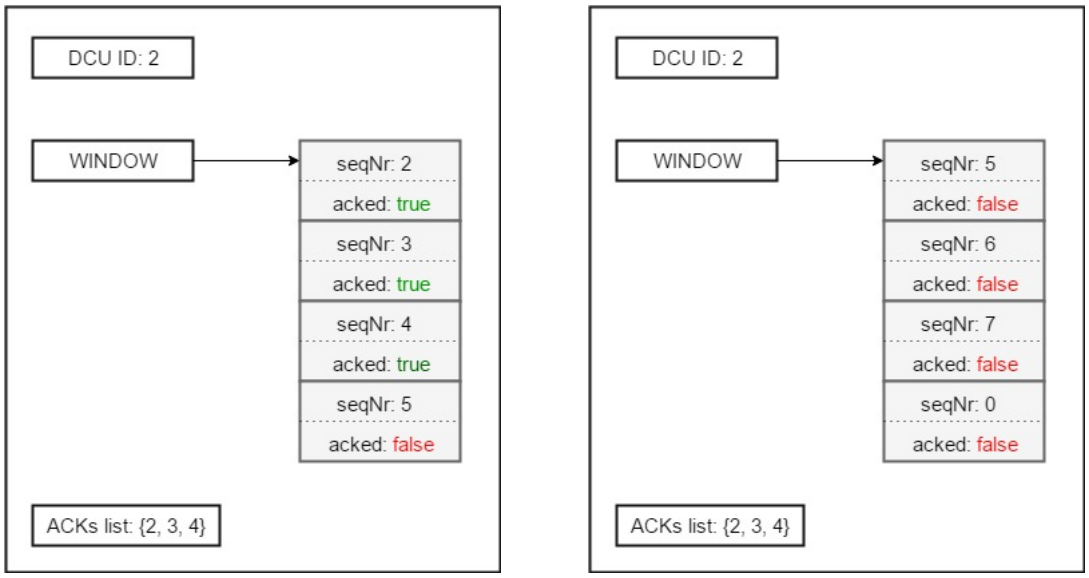


Figure 5.7: Example of one of the *DCU_Windows* elements, instant t_1 and t_2

5.2.2 ACKs Sender

The second thread acts as an HTTP server. It waits for GET requests from the OBUs and, as a response, it sends the ACKs lists generated by the *ACKs Producer* thread. Each GET request from the OBUs specifies the DCI_ID so that the receiver knows which ACK list to send.

To enable the communication between the receiver and the OBUs, this endpoint will be running on a machine with a public IP address.

5.3 Implementation in the OBUs

The OBUs will have two roles in this system. Firstly, they will be in charge of communicating with the *receiver* in order to get a list of ACKs for each DCU. Secondly, they are accountable for receiving the bundles sent from the DCUs and sending back the ACKs list for that specific DCU.

5.3.1 Requesting ACKs Thread

This is a thread which acts as an HTTP client. Its role is to periodically ask to the *receiver* for the lists of ACKs. In the current implementation, the OBUs ask for the list of ACKs of each DCU. For example, if there are 19 DCUs available, the OBU will make 19 requests to the receiver, which one specifying the DCU in interest for that particular request. This was implemented in this way so that, in the future, some intelligence can be added in these requests, as it will be later discussed in the section 7.2.

The OBU has a file for each DCU. The *receiver* responses are appended to the existing contents of each file. The contents of the files are cleaned when their data is sent to the DCU in question.

5.3.2 Veniam's Local API

Veniam removed the *Spencer server*, which used CoAP, from the OBUs in November 2016. As an alternative, it developed a local API, which uses HTTP2, in order to allow the client's devices to upload data when connected to Veniam's network. The data upload is done through a POST request. This is almost enough for the system in study.

However, as mentioned above, the OBU should respond back to the DCUs with an ACKs list, in case of a proper request. These were the changes that were performed on the local API. Every time a message is received, the DCU looks for the *DCI_ID* field in the received bundle. After knowing which DCU is communicating with, it reads the content of the correspondent file and adds it to the message-body of the *201 OK* message that sends to the DCU.

Just to exemplify, here is the output at the DCU side, when a bundle is sent:

```
SEND BUNDLE {'temperature': 19, 'seqNr': 3, 'DCU_ID': 2}
Response from server:
201
['0, 1']
```

The first line was printed after the displayed bundle was sent. The message itself is just an example. The fields *seqNr* and *DCU_ID* are the relevant ones. The last lines display the response got from the DCU. It was received the 201 code and the list of ACKs $\{0, 1\}$.

