UNIVERSIDADE D
COIMBRA

Tiago Miguel Vitorino Simões Gomes

DELAY TOLERANT NETWORK ROUTING

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Carlos M. Fonseca and Prof. José Luis Santos and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering

October 2020

Faculty of Sciences and Technology

Department of Informatics Engineering

Delay Tolerant Network Routing

Tiago Miguel Vitorino Simões Gomes

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Carlos M. Fonseca and Prof. José Luis Santos and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

October 2020

UNIVERSIDADE D
COIMBRA

The work presented in this dissertation was partially carried out in the scope of the MobiWise project: From mobile sensing to mobility advising (P2020 SAICTPAC/0011/2015), co-financed by COMPETE 2020, Portugal 2020 - Operational Program for Competitiveness and Internationalization (POCI), European Union's ERDF (European Regional Development Fund), and the Portuguese Foundation for Science and Technology (FCT).

Cofinanciado por:

Acknowledgements

Firstly, I would like to thank my advisors Prof. Carlos M. Fonseca and Prof. José Luis Santos for inviting me to be a part of this project and for their guidance throughout this last year. This work has helped me expand my knowledge in fields I had not anticipated and for that I am thankful. I would also like to particularly thank them for their continued patience and understanding, even when I could not manage to meet deadlines.

A warm thank you to all colleagues who have been a part of my 5 years in this department, be it by collaborating in projects, sharing ideas, studying together or, most importantly, sharing fun moments in our free times. Your camaraderie made these years easier and certainly more enjoyable. Particularly, I would like to thank my close friends Artur, Carolina, Guilherme, Leonardo and Pedro for their support and honesty. I know that I can always count on you and that we will stick together. A heartfelt thanks to Mariana for always being able to cheer me up, even in this tough period. You make me a better and happier person.

Lastly, but just as importantly, thank you to my parents and brother for being the best support network I could hope for. You made sure I managed to stay focused on this task, by helping me in every single way you could; for that I am eternally grateful. I owe my successes to you. Especially, I would like to thank my parents for letting me play with a computer from a very early age, and my brother for teaching me how to love it. Ultimately, you are to blame for all the words that follow!

Abstract

In the context of computer networks, node mobility can pose challenges to data delivery. Particularly, it increases the likelihood of disruptions to the connections, making it unlikely for an end-to-end path from the source to the destination to ever exist. Given that the architecture and protocols of today's Internet depend on such paths, they are likely to perform poorly under such conditions.

Delay-Tolerant Networks handle these challenging conditions by assuming the existence of disruptions, which can be compensated by having some storage in each node. This allows nodes to store data as soon as it is received, carrying it with them as they move, and being able to later forward it to another node they come into contact with. Therefore, in this store-carry-forward paradigm, mobility is seen as something to be exploited for the benefit of the network. Each node forwards and manages its data according to a certain routing strategy, which has a major impact in the network's performance.

In this dissertation, a simplified model of a urban sensing scenario is proposed, where Data Collecting Units gather sensor data which is to be delivered to Road Side Units by entities that move between them. A finite-state discrete-time homogeneous Markov chain is used to describe this Delay-Tolerant Network. In that context, mobility, communication and routing are modelled separately, permitting the usage of different implementations of each. We also introduce an analysis framework for this model, which describes how to calculate multiple network performance metrics. With this, a specific real-world scenario can be analysed, enabling a certain type of routing strategy to be optimized.

Keywords

Delay-Tolerant Network, Store-Carry-Forward, Mobility Model, Communication Model, Routing Strategy, Urban Sensing.

Resumo

No contexto de redes de computadores, a mobilidade dos nós pode causar desafios à entrega de dados. Particularmente, isso aumenta a probabilidade de quebras nas conexões, tornando improvável a ocorrência de caminhos de ponta a ponta desde a origem até ao destino. Sendo que as arquiteturas e protocolos usados pela Internet de hoje em dia dependem da existência de tais caminhos, estes terão provavelmente mau desempenho sob estas condições.

As Redes Tolerantes a Atraso lidam com estas condições difíceis assumindo a existência de quebras, que podem ser compensadas pela existência de armazenamento em cada nó. Isto permite aos nós armazenar dados assim que são recebidos, transportá-los enquanto se movem, e mais tarde encaminhá-los para outro nó com que entre em contacto. Assim, neste paradigma de Armazenamento-Transporte-Encaminhamento, a mobilidade é vista como algo que pode ser explorado para o benefício da rede. Cada nó encaminha e gere os seus dados de acordo com uma certa estratégia de encaminhamento, que tem um impacto significativo no desempenho da rede.

Nesta dissertação, propomos um modelo simplificado de um cenário de sensores em ambiente urbano, em que Unidades de Recolha de Dados reúnem dados de sensores que têm que ser entregues a Unidades de Beira de Estrada, por entidades que se movem entre elas. Para descrever esta Rede Tolerante a Atraso, usamos uma Cadeia de Markov homogénea, de tempo discreto, e de estados finitos. Neste contexto, a mobilidade, comunicação e encaminhamento são modelados separadamente, permitindo o uso de diferentes implementações para cada um deles. Também introduzimos um quadro de análise para este modelo, que descreve como calcular múltiplas métricas de desempenho da rede. Com isto, analisamos um cenário real específico, permitindo-nos otimizar a estratégia de encaminhamento.

Palavras-Chave

Redes Tolerantes a Atraso, Armazenamento-Transporte-Encaminhamento, Modelo de Mobilidade, Modelo de Comunicação, Estratégia de Encaminhamento, Sensores em Ambiente Urbano.

Contents

Acronyms

CA Cellular Automaton. 6

DCU Data Collecting Unit. 1, 2, 15–18, 21–29, 34, 37, 38, 41–43, 48

DTN Delay-Tolerant Network. 1–8, 15, 47

MANET Mobile Ad Hoc Network. 3, 5, 7

OBU On Board Unit. 1, 2, 15

RSU Road Side Unit. 1, 2, 15–18, 20, 23–26, 34, 37, 41, 48

SCF Store-Carry-Forward. 1, 5, 15, 47

SF Store-and-Forward. 1, 5

List of Figures

List of Tables

Notation

Representation	Meaning
a	variable
A	random variable
A	generic set
$\mathcal{A}$	set of states
$\mathbf{a}$	row vector
a_i	ith element of row vector $\mathbf{a}$
$\mathbf{A}$	matrix
$\mathbf{a}_i$	ith row of matrix $\mathbf{A}$
a_{ij}	element on the ith row and jth column of matrix $\mathbf{A}$

Table 1: General notation used for all equations in this dissertation.