5.4 Used Technologies

Three different elements in the system were implemented. The *OBUComm*, running on the DCU, was implemented in python. The receiver, running on the backoffice, was also implemented in python. The changes in the local API on the OBU were done in C, the language used by Veniam in the OBUs. Additionally, it was also used the command tool *curl* in order to test the implementation in the OBU.

5.5 Chapter Considerations

This chapter looked into the implementation of the three main elements of the end-to-end system.

The *OBUComm* has the role of a sender. It is implemented by three threads. The first one figures out whether the DCU is connected to an OBU. The second receives the messages created by a "message generator", also running on the DCU, such as the *data sender*. The third one deals with the communication with the OBU, by transmitting or retransmitting the bundles and by receiving the list of ACKs.

The receiver is a service with a public IP address, so that the OBUs can communicate with it. This endpoint periodically checks which messages were stored in the database and it generates the lists of ACKs accordingly. At the same time, it answers to the OBU requests by sending them the lists of ACKs.

Finally, the OBU receives the bundles from the DCUs. It looks for the DCU ID in order to understand to which DCU is talking to. After that, it sends back the list of ACKs that was generated by the receiver for that specific DCU.

Chapter 6

Results of the End-to-End System Experiments

The end-to-end system was tested in two different environments. Firstly, some experiments were performed in a controlled environment, with the OBU and the DCU used during the implementation phase. The variations of the RTO estimation presented in the section 4.5 were tested. The one with the best results was also tested in a more realistic environment, with one of the DCUs available in Porto (24 de Agosto) and with the same OBU.

In both locations, the backhaul used by the OBU was mainly the cellular. Tests using the DTN are not feasible for these experiments because when the OBUs cannot forward the bundles to an RSU or to another OBU, they add them to its cache. The OBUs will only retry to forward them 24 hours later. The only way possible to perform the tests was to set the bundles with a critical priority. In this way, if the OBU has no way of offloading the bundles immediately, it uses the cellular backhaul.

Additionally, it is worth mentioning that it was not possible to perform tests in the STCP bus fleet because the implementation in the OBUs would have to go through a validation process, such as a code review phase. Since the validation process is long, it would not be possible to finish it in time, given the dissertation deadline.

6.1 Performance Metrics

During the experiments, the following performance metrics were recorded:

- for each successfully sent bundle, the number of retransmissions that occurred before the ACK was received;
- the number of discarded/lost bundles;
- the evolution of the estimated values regarding the retransmission timeout.

6.2 Test Results in Controlled Environment

6.2.1 Experiments Setup

The following map shows the area where the experiments were performed. The line in blue indicates the path followed by the car which was carrying the OBU. The circle in red signals the location where the DCU was placed.

Figure 6.1: Location of the first experiments

The experiments had approximately the same duration, between 20 and 25 minutes.

As for the system settings, a 5-bit field was chosen for representing the sequence numbers. A larger number was not chosen in order to facilitate the logs analysis in real time.

6.2.2 Static RTO

This test used a static RTO and it was performed to evaluate the RTO design option presented in the subsection 4.5.1.

The RTO was set to 16 seconds. The value was obtained by the summation of the longest response times of each system element. Starting on the receiver side, the *ACKs Producer Thread* runs a cycle each 8 seconds, which means that, in the worst case, it will produce an ACK with 8 seconds of delay. As for the OBU, the *Requesting ACKs Thread* asks for the lists of ACKs to the receiver every 8 seconds. Thus, if the OBU and the DCU are connected, the DCU can upload a bundle at a given instant and, 16 seconds later, the correspondent ACK will be available in the OBU.

The following table sums up the results obtained for the number of retransmissions needed until the ACK reception.

Number of Retransmissions	Percentage (%)
0	29,6
1	40,9
2	21,6
3	5,7
4	2,2

Table 6.1: Test Results of static RTO - Number of bundles retransmissions

As expected, the efficiency obtained regarding the number of retransmissions was not so good as desired. The following alternatives will add some improvements in order to increase the percentage of the 0 retransmissions.

6.2.3 Dynamic RTO based on RTT

This experiment used a dynamic RTO, it implemented the alternative discussed in the subsection 4.5.2.

In this case, the RTO was computed online, during the program execution. The first bundles (in number equal to the window size) were retransmitted according to the static RTO used previously. From that point onwards, the RTO was estimated every time a new set of bundles were acknowledged. The RTO was then computed based on the round trip times calculated for those acknowledged bundles. The RTTs were ordered and the lowest/highest 5% values were discarded. Finally, the RTO estimation was the average of the remaining values.