Representation	Meaning
$\mathcal{X}$	State space
s	Number of states in $\mathcal{X}$
E	Set of entities in the model
m	Number of entities in E
n	Number of locations
t	Data's time-to-live
c	Contact range
τ	Sampling period
r	Replacing ratio
l_i^x	Location of entity $i \in$ E on state $x \in \mathcal{X}$
d_i^x	Data of entity $i \in$ E on state $x \in \mathcal{X}$
$\mathbf{P}$	Transition probability matrix
$\boldsymbol{\pi}$	Stationary state probability vector

Table 2: Notation for parameters and concepts used on the models defined in Chapter 4.

Chapter 1

Introduction

Delay-Tolerant Networks (DTNs) [8] arise in situations where nodes in a network are not permanently connected, but may connect to one another at discrete time intervals in predictable or unpredictable ways. Under these circumstances, an end-to-end path between a source of data and its intended destination is not guaranteed to occur at any interval, and data transfer may have excessive delays or simply never happen. These characteristics contradict some of the assumptions of the current Internet architecture and protocols [12], and thus a different architecture is required. DTN architectures typically feature nodes with persistent storage (buffer), allowing them to save the data that they receive, and to forward it to other nodes when a connection is made. This approach is known as Store-and-Forward (SF) operation and it allows the network to accommodate delays and disruptions.

The interruption of connectivity between nodes can be a consequence of many different scenarios (such as limited energy resources or noisy connections), but a particularly interesting one occurs when the network is composed of sparse mobile nodes. In this scenario, disruption of connections takes place when nodes move in such a way that the receiving node is no longer within a certain contact range of the transferring node. Whenever a mobile node with data is within contact range of another, it must decide whether to forward the data to the other or not, and also whether to keep carrying it, making this a Store-Carry-Forward (SCF) operation. The policies behind these decisions are the basis of the network's routing protocol, which may or may not take into account prior knowledge regarding the mobility of its nodes (mobility model), or some sort of network topology information; the more information this protocol has access to, the more likely it is for data to reach its intended destination sometime in the future.

A possible real-world application of a scenario akin to the one described above is considered in the MobiWise project, where sensors located at certain points in a city collect various types of environmental data (such as temperature, humidity and pollution), which ultimately needs to be delivered to some device through the Internet. Each set of sensors is directly connected to a Data Collecting Unit (DCU), which can store the data being generated but has no direct connection to the Internet. In order for data to be delivered, it needs to eventually be transmitted to Road Side Units (RSUs), which are located in other points in that city and have a direct connection to the Internet; these are essentially data sinks which function as the end node in this scenario. Both DCUs and RSUs are capable of communicating remotely with nearby devices. However, they are never sufficiently close to each other to allow direct communication between them. Mobile nodes (e.g., cars, bicycles, buses, drones or others) in the area have an On Board Unit (OBU) which allows them to

receive, carry and transmit such data. They are, therefore, able to receive data from the DCU and eventually forward it either to other mobile nodes or directly to an RSU. When within contact range of other units, OBUs are faced with the challenge of deciding which units they should communicate with and what data should be transmitted between them. These routing decisions have consequences on the performance of the network, which may be reflected on network metrics such as the likelihood of delivering data or the delivery latency. Anticipating how changes in routing strategies influence the network's performance is, therefore, one of the main interests of this dissertation.

While most of the literature that investigates similar applications focuses on analysing the results of simulations of certain scenarios [3], we are mostly interested in exploring the possibility of building an analytical model which would allow us to calculate the expected network metrics under any condition. In order to have tractable calculations, it is necessary to model a simplified version of a real-world scenario which is still detailed enough to replicate real behaviours. We aim to achieve this by separating the scenario into smaller components that are simpler to model. Having defined this model, we can then use it to produce a simplified model of a real-world scenario and study how it behaves under certain scenarios and how different parameters change the network's performance.

The remainder of this document is laid out as follows: In Chapter 2 we provide an overview of mobility in the context of data delivery, introducing DTNs as a specific approach that can be described in three different layers. Chapter 3 introduces some mathematical concepts that are needed for later chapters. Then, in Chapter 4, we propose and fully describe a simplified model for a particular urban sensing scenario, and we also study how to analyse and implement such a model. In Chapter 5, we employ the proposed model to analyse a specific real-world scenario and describe its optimization considerations. Finally, in Chapter 6 we present our main conclusions and put forward some possible approaches for future work.

Chapter 2

Background

In the previous chapter we introduced the topics which will be the focus of this dissertation, as well as a possible application of interest. In this chapter, we will introduce the necessary background which will help contextualizing the issue at hand, as well as further defining and detailing some concepts that were used. Moreover, we will reference works in related areas which helped guiding the development of our research. Namely, on Section 2.1 we will review the usage of mobility for data delivery in general. Then, on Section 2.2 we will present DTNs as a specific instance of this approach, exploring aspects of its mobility (2.2.1), communication (2.2.2) and routing (2.2.3).

2.1 Mobility as a Means of Carrying Data

As previously mentioned (Chapter 1), the conventional Internet architecture and protocols are not well suited to deal with the challenges resulting from the mobility of nodes in a network. Mobile Ad Hoc Networks (MANETs), which were among the earliest approaches to dealing with networks of this type, assume the existence of mobile nodes with wireless networking capabilities which allows them to communicate with one another without the need for a supporting infrastructure, while being able to self-configure and self-organize [59]. While MANET concepts have been successfully used in many real-world applications (such as smart home devices [2, 21, 68] and military networks [42, 67]), the traditional MANET Routing Protocols (namely link-state [34] or distance-vector routing [20]) assume that we can, at some point in time, establish end-to-end paths through which packets are sent [44, 58]; this is unlikely to occur in sparsely connected mobile networks and, consequently, node mobility is usually seen as having undesirable effects on network performance [69].

While mobility in those architectures is mostly seen as an obstacle, the literature also describes a wide variety of other approaches in which mobile entities (e.g., vehicles [40], humans [6], animals [26, 50]) are seen as desirable elements that carry data either as a replacement for conventional communication channels or as an extension to them. This method of physically transferring information through the movement of entities constitutes an alternative communication channel (sometimes informally referred to as Sneakernet [30]) which can be described in ways that are analogous to normal communication channels; therefore, network metrics such as delay, delivery ratio and throughput still apply. Channels of this kind can achieve higher throughput than conventional channels (since petabytes of data can be carried at once, e.g. large-scale data transfer to cloud services [9, 35, 45]), but usually at the cost of higher latency or a lower delivery ratio due to the nature of their entities. As a result, they are better suited to use-cases that are not time-sensitive, being

particularly unsuited to real-time communications. Besides this, since they do not use the Internet infrastructure, these channels can be employed in places where there is a lack of coverage (such as developing countries [40, 46]), while also allowing lower implementation costs and avoiding network capacity limitations.

In a 2018 survey, Baron et al. [3] propose a classification system for different methods of data transfer with mobile entities, which takes into account the characteristics of the entities being used and also how the data is delivered; we believe that it provides helpful abstractions which we will be using in this section. Mobile entities can be categorized as Instrumentalized, Controlled or Contracted. The first category, commonly known as data mules [47], describes a scenario in which we exploit the pre-existent movement of an entity (e.g., an animal [73] or public transportation [4]), which can either be random, predictable in some way, or follow a certain schedule. The second category, also known as data ferries [71], refers to entities which we deliberately move through a trajectory that suits our data delivery purposes (e.g., unmanned aerial [19] or underwater vehicles [1, 38]). The final category refers to delivery services which we do not control directly, such as airplanes or delivery trucks. Data mules are generally the preferred type of mobile entity, due to the lower associated implementation costs, but they are not always feasible in practice, since a scenario might not have exploitable mobility or there might be a need for increased reliability.

Data delivery may be direct, if it occurs through entities that carry the data directly from a source to a destination, or indirect, if it occurs as a result of data being forwarded through multiple entities until it reaches a destination. For the purpose of this dissertation, direct data deliveries are not particularly interesting, since it is assumed that they are unlikely for the application being studied, and we are mainly concerned with understanding entity interactions; therefore strategies for direct data delivery will not be explored. Regarding indirect data delivery strategies, entities can exchange data synchronously or asynchronously. Synchronous transmissions occur when entities exchange data directly with one another by being within contact range, and this may be possible at any location (in the case of "floating" compositions) or only at certain predefined locations (in the case of "pre-positioned" compositions). As for asynchronous transmissions, these occur when entities drop off data at a certain intermediate nodes, from which other entities can obtain data later, which may be stationary (throwboxes [72]) or mobile (e.g., data ferries).

In the application being studied, we assume the existence of multiple entities with pre-existing mobility that can be exploited for our goals. Given our lack of certainty about which specific entities may pass by our desired locations, their movement is assumed to be hard to predict and, consequently, will be described as random. Moreover, communication can occur synchronously at any location. Therefore, the method of data transfer in this application can be classified as an *indirect* data delivery with *synchronous* data passing which occurs in *floating* locations between entities with *random* mobility. This type of data transfer is particularly well suited to DTN approaches, largely due to its synchronous communication; therefore, in the next chapter we will explore these approaches with greater detail.

2.2 Delay Tolerant Networks

In Chapter 1, we have already introduced DTNs and some of the challenges related to them. However, in this section, we intend to further explore their specification, applications and modelling considerations.

While sometimes described as a type of MANET, literature for DTNs differs significantly in that disruptions to connections are assumed to be frequent and thus the transmission of data is done via successive hop-by-hop exchanges, instead of waiting for an unlikely end-to-end path to occur. This is the Store-and-Forward (SF) paradigm, in which nodes that receive data simply store it in a buffer until there are opportunities to forward it to some other node. These concepts make them successful at handling sparsely connected networks such as the one being studied, as well as many other applications, including interplanetary communication [36], Internet kiosks in areas without normal coverage [40, 46], sensor data collection [1, 38, 47] and animal tracking [26, 50, 73]. However, these networks imply a high likelihood of having large delivery delays and failed transmissions and thus are unsuited for critical or urgent communications.

In DTNs, there are many possible sources of disruptions, such as limited energy resources (nodes shutting down after running out of battery) or noisy connections, but a particularly interesting scenario occurs when the disruptions are a result of the mobility of nodes. In this case, a compelling instance of the SF paradigm is the Store-Carry-Forward (SCF) paradigm, in which nodes are able to physically transport data while it is in its stored phase [70]. This allows us to leverage the mobility of the nodes to improve the network's performance, making it particularly well suited for applications like the one being studied, which are sometimes termed Intermittently Connected Mobile Networks or Opportunistic Networks. In general, a SCF network can be broken down into three distinct components: 1) the mobility of its nodes, 2) how data is transferred between nodes and 3) how the nodes decide which nodes they send data to and which data they keep. By formalizing these components, we create a Mobility Model, a Communication Model and a Routing Strategy, respectively, which are enough to fully describe the SCF network. In the following sections, we explore these three layers in greater detail.

While much of the SCF literature focuses on simulations of specific scenarios [3], we are interested in having an approach that is more analytical. In that context, some papers have explored the possibility of describing SCF interactions through Markov Chain models [32, 48, 66], which we believe to be a promising approach.

2.2.1 Mobility Models

The performance of a SCF network is typically highly dependent on the mobility of its nodes and their resulting contacts [41]. Consequently, SCF scenarios are usually analysed under the assumption of a certain mobility model.

When studying a real-world scenario, it is desirable to have a mobility model with properties that mimic the entities' behaviour. Ideally, we would gather data from existing entities, which could then be used either to directly emulate their behaviour or to deduce a fitting model that could later be used for simulations (e.g., [26]). However, collecting mobility traces from these real-world entities might be difficult or expensive; in Vehicular DTN literature, a common alternative is to create mobility traces through the usage of traffic simulators (e.g., [29]), though these typically require knowledge about the road topology being considered.