The following table and chart summarize the results obtained for the number of bundles retransmissions and for the evolution of the RTO estimation.

Number of Retransmissions	Percentage (%)
0	20,8
1	72,7
2	6,5

Table 6.2: Test Results of dynamic RTO - Number of bundles retransmissions

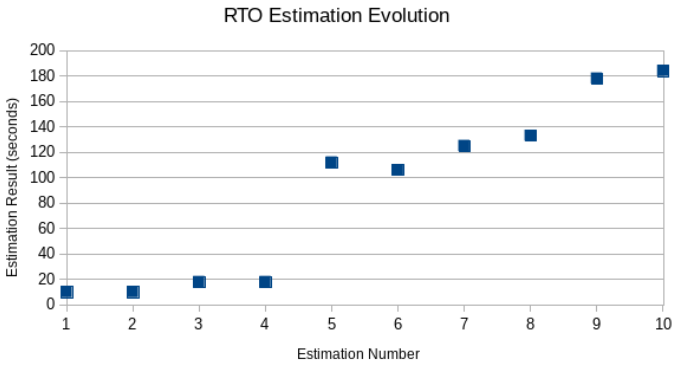


Figure 6.2: Test Results for dynamic RTO - RTO Estimation Evolution

By analysing the results, it can be seen that an improvement was obtained in the number of retransmissions performed until the ACK reception.

As for the RTO estimation evolution, the results obtained show that the RTO estimations did not converge to a value. On the other hand, they increased until the end of the experiment.

6.2.4 Dynamic RTO based on RTT, with improvement for the number of retransmissions

This simulation was similar to the previous one. However, an optimization was applied in order to increase the percentage of the 0 retransmissions case.

In fact, there is a maximum theoretical percentage that can be achieved for the 0 retransmissions. The maximum efficiency can be achieved in the following scenario: the sender window is full of bundles waiting for the arrival of the corresponding ACKs. If only one of those bundles is retransmitted and the response from the OBU brings ACKs for all of them, the number of retransmissions is the minimum possible. Thus, the percentage of the 0 and 1 retransmissions would be:

$$\#0 = \frac{WINDOW\_SIZE - 1}{WINDOW\_SIZE} \times 100$$

$$\#1 = \frac{1}{WINDOW\_SIZE} \times 100$$

The change applied in order to approximate the results to these theoretical percentages was to establish a shorter RTO for the first element of the window. By doing so, it is expected that the first window element is retransmitted before the other ones, so that its retransmission can bring the ACKs for all of the bundles in the window.

The results obtained were as follows:

Number of Retransmissions	Percentage (%)
0	50
1	35,7
2	10,7
3	3,6

Table 6.3: Test Results of dynamic RTO - Number of bundles retransmissions

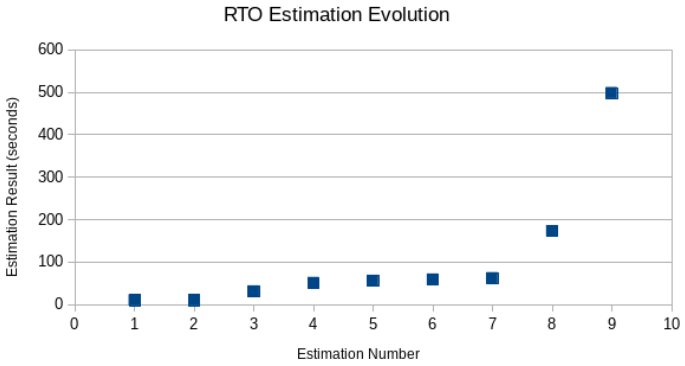


Figure 6.3: Test Results of dynamic RTO - RTO Estimation Evolution

As highlighted, the change applied on the RTO of the first window element has allowed the achievement of even better results. However, they are lower than the theoretical ones. Recalling that the number of bits used for the sequence numbers was 5, this results in a window size equal to 16. Thus, the theoretical percentage for the number of 0 and 1 retransmissions is $\frac{15}{16} \times 100 = 93,75\%$ and $\frac{1}{16} \times 100 = 6,25\%$, respectively.

The difference between the results obtained and the theoretical percentages can be explained by the fact that the situation where only the first window element was retransmitted rarely occurred. In most of the cases, what happened was that when a contact was established, all the RTOs had already expired, which caused the retransmission of all of them and not only the first one. The retransmission of only the first window element is rare and is more likely to occur during long contacts.