For the application being studied, we have no information about the behaviour of real-world entities or even which road topology is adequate. Therefore, as a first approach, it would be highly desirable to build a simple mathematical model that does not depend on any of these assumptions and that can be validated and analyzed from a theoretical standpoint. It would also be important for us to be able to experiment how a network

performs under different entity behaviours, by simply changing the model's parameters. The literature shows a wide variety of mobility models employed in the study of these networks, with differing levels of realism (how accurately they represent a real scenario), diversification (how well they can be adapted to different mobile nodes and environments) and complexity (how computationally intensive they are) [49].

Although mobility models based on random behaviours suffer from poor realism, they are extremely prevalent (particularly the Random Walk [39] and Random Waypoint [24] models), due to their low complexity and easy diversification. This diversification capability is particularly interesting to us, given the aforementioned lack of knowledge about the scenario. The Random Walk Model is a simple mobility model in which every node in the network moves independently and decides, at each moment, a random direction to move towards and sometimes a random speed or distance, without any sort of memory of previous movements. Some examples of its usage are [47, 54, 55, 56].

Another common way of studying road traffic is through the usage of models based on Cellular Automata (CA). A Cellular Automaton [65] is a computational model which typically consists of a grid of cells, each with a binary state ("on" or "off") [11]. Each cell has a notion of its neighbourhood (that is, cells that are considered to be nearby) and decides its state in the next instant based on those neighbours, according to some rule; this "update" of a cell's status typically occurs synchronously for all cells. In particular, we are interested in stochastic cellular automata, which are CA in which the previously mentioned rule is stochastic. Moreover, in the context of CA for road traffic, it is very common to assume a one-dimensional grid; this will essentially represent a one-way street in which each cell corresponds to a position on the road, and a "on" state represents a car occupying that position. Consequently, such a simplification implies a one-way street where overtaking cars is impossible. Although these models have a relatively low complexity, they can still be realistic at describing macroscopic behaviours of traffic [33], and thus they may be useful for our application.

2.2.2 Communication Models

In general terms, when modelling the communication of a network of this type, we are mainly concerned with aspects related to node connectivity and data transfer. On DTNs, nodes communicate using some device with wireless technology (such as Bluetooth or Wi-Fi), which is typically the same for all nodes. The wireless device and protocols that are used dictate factors such as communication range and data transfer rate. Communication range (or contact range) is the maximum distance between two nodes that still allows them transfer data to one another, while the data transfer rate is the amount of data that can be transferred between these nodes per unit of time. Evidently, higher data transfer rates and communication ranges are beneficial to a network's performance, but an increase in these capabilities typically also implies an increase in power usage. Given that the nodes in this network have to be powered by some finite battery, this may be detrimental for the network's performance in the long-run [52].

In a real-world scenario, the performance of a wireless device may decrease due to factors such as physical obstacles and signal interference; however, when analysing communication, range and data rates may be simplified by assuming a fixed value based on average/typical values. It may also be assumed that the devices are powered by a battery that has a sufficiently large capacity to ensure that devices do not run out of battery under a typical usage. Moreover, we may further simplify the communication by assuming that the data transfer rate is large enough that a message (data packet) can be transferred practically

instantly to multiple nodes simultaneously. These simplifications are quite common in the DTN literature, as they allow researchers to reduce the number of variables in consideration and focus on optimizing a system under certain assumptions. However, they may prove to be an obstacle in achieving realistic results.

2.2.3 Routing Strategies

In DTNs, routing is essentially a sequence of independent decisions made in each node with the information that they have locally. Good routing decisions become more important the sparser the network is, but good performance can be achieved even without global knowledge [22]. A routing strategy can be seen as a general term for a set of 4 different policies, namely the forwarding policy, the replication policy, the dropping policy and the scheduling policy [23]. We will describe what these policies are separately and talk about routing strategies in general.

When a mobile node is carrying data and it comes across another mobile node, it has to decide whether to forward that data or not; this is the *forwarding policy* and it is the most commonly studied element of a routing strategy. This decision may be based on, for example, some information about the other node or the overall state of the network, and in general the more information a node has access to, the easier it is for it to make a "good" decision. Typically, a decision is considered to be "good" when data is passed to a node with a high likelihood of eventually delivering that data to the destination (or to another node that will deliver it); on the other hand, if data is delivered to nodes where that likelihood is low, the chances of it representing a waste of energy and/or storage resources increase, making it a "bad" decision.

When a node decides to forward its data to another, after that transmission is finished, the node may keep a copy of the data sent or simply drop it; this is the *replication policy*. If data is never replicated, then there is at most a single copy of each message in the entire network. This is usually referred to as forwarding-based routing, and it is typically less resource-intensive and simpler to implement, as handling replicas of data is not necessary. These protocols usually require information about the network topology in order to select a good path, which is then followed hop-by-hop, that is, an approach which is similar to traditional MANET protocols. While efficient, forwarding-based routing may not provide enough guarantees in terms of delivery ratio and delays [55], since messages can easily get lost when nodes disconnect or transmissions fail. Replication-based routing tries to solve these issues by allowing nodes to keep copies of data when it is deemed necessary, creating data redundancy in the network. While this may decrease the likelihood of data getting lost, when this replication is excessive it may have downsides. Specifically, we may be using more resources than necessary (that is, we could achieve the same performance with fewer copies) and we also may be increasing the buffer utilization in the network to the point where it would force some messages to be dropped (more on that later). Due to this, one of the main considerations of replication-based protocols is how to minimize replication without significantly affecting the delivery ratio or other network performance metrics.

In order for mobile nodes to carry data, they need to have some sort of storage mechanism, known as a buffer. While in theoretical models the buffer is frequently assumed to be infinitely large [53], it is clear that in a real world scenario there is always some storage limitation, which can significantly affect the performance of a network. Therefore, it is important to consider the situation in which a node may receive new data without having the necessary space to store it, which is particularly common when replication is possible. When this occurs, the node has to decide which data it must keep and which data it must

drop, that is, it must have a *dropping policy*. This policy is the centerpiece of an active topic of research in the context of DTNs called buffer management (some examples are [28, 43, 53]). Interestingly, while a larger buffer may provide better delivery guarantees, it may also cause bufferbloat [15]; this is a common network phenomenon where increasing the size of the buffers simply provides more latency for no benefit [13]. It follows that buffer management is crucial, as it cannot be avoided by simply increasing the available storage.