As for the RTO estimation evolution, the results were similar to the ones of the previous experiment. The evolution of the estimated values was not stable. In contrast, the computed RTO values increased from estimation to estimation.

6.2.5 RTO based on the number of established contacts

The previous approaches to estimate the RTO proved to be inefficient. The estimated RTO values did not stabilize and kept increasing from estimation to estimation. This affects the system performance because, in the worst case, there are contacts that are completely wasted because no RTO has expired.

For this reason, an improvement was performed on the decision of the best time to retransmit. This approach, addressed in the subsection 4.5.3, is based on the number of established contacts.

The following table outlines the results obtained for the number of retransmissions:

Number of Retransmissions	Percentage (%)
0	70
1	27,7
2	2
3	0,3

Table 6.4: Test Results of RTO based on the number of contacts- Number of bundles retransmissions

This alternative is clearly the one with the best results. Besides good results regarding the number of retransmissions per bundle, it also has a higher efficiency since no contacts are wasted due to unsuitable RTOs.

6.3 Test Results with the 24 de Agosto DCU

The implementation which achieved the best results (RTO based on the number of contacts) in the controlled environment was also tested in Porto, with the 24 de Agosto DCU.

6.3.1 Experiment Setup

The following map shows the area where the experiments were performed. The line in blue indicates the path followed by the car which was carrying the OBU. The circle in red signals the location where the DCU was placed.

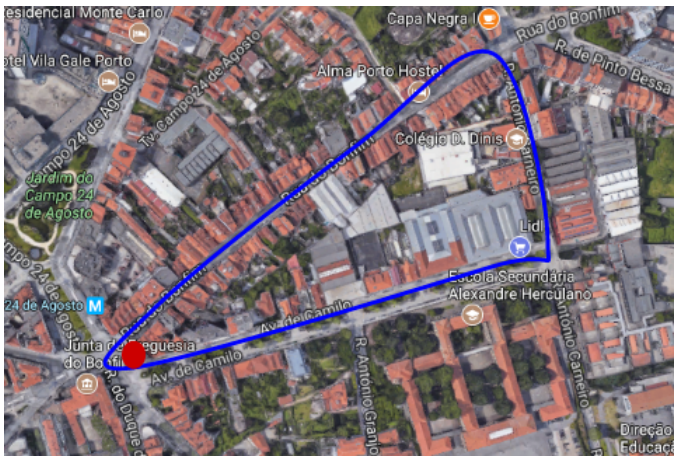


Figure 6.4: Location of the last experiment

The experiment had a duration of approximately 30 minutes.

As for the system settings, a 9-bit field was chosen for representing the sequence numbers. This value was previously determined in the subsection for the DCU placed at Praça da Galiza. This DCU was used as reference as it is the DCU with more recorded contacts.

6.3.2 RTO based on the Number of Contacts

The approach presented in the subsection 4.5.3 was also tested in the DCU of 24 de Agosto.

Number of Retransmissions	Percentage (%)
0	48,4
1	39,5
2	10,8
3	1,3

Table 6.5: Test Results of RTO based on the number of contacts- Number of bundles retransmissions

The system efficiency was not so high as in the controlled environment. Nevertheless, the number of retransmissions equal to 0 was still the case with the highest percentage.

6.4 Results Summary

The following charts sum up the results obtained for the number of retransmissions obtained in the controlled environment for each RTO estimate approach.