As previously mentioned in Section 2.2.2, the duration of contact between two nodes has direct consequences on the amount of data that can be transmitted in that period, particularly if the data transfer rates are not very high. Given that contact duration is typically short in this type of networks, it may be important to estimate how many messages we can exchange during the next contacts, so that we can choose to prioritize certain messages in advance. The *scheduling policy* deals with issues of this sort and we may, for example, prefer to prioritize messages that are shorter or that contain newer data. These decisions may have an impact in multiple performance metrics such as delivery ratio and latency [57].

With the challenges of implementing these policies in mind, it becomes clear that there are three main considerations when developing a routing strategy for DTNs: whether we have information about future contacts, whether mobility can be exploited and what is the resource availability [63, page 20]. While in our case we assume the possibility of exploiting mobility, it is still important to adapt the routing strategy to the underlying mobility model while not using resources in excess. In other words, the routing strategy ends up being a trade-off between resource utilization and network performance under a certain mobility model [51].

A very commonly studied routing strategy is Epidemic routing [60]. In its simplest form, Epidemic routing is a flooding technique in which nodes send copies to every node they contact and keep a copy to themselves, that is, its forwarding policy is to always send, while its replication policy is to always replicate. No dropping or scheduling policy is defined in its general characterization, though the optimization of these policies is a common concern in the literature (e.g., [5, 10]). Assuming an infinite buffer, this approach guarantees a delivery in the minimum amount of time, though at the cost of excessive resource consumption due to its exponential growth [25]. Given that an infinite buffer may be an unreasonable assumption (as previously discussed), the literature shows a substantial effort in finding ways to reduce the resource consumption of an Epidemic model (e.g., [17, 27, 31, 37]).

Chapter 3

Preliminaries

Our approach to the problem at hand utilizes some mathematical concepts which need to be defined clearly beforehand, in order to facilitate an unambiguous interpretation. Therefore, this chapter aims to lay the necessary groundwork for the later chapters. Specifically, in Section 3.1 we will describe Markov Chains and the related concepts of Markov property, transition probabilities, stationary state probability vector (3.1.1), unichains (3.1.2) and ergodicity (3.1.2), while in Section 3.2 we will define the Kronecker Product and its properties.

3.1 Markov Chains

A Markov Chain [7] is a model that aims to describe stochastic behaviour by defining the probability of transitioning from any current state to each possible future state. These transitions possess the Markov property, meaning that their probability depends solely on the current state and not on any past sequence of states (also known as the memoryless property).

From this point on, we will only address a specific subset of Markov chains, the finite-state discrete-time homogeneous Markov chains. This type of chain is essentially a sequence of random variables $X_1, X_2, X_3, ...$, each being an element present in the state space $\mathcal{X}$, which has size s. Transitions from one state to another occur at time instants $0, 1, 2, ...$, with a fixed probability. Therefore, a transition from any state $x \in \mathcal{X}$ to a next state $y \in \mathcal{X}$ at any instant $k \in \mathbb{N}_0$, has a conditional probability that can be represented as

$$p_{xy} \equiv P[X_{k+1} = y | X_k = x]. \tag{3.1}$$

This representation stems from the fact that the probability is independent of any past state history (Markov property) and of the instant in which the transition occurs (time-homogeneous). Given this, we can describe the probability of all state transitions through a square matrix (one row and column for each state) of non-negative terms in which the sum of each row is equal to 1. This is the transition probability matrix $\mathbf{P} \equiv [p_{xy}]$.

Additionally, the probability of finding the chain on a certain state $x \in \mathcal{X}$ at instant $k \in \mathbb{N}_0$ (known as state probability), is described as

$$\pi_x(k) \equiv P[X_k = x] \tag{3.2}$$

and consequently we can define a state probability vector, which is a probability distribu-

tion described as

$$\begin{aligned} \boldsymbol{\pi}(k) &= [\pi_0(k), \pi_1(k), ...] \\ \boldsymbol{\pi}(k+1) &= \boldsymbol{\pi}(k) \cdot \mathbf{P} \end{aligned} \tag{3.3}$$

It follows that, given the initial state probability vector $\boldsymbol{\pi}(0)$, we can use the recursive equation (3.3) to calculate the state probability vector for any instant.

Taking all this into account, it becomes apparent that a chain of this type can be uniquely described by specifying its state space, transition probability matrix and initial state probability.

3.1.1 Stationary State Probability Vector

As mentioned previously, by having a fully described Markov chain we are able to analyse future behaviour of the model, and thus it is possible to calculate the probability of being at a certain state at some point in time (that is, we know the value of $\pi_x(k)$ for any $x \in \mathcal{X}, k \in \mathbb{N}_0$). This type of analysis in which we focus on the behaviour of the chain over a finite period of time is called *transient analysis*.

However, it would also be interesting to analyse how a model behaves in the "long run", that is, after operating for a period of time which is long enough that the probability of being in each state of the chain will no longer change with time [7]. In other words, we are carrying out a *steady state analysis* of the chain at a $k \in \mathbb{N}_0$ large enough such that

$$\boldsymbol{\pi}(k+1) = \boldsymbol{\pi}(k). \tag{3.4}$$

Once the scenario in (3.4) is met, the state probabilities will no longer change over time, that is, they are *stationary*. In that case, these state probabilities form the stationary state probability vector $\boldsymbol{\pi}$. More formally, it can be described as:

$$\pi_x = \lim_{k \to \infty} \pi_x(k) \tag{3.5}$$

$$\boldsymbol{\pi} = [\pi_1, ..., \pi_s] \tag{3.6}$$

where $\pi_x \geq 0 \ \forall \ x \in \{1, \ldots, s\}$ and $\boldsymbol{\pi}$ is independent of the initial state probability vector.