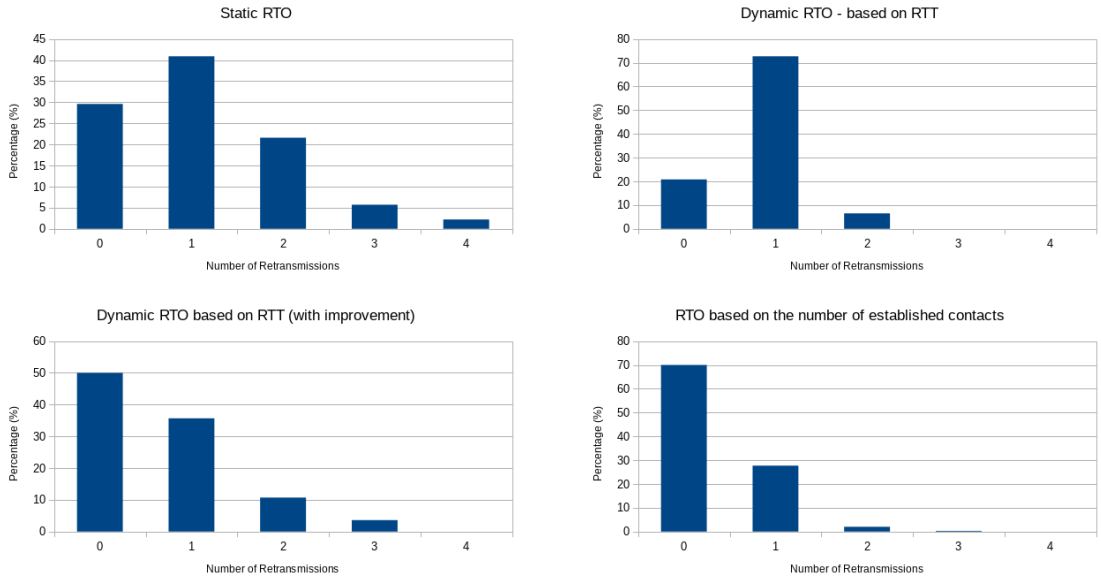


Figure 6.5: Results of the Experiments performed in controlled environment - Number of Retrasmissions

As it can be seen, the last RTO estimate was the one which has reached the results closer to the theoretical percentages presented previously in the subsection 6.2.4. With this approach, it was possible to increase the percentage of 0 retransmission in 40%, approximately. It is worth mentioning that two improvements allowed this result to be achieved. Firstly, the RTO of the first window element was anticipated, which sometimes avoided the retransmission of the remaining

bundles of the window. Secondly, instead of being based in the RTTs, the choice of the RTO was based in the number of the established contacts.

There is still another RTO estimate approach, discussed in the subsection 4.5.4, that could be implemented and tested. It is expected that this alternative will increase the efficiency of the previous approaches when DTN is used. However, the available testing conditions do not allow any conclusion to be obtained. If for some reason the OBU could not offload a bundle, it would keep it on its cache for 24 hours, which would not be feasible for the type of experiments performed.

Chapter 7

Conclusions

The results presented in the previous chapter will now be discussed. Later on, some improvements that could be applied to the proposed system will be addressed as future work.

7.1 Discussion of Results

From the results obtained in the previous chapter it can be concluded that the approaches of the static RTO and of the dynamic RTO based on the RTTs do not achieve the best performance.

The static RTO does not take the system conditions into consideration. A wrong initial prediction of the RTO will permanently affect the efficiency of the system. Even a first good estimation could become unsuitable as the system evolves.

The estimation of the RTO based on the previous RTT is also not the best option. The RTT should not be taken as a reference for the RTO because it includes not only the effective time needed to carry the bundle and the ACK, but also the time the bundle was kept in cache by the delay tolerant network. Other disadvantage that was highlighted during the experiments was that because this RTO estimation increased from estimate to estimate, there were some contacts that were totally wasted.

The last alternative, the one which predicts the RTO based on the number of established contacts, achieved the best results regarding the number of retransmissions and the contacts utilization. However, this approach, if tested using DTN, might have a lower performance. This RTO estimation is based on the assumption that the ACK of a given bundle will be available in the contact which follows the one when the bundle was transmitted. This approach has great results if the OBU can offload the bundles in a short range of time. On the other hand, when DTN is used, this assumption will fail because the bundles can be stored in intermediate nodes by long periods of time.

Additionally, it was found a maximum theoretical percentage for the number of retransmissions equal to 0. It is obtained by:

$$\#0 = \frac{1 - WINDOW\_SIZE}{WINDOW\_SIZE} \times 100$$

This situation happens when the RTO of one of the window elements expire before the others and that retransmission gets, as response from the OBU, the list of ACKs for all of the window elements. When the RTO is set based on the number of contacts, it is easier to make this scenario to happen frequently. On the other hand, when the RTO is set according to the RTTs, it is not so likely that this situation will happen. Most probably, when a contact is established either all of the RTOs will have already expired or none will.

As for the delivery ratio, the *selective repeat* protocol itself does not allow the loss of the bundles. Also, in the experiments performed, the delivery ratio was not an issue to worry about because the tests were of short duration. However, it is important to recall that the DCUs are resource constrained devices, namely in storage capabilities. Therefore, if the messages generation rate is higher than the messages upload rate, the situation where the DCUs have to discard bundles will eventually happen.

Nevertheless, the bundles will only be discarded for limitations in the DCUs storage capabilities. This is the only possible way the bundles will be lost, which represents a great improvement from the first system implementation that used CoAP.

7.2 Future Work

Throughout this dissertation, there are some implementation details that can be improved and may be pointed out as future work.

Firstly, some intelligence could be added on the receiver side. Rather than delivering the lists of ACKs to any DCU, it would be preferable to make that decision based on some relevant OBU current status. For example, when the OBU asks to the receiver for the lists of ACKs, it could inform it about its current position and even about its storage status. Therefore, the receiver could decide to respond with the lists of ACKs only for the DCUs that were inside a region, larger or smaller depending on the storage status of the DCU. Alternatively, that intelligence could also be added on the OBU side. As mentioned, the OBUs, when asking to the receiver for the lists of ACKs, can specify which DCU they are interested in. Thus, the OBU could learn about the DCUs to which it usually connects, for example. This way, the OBUs could avoid making requests for the DCUs they never found on their route.

Another implementation that could be improved is related to the periodic thread running on the OBU. It is responsible for communicating with the receiver in order to get the lists of ACKs. However, the request is done without taking the current communication technology available into consideration, whether it is cellular or not. This might have an higher cost to the network manager. For this reason, the thread implementation could be changed so that this request is only done when the OBUs are in the range of an RSU.

Appendix A

System Characterization

A.1 First Hop Analysis

A.1.1 Maximum Throughput per DCU

The following table presents the maximum throughput (number of acknowledges received per second) recorded in each DCU. These values were useful to determine the N-bit field used for representing the sequence numbers of the *selective repeat protocol*.

DCU	Throughput
Damião Góis Id	0,88
Bolhão	1,60
24 de Agosto	3,70
Combatentes	2,15
Campo Alegre	1,04
Casa da Música	1,94
Praça da Galiza	2,11
Praça da Liberdade	2,43
FEUP	2,5

Table A.1: Maximum Throughput per DCU (bundles/second)

A.1.2 Maximum Average Contact Duration per DCU

In contrast to the throughput, the values chosen for the duration were not the maximum contact duration ever registered in each DCU. Alternatively, it was selected the maximum average contact duration from the values presented in 3.3.3. This decision was adopted because the duration values are highly variable. For example, in the DCU placed at Praça da Galiza the maximum contact duration is 5685 seconds, which has only occurred once during the mentioned period.

The following table lists those values:

DCU	Duration
Damião Góis Id	63
Bolhão	36
24 de Agosto	32
Combatentes	178
Campo Alegre	158
Casa da Música	260
Praça da Galiza	92
Praça da Liberdade	62
FEUP	340

Table A.2: Maximum Average Contact Duration per DCU (seconds)

References

[1] Yingtian Du, Lin Zhang, Yufei Feng, Zhanyang Ren, and Zi Wang. Performance analysis and enhancement of IEEE 802.11 p/1609 protocol family in vehicular environments. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1085–1090. IEEE, 2010. URL: http://ieeexplore.ieee.org/abstract/document/5625013/.

[2] Georgios Karagiannis, Onur Altintas, Eylem Ekici, Geert Heijenk, Boangoat Jarupan, Kenneth Lin, and Timothy Weil. Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions. *IEEE Communications Surveys & Tutorials*, 13(4):584–616, 2011. URL: http://ieeexplore.ieee.org/document/5948952/, doi:10.1109/SURV.2011.061411.00019.

[3] Yunior Luis, Pedro M. Santos, Tiago Lourenco, Carlos Pérez-Penichet, Tânia Calcada, and Ana Aguiar. UrbanSense: An urban-scale sensing platform for the internet of things. In *Smart Cities Conference (ISC2), 2016 IEEE International*, pages 1–6. IEEE. URL: http://ieeexplore.ieee.org/abstract/document/7580869/.

[4] Wai Chen, Ratul K. Guha, Taek Jin Kwon, John Lee, and Yuan-Ying Hsu. A survey and challenges in routing and data dissemination in vehicular ad hoc networks. *Wireless Communications and Mobile Computing*, 11(7):787–795, July 2011. URL: http://doi.wiley.com/10.1002/wcm.862, doi:10.1002/wcm.862.

[5] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, 8(2/3):153–167, 2002. URL: http://dl.acm.org/citation.cfm?id=506905.

[6] Tatsuaki Osafune, Lan Lin, and Massimiliano Lenardi. Multi-hop vehicular broadcast (mhvb). In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pages 757–760. IEEE, 2006. URL: http://ieeexplore.ieee.org/document/4068699/.

[7] M. N. Mariyasagayam, T. Osafune, and M. Lenardi. Enhanced multi-hop vehicular broadcast (MHVB) for active safety applications. In *Telecommunications, 2007. ITST'07. 7th International Conference on ITS*, pages 1–6. IEEE, 2007. URL: http://ieeexplore.ieee.org/abstract/document/4295866/.