While $\boldsymbol{\pi}$ is of particular interest for our analysis, in practice it is evidently impossible to calculate it by operating the model infinitely and, in addition, the convergence is not guaranteed to be the same regardless of the initial state, or to even occur. However, we know that $\boldsymbol{\pi}$ stays unchanged after one instant and that the sum of its elements is equal to one (because it is a probability distribution). Therefore, we can easily deduce that $\boldsymbol{\pi}$ will be a solution of the following system of linear equations

$$\begin{cases} \boldsymbol{\pi}\mathbf{P} &= \boldsymbol{\pi} \\ \boldsymbol{\pi}\mathbf{e}^{\mathsf{T}} &= 1 \end{cases} \tag{3.7}$$

with $\mathbf{e}$ being a row vector with the appropriate dimensions where every element is equal to one and $^{\mathsf{T}}$ being the transpose operator.[1]

Given that the chain being considered is finite, the system of linear equations at (3.7) is guaranteed to have at least one solution, although it might not be unique [14]. Another

[1] For computational reasons, it is convenient to express this system in the matrix form $\mathbf{Ax} = \mathbf{b}$, with $\mathbf{A}$ being a square matrix. Therefore, this system can also be represented as

$$\boldsymbol{\pi}[\mathbf{P} \ \mathbf{e}^{\mathsf{T}}] = [\boldsymbol{\pi} \ 1] \tag{3.8}$$

important consideration is that, if the limits (3.5) exist, then (3.6) is guaranteed to be a solution to the system (3.7), but the inverse is not necessarily true; that is, finding a solution to this system does not imply that the limits (3.5) exist. Taking all this into account, the calculation of a stationary state probability vector is only guaranteed if the chain being analysed has certain characteristics which force the system (3.7) to have a unique solution, to which the state probabilities converge regardless of initial conditions.

The literature indicates that if we have a *unichain* we are guaranteed to obtain a unique solution for this system, which is independent of the initial state probability distribution, and, if that unichain is also *ergodic*, then the state probabilities are guaranteed to converge to this solution [14]. Therefore, being an ergodic unichain is a sufficient condition to have a stationary state probability vector which is computable. In the next section we will explore this type of Markov chain.

3.1.2 Unichains

In order to define *unichains*, we first need to introduce some Markov chain concepts. A *class* is a non-empty set of states in which each state in the set can *communicate* with any other state in the set but cannot communicate with any state outside of that set. We say that the states x and y communicate if x is *accessible* from y and y is *accessible* from x, that is, the chain can eventually reach one state starting from the other and vice-versa. Consequently, by definition, a state always belongs to one class, since it is guaranteed to at very least belong to a class composed by only itself. Classes are classified as either *recurrent* or *transient*. A recurrent class is a class in which every one of its states is recurrent, that is, if the chain is at one of these states we are sure that it will eventually return to that same state. A transient class is a class in which every one of its states is transient, that is, there are no recurrent states.

A unichain is a finite Markov chain that contains a single recurrent class plus, perhaps, some transient states [14]. In a unichain, after a long enough run, states in the recurrent class have a non-zero probability of being visited eventually, while all other states have zero probability. Summing up, we can interpret a unichain as being a Markov chain which may have some initial transient behaviour, but that will inevitably enter the recurrent class in the future; thus, its long term behaviour will be the same as if it did not have its transient states, that is, it will act as an irreducible Markov chain (a chain in which all states of its state space are recurrent), which has desirable properties for a steady state analysis.

Let's consider a certain finite-state Markov chain in which we have one state x which is accessible from all other states but does not necessarily communicate with them. It is clear that x is recurrent and therefore it is part of a recurrent class $\mathcal{A}$. If a certain state $y \neq x$ is accessible from x, then x and y communicate and therefore $y \in \mathcal{A}$. On the other hand, if y is not accessible from x, then y is part of a certain class $\mathcal{B}$, which is composed of states that are not accessible from any state in $\mathcal{A}$. Given that $\mathcal{A}$ is accessible from all states in $\mathcal{B}$, it is clear that if the chain is at a state in $\mathcal{B}$ it will eventually reach a state in $\mathcal{A}$ and never be able to return to a state in $\mathcal{B}$, meaning that every state in $\mathcal{B}$ is transient. Thus,

which is equivalent to

$$[(\mathbf{P} - \mathbf{I})\ \mathbf{e}^\mathsf{T}]^\mathsf{T}\boldsymbol{\pi}^\mathsf{T} = [0\ 0\ ...\ 0\ 1]^\mathsf{T} \tag{3.9}$$

with $\mathbf{I}$ being the identity matrix and $[\mathbf{A}\ \mathbf{B}]$ representing the horizontal concatenation of matrices $\mathbf{A}$ and $\mathbf{B}$. The system at (3.9) is now in the intended matrix form (with $\mathbf{A} = [(\mathbf{P} - \mathbf{I})\ \mathbf{e}^\mathsf{T}]^\mathsf{T}$, $\mathbf{x} = \boldsymbol{\pi}^\mathsf{T}$ and $\mathbf{b} = [0\ 0\ ...\ 0\ 1]^\mathsf{T}$), but $\mathbf{A}$ is a $(s+1) \times s$ matrix and therefore not square as is preferred. Given that the first sth rows of $\mathbf{A}$ are linearly dependent, we can obtain a square system of linear equations without modifying the solution $\mathbf{x}$ by removing one of those rows, as well as the corresponding row of $\mathbf{b}$.

we conclude that if a Markov chain has at least one state such as x, then there is a single recurrent class and possibly some transient classes, that is, it is a unichain. Consequently, if we can demonstrate that, for a certain chain there is one state which is accessible from all other states, then we prove that it is a unichain (conversely, if we prove that no such state exists, we also prove that it is not a unichain). This conclusion will be useful in the future for us to determine the characteristics of our models.

Ergodicity

An ergodic unichain is a unichain for which the recurrent class is ergodic [14], that is, all of its recurrent states are aperiodic positive recurrent. Since this recurrent class is (by definition) a finite closed irreducible set of states, then all of its states are positive recurrent [7, theorem 7.6] and they have the same period [7, theorem 7.7]; consequently, proving that a state of the recurrent class is aperiodic also proves that the unichain is ergodic. As mentioned previously, if the chain is at a state in the recurrent class, then it is guaranteed return to that state in the future. There may be many different sequences of transitions which lead to this return, possibly with different numbers of transitions (at least one), that is, different return times; If, for a certain state, we form a set of all these possible return times, and calculate the greatest common divisor d of this set, then that state is aperiodic if $d = 1$ and periodic with period d if $d > 1$.