[8] Christian Schwingenschlogl and Timo Kosch. Geocast enhancements of AODV for vehicular networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(3):96–97, 2002. URL: http://dl.acm.org/citation.cfm?id=581307.

[9] Elena Fasolo, Andrea Zanella, and Michele Zorzi. An effective broadcast scheme for alert message propagation in vehicular ad hoc networks. In *Communications, 2006. ICC'06. IEEE*

International Conference on, volume 9, pages 3960–3965. IEEE, 2006. URL: http://ieeexplore.ieee.org/abstract/document/4025102/.

[10] Jaehoon (Paul) Jeong, Tian He, and David H.C. Du. TMA: Trajectory-based Multi-Anycast forwarding for efficient multicast data delivery in vehicular networks. *Computer Networks*, 57(13):2549–2563, September 2013. URL: http://linkinghub.elsevier.com/retrieve/pii/S1389128613001424, doi:10.1016/j.comnet.2013.05.002.

[11] Mohammad M. Qabajeh, Aisha H. Abdalla, Othman O. Khalifa, and Liana K. Qabajeh. Position-based multicast routing in Mobile Ad Hoc networks. In *Computer and Communication Engineering (ICCCE), 2012 International Conference on*, pages 104–108. IEEE, 2012. URL: http://ieeexplore.ieee.org/abstract/document/6271161/.

[12] Matthias Transier, Holger Fü\s sler, Jörg Widmer, Martin Mauve, and Wolfgang Effelsberg. Scalable position-based multicast for mobile ad-hoc networks. In *BroadWim*, 2004. URL: https://infoscience.epfl.ch/record/30084.

[13] R. Uzcategui and G. Acosta-Marum. Wave: A tutorial. Communications Magazine, IEEE, vol. 47, no. 5, pp. 126–133. URL: http://ieeexplore.ieee.org/document/4939288/.

[14] T. Kwon and Y. Choi N. Choi S. Choi, Y. Seok. A solicitation-based ieee 802.11p mac protocol for roadside to vehicular networks. *Mobile Networking for Vehicular Environments, 2007, pp. 91–96*, 2016. URL: http://ieeexplore.ieee.org/document/4300811/.

[15] D. Jiang and L. Delgrossi . Ieee 802.11p: Towards an international standard for wireless access in vehicular environments. *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE, 2008, pp. 2036–2040*, 2008. URL: http://ieeexplore.ieee.org/document/4526014/.

[16] R. Cooper D. Stancil and F. Bai L. Cheng , B. Henty. A Measurement Study of Time-Scaled 802.11a Waveforms Over The Mobile-to-Mobile Vehicular Channel at 5.9 GHz. *Com- munications Magazine, IEEE, vol. 46, no. 5, pp. 84–91*, 2008. URL: http://ieeexplore.ieee.org/document/4511654/.

[17] Andreas Festag. Standards for vehicular communication—from ieee 802.11 p to 5g. *e & i Elektrotechnik und Informationstechnik*, 132(7):409–416, 2015. URL: http://link.springer.com/10.1007/s00502-015-0343-0.

[18] Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott, and Howard Weiss. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine*, 41(6):128–136, 2003. URL: http://ieeexplore.ieee.org/abstract/document/1204759/.

[19] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34. ACM. URL: http://dl.acm.org/citation.cfm?id=863960.

[20] Kevin Fall and Stephen Farrell. Dtn: an architectural retrospective. *IEEE Journal on Selected areas in communications*, 26(5), 2008. URL: http://ieeexplore.ieee.org/document/4530739/.

[21] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. *SIGCOMM Comput. Commun. Rev.*, 34(4):145–158, August 2004. URL: http://doi.acm.org/10.1145/1030194.1015484, doi:10.1145/1030194.1015484.

[22] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. Controlling the mobility of multiple data transport ferries in a delay-tolerant network. In *INFOCOM 2005. 24th annual joint conference of the IEEE computer and communications societies. Proceedings IEEE*, volume 2, pages 1407–1418. IEEE. URL: http://ieeexplore.ieee.org/abstract/document/1498365/.

[23] Elizabeth M. Daly and Mads Haahr. Social network analysis for routing in disconnected delay-tolerant manets. In *Proceedings of the 8th ACM international symposium on Mobile ad hoc networking and computing*, pages 32–40. ACM. URL: http://dl.acm.org/citation.cfm?id=1288113.