An interesting situation arises when a unichain is not aperiodic. In this case, (3.7) will still result in a unique state probability vector, which is "stationary" in the sense that it stays unchanged with iterations of (3.3), that is, the state probabilities do not change over time. However, because the states are periodic, the limit (3.5) does not exist, given that if we start the chain at any state, in the long run the state probabilities will keep alternating between d different vectors. This occurs because the recurrent class can be divided into d mutually exclusive subsets, which can be arranged in a cycle so that states from each subset can only transition to states of the next subset. Intuitively, it is clear that, in the long run, the chain will spend the same amount of time in each of these subsets and, therefore, the proportion of time spent on each state is simply the arithmetic mean of these d different state probability vectors. Conveniently, this result is equivalent to the solution of (3.7). This means that, when dealing with a periodic unichain, long-term properties that are dependent on this proportion of time can still be analysed.

3.2 Kronecker Product

The Kronecker product [61] is an operation between a matrix $\mathbf{A} = [a_{ij}]$, with dimensions $x \times y$, and another matrix $\mathbf{B}$, with dimensions $u \times v$, which can represented as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1y}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{x1}\mathbf{B} & \cdots & a_{xy}\mathbf{B} \end{bmatrix} \tag{3.10}$$

with $x, y, u, v \in \mathbb{N}$ and the result being a $xu \times vy$ matrix. It is important to note that while this operation is associative, it is not commutative and therefore $\mathbf{A} \otimes \mathbf{B} \neq \mathbf{B} \otimes \mathbf{A}$ in the general case.

A particularly interesting property occurs when these matrices have a single row, essentially being the row vectors $\mathbf{a}$ and $\mathbf{b}$, of lengths y and v respectively. In this case, the Kronecker

product can be represented as

$$\begin{aligned}\mathbf{a} \otimes \mathbf{b} &= \begin{bmatrix} a_1\mathbf{b} & \cdots & a_y\mathbf{b} \end{bmatrix} \\ &= \begin{bmatrix} a_1b_1 & a_1b_2 & \cdots & a_1b_v & a_2b_1 & \cdots & \cdots & a_yb_1 & a_yb_2 & \cdots & a_yb_v. \end{bmatrix}\end{aligned} \tag{3.11}$$

resulting in a row vector of length yv. The interest in this resulting vector lies in the fact that its elements essentially correspond to a Cartesian product between the sets of elements of $\mathbf{a}$ and $\mathbf{b}$, respectively; therefore, if vectors $\mathbf{a}$ and $\mathbf{b}$ represented probability distributions (each element representing the probability of an event), the result of this operation would be the probability distribution of all event intersections, assuming that the events of one vector are independent from the events of the other vector. This is particularly useful when constructing a transition probability matrix of a Markov chain if each state of that chain corresponds to a combination of events.

Another useful property of the Kronecker product of row vectors is that it allows us to simplify some operations, for example

$$\begin{bmatrix} \mathbf{a} \otimes \mathbf{b} \\ \mathbf{a} \otimes \mathbf{c} \\ \mathbf{a} \otimes \mathbf{d} \end{bmatrix} = \mathbf{a} \otimes \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \\ \mathbf{d} \end{bmatrix} \tag{3.12}$$

with $\mathbf{a}$ of length y and $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$ of length v. This is extremely useful for computational purposes, as it allows us to describe multiple Kronecker products between two row vectors as a single Kronecker product between a row vector and a matrix composed by multiple row vectors.

Chapter 4

Modelling a Urban Sensing Scenario

Having introduced all the necessary concepts in previous chapters, this Chapter will be focused on exploring a model that we have designed to analyse a certain type of DTN scenarios. Therefore, in Section 4.1 we will describe the assumptions and simplifications of our model along with details regarding its states (4.1.1) and transitions (4.1.2). Then, in Section 4.2 we will detail an analysis framework which allows us to calculate multiple network metrics for our model. Afterwards, in Section 4.3 we will exemplify how we could implement other mobility models without significantly changing our general model. In Section 4.4, we will also explore a change to the model which would allow us to describe some scenarios with greater detail, and we will discuss how that would affect the properties of our model (4.4.1) and consequently its analysis (4.4.2). Section 4.5 has an overview of how the necessary calculations were implemented and the main considerations that were taken into account. Finally, in Section 4.6 we will discuss the verification of this model, while in Section 4.7 we will run some small experiments to understand how the system behaves with different parameters.

4.1 Model Description

As mentioned in Chapter 1, one interesting application of a DTN with a SCF paradigm emerges in a urban sensing scenario, where sensors collect data to be gathered in a DCU, which then forwards such data to nearby mobile entities carrying an OBU (mobile nodes). The goal is for this data to eventually be delivered to an RSU (end node). Since this application is described in a very general form, in order to tackle the challenge of modelling it is necessary that we simplify some of its elements and make some assumptions about it.

We will start by assuming that all units (that is, one DCU, one RSU and multiple OBUs) are located on a line segment (a street) of a certain length. To simplify the analysis, we will discretize the system spatially, by dividing this segment into $n-1$ intervals of equal length, so that the ends of these intervals represent n possible locations for units to be in. Increasing the number of positions being considered implies that the intervals are smaller, which logically increases the spatial resolution of the model. We assume that the DCU is always in the first location, while the RSU is always in the last location. In this context, the contact range of all units is assumed to be c, such that, if the absolute difference between the positions of two units is no greater than c, then communication is possible. Moreover, n and c have to be such that the direct communication between the DCU and the RSU is impossible (otherwise communication would not require entities, making it a trivial scenario).

It is assumed that entities can be at any position, including where there are other entities; this is analogous to entities passing by each other in a real-world scenario. While we do not assume anything about the nature of the entities themselves, it is assumed that all entities move in a similar way, that is, according to the same mobility model. Moreover, we assume that the number of entities present within this segment is always m.

As a further simplification, we will also discretize the advance of time. New data is generated at each instant, such that the DCU always carries data that was created in that instant. In order to distinguish data generated at different points in time, we will associate each copy of data with its age, that is, the number of instants that have passed since that data was generated at the DCU. Therefore, entities carrying data of the same age are carrying copies of the exact same data, and data that is older by one unit in the next instant is still the same data. However, in order to keep this model finite, we will assume that all data in the system has a time-to-live denoted by t, such that when data reaches an age of t it is considered to be expired; this represents the maximum amount of time that the RSU is willing to wait to receive data after its creation.

While in the real world it could be possible for units to have enough storage to carry more than a data packet simultaneously (buffer), we will simplify the scenario to assume that every unit has a storage that fits exactly one packet of data. Consequently, some sort of buffer management will be necessary.