[24] Kevin Fall, Wei Hong, and Samuel Madden. Custody transfer for reliable delivery in delay tolerant networks. *IRB-TR-03-030, July*, 2003. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.59.1856&rep=rep1&type=pdf.

[25] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, November 1984. URL: http://doi.acm.org/10.1145/357401.357402, doi:10.1145/357401.357402.

[26] Tim Moors. A critical review of" end-to-end arguments in system design". In *Communications, 2002. ICC 2002. IEEE International Conference on*, volume 2, pages 1214–1219. IEEE. URL: http://ieeexplore.ieee.org/abstract/document/997043/.

[27] Rahul C Shah, Sumit Roy, Sushant Jain, and Waylon Brunette. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2):215–233, 2003. URL: http://www.sciencedirect.com/science/article/pii/S1570870503000039.

[28] Shlomi Dolev, Michael Kate, and Jennifer L Welch. A competitive analysis for retransmission timeout. In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pages 450–455. IEEE, 1995. URL: http://faculty.cs.tamu.edu/welch/papers/ntwks99.pdf.

[29] Hannes Ekstrom and Reiner Ludwig. The peak-hopper: A new end-to-end retransmission timer for reliable unicast transport. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2502–2513. IEEE, 2004. URL: http://ieeexplore.ieee.org/abstract/document/1354671/.

[30] Alex Kesselman and Yishay Mansour. Optimizing TCP retransmission timeout. In *International Conference on Networking*, pages 133–140. Springer. URL: http://link.springer.com/chapter/10.1007/978-3-540-31957-3_17.

[31] Ruhai Wang, Mingjian Qiu, Kanglian Zhao, and Yi Qian. Optimal rto timer for best transmission efficiency of dtn protocol in deep-space vehicle communications. *IEEE Transactions on Vehicular Technology*, 66(3):2536–2550, 2017. URL: http://ieeexplore.ieee.org/document/7478130/.

[32] Angelo P. Castellani, Mattia Gheda, Nicola Bui, Michele Rossi, and Michele Zorzi. Web Services for the Internet of Things through CoAP and EXI. In *Communications Workshops (ICC), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011. URL: http://ieeexplore.ieee.org/abstract/document/5963563/.

[33] Carsten Bormann, Angelo P. Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012. URL: http://ieeexplore.ieee.org/abstract/document/6159216/.

[34] Matthias Kovatsch, Simon Duquennoy, and Adam Dunkels. A Low-Power CoAP for Contiki. pages 855–860. IEEE, October 2011. URL: http://ieeexplore.ieee.org/document/6076698/, doi:10.1109/MASS.2011.100.

[35] Shahid Raza, Hossein Shafagh, Kasun Hewage, Rene Hummen, and Thiemo Voigt. Lithe: Lightweight Secure CoAP for the Internet of Things. *IEEE Sensors Journal*, 13(10):3711–3720, October 2013. URL: http://ieeexplore.ieee.org/document/6576185/, doi:10.1109/JSEN.2013.2277656.

[36] Matthias Kovatsch, Martin Lanter, and Zach Shelby. Californium: Scalable cloud services for the internet of things with coap. In *Internet of Things (IOT), 2014 International Conference on the*, pages 1–6. IEEE, 2014. URL: http://ieeexplore.ieee.org/abstract/document/7030106/.

[37] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE, 2008. URL: http://ieeexplore.ieee.org/abstract/document/4554519/.

[38] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks (mqtt-sn) protocol specification. *International business machines (IBM) Corporation version*, 1, 2013. URL: http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf.

[39] Meena Singh, M.A. Rajan, V.L. Shivraj, and P. Balamuralidhar. Secure MQTT for Internet of Things (IoT). pages 746–751. IEEE, April 2015. URL: http://ieeexplore.ieee.org/document/7280018/, doi:10.1109/CSNT.2015.16.

[40] Shinho Lee, Hyeonwoo Kim, Dong-kweon Hong, and Hongtaek Ju. Correlation analysis of MQTT loss and delay according to QoS level. In *Information Networking (ICOIN), 2013 International Conference on*, pages 714–717. IEEE, 2013. URL: http://ieeexplore.ieee.org/abstract/document/6496715/.

[41] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan. Performance evaluation of MQTT and CoAP via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE, 2014. URL: http://ieeexplore.ieee.org/abstract/document/6827678/.

[42] Andrew S. Tanenbaum and D. Wetherall. *Computer networks*. Pearson Prentice Hall, Boston, 5th ed edition, 2011. OCLC: ocn660087726.