In order to reach our goal, entities within contact range of one another may transmit data between one another, depending on the data that they are currently carrying and on the routing strategy; entities carrying non-expired data within contact range of the RSU will only transmit to it and they will discard the data afterwards (since our goal is only to deliver each data packet once).

We assume that data transmission occurs instantly, and therefore if two entities initiate a transfer on a certain instant, by the next instant it will have been completed, the data will be one time unit older and the entities may be in different positions. In practical terms this means that, between instants, this sequence of processes occurs: 1) the data is shared between units according to the communication model, routing strategies, and the entities' current position and data; 2) all data is aged; 3) the position of the mobile entities is updated according to the mobility model.

Due to these simplifications, we are describing behaviours that do not change over time and can be defined in a discrete and finite way. Therefore, this can be modelled with a homogeneous discrete-time Markov chain, as described in Section 3.1. In the following sections we will describe the state space and transition probability matrix. However, as it will be described in Section 4.2, our analysis will be focused on long-term behaviour, and therefore we do not need to define an initial state probability vector.

4.1.1 State Space

In this scenario, we have a set of m mobile entities, which we will map onto the set of positive integers such that $\mathrm{E} = \{1, 2..., m-1, m\}$. Each entity can move and carry data, therefore the state of the system can be uniquely described by a vector containing the location and data of each entity, that is, a vector $x = (l_1^x, ..., l_m^x, d_1^x, ..., d_m^x)$, $x \in \mathcal{X}$, in which l_i^x is the location of entity $i \in \mathrm{E}$ on that state and d_i^x is the age of the data being carried by entity $i \in \mathrm{E}$ on that state. It is assumed that the states in $\mathcal{X}$ are sorted in lexicographic order and are mapped onto the set of positive integers.

At any given instant, each entity is at one of n different locations, and multiple entities can be at the same location simultaneously, therefore $l_i^x \in \{1, 2, ..., n-1, n\}$. Given that there is no functional difference between carrying data that is expired or not carrying any data at all, we can represent both situations in the same way. Therefore, $d_i^x \in \{1, 2, ..., t-1, t\}$, and $d_i^x = t$ represents expired data or no data being carried.

Given that each of the m entities can be in n different locations and carry data of t different ages, the state space $\mathcal{X}$ contains $s = (n \cdot t)^m$ different states. Since this model is only reasonable with at least one entity, different locations for the DCU and the RSU and the possibility of having non-expired data, from here on we assume $m \geq 1$, $n \geq 2$ and $t \geq 2$.

4.1.2 State Transitions

Given that the state of the network is a combination of the location and data of each entity, we can describe all state transitions by first describing how an entity's location and data changes from one instant to another. Therefore, we will first explore the mobility and communication in our model separately, which will then allow us to describe all transition probabilities.

Entity Mobility

We will start by describing how an entity moves between locations. In our model we assume that this movement is stochastic and only depends on the current location of each entity, meaning that the movement of one entity at a given instant is independent of the movement of every other entity at that same instant. Therefore, given an entity $i \in \mathrm{E}$ in a state $x \in \mathcal{X}$, we want to describe g_{ib}^x which details the probability of that entity going from its current location l_i^x to a location $b \in \{1, ..., n\}$ in the next instant. This is referred to as a mobility model.

A specific instance of g_{ib}^x is one in which the entities perform a Random Walk. In this mobility model, entities have a certain probability of staying at their current position (α^*), moving one position towards n (α^+) or moving one position towards 1 (α^-), with the moves past the edges being impossible. Therefore, we have

$$g_{ib}^x = \begin{cases} \alpha^-, & if\ 1 \leq b = l_i^x - 1 \\ \alpha^+, & if\ b = l_i^x + 1 \leq n \\ \alpha^*, & if\ 1 < b = l_i^x < n \\ \alpha^* + \alpha^-, & if\ b = l_i^x = 1 \\ \alpha^* + \alpha^+, & if\ b = l_i^x = n \\ 0, & otherwise \end{cases} \tag{4.1}$$

with $\alpha^*, \alpha^+, \alpha^- > 0$ and $\alpha^* + \alpha^+ + \alpha^- = 1$.

Entity communication

We are also interested in describing how the model behaves regarding data. In other words, given an entity $i \in \mathrm{E}$ in a state $x \in \mathcal{X}$, we want to describe u_{ia}^x which details the probability of that entity transitioning from carrying data of age d_i^x to a carrying data

of age $a \in \{1, ..., t\}$ in the next instant. These transitions are direct consequences of the communication model and the routing strategies being used.

Regarding the communication model, it is important to note that the contact range c is at least 0 (meaning that entities can only communicate with other entities in the same location) and we will not consider scenarios in which the range is large enough to allow the DCU to communicate directly with the RSU, therefore $0 \leq c < n - 1$.

A specific instance of u^x_{ia} is one in which we consider a routing strategy similar to epidemic routing. In that case, entities will generally send their data to nearby entities whenever possible and keep a copy of that data. However, given that our buffer is limited to having a size of 1, the entity has to make decisions regarding which data to keep, if it has multiple choices (buffer management). We will assume that if an entity does not receive data that is newer than the one it is carrying, it will keep it; otherwise there is a chance r (replacing ratio, $0 \leq r \leq 1$) that it will replace its current data with newer data. Intuitively, low values of r may cause newer data to get lost before it has a chance to replicate throughout the network, while high values of r may reduce the variability of the data being delivered, thus the best value of r should depend on the specific situation.

With that in mind, the data that an entity will have in the next instant is either the one it is currently carrying or the newest data that it can receive from other entities. For a state $x \in \mathcal{X}$ and an entity $i \in \mathrm{E}$, the age of the data in these options are k^x_i and r^x_i (*keeping* and *receiving*), respectively. In a situation where neither the data it can keep nor the data it can receive have expired, and the data it can receive is newer than than the one it can keep (that is, $r^x_i < k^x_i < t$), the entity has a probability r of picking the new data and $1 - r$ of doing otherwise. In every other scenario (that is, $k^x_i = t$ or $k^x_i \leq r^x_i$), an entity will always pick the option that grants it the newest data. Therefore, this instance of u^x_{ia} can be described as

$$u^x_{ia} = \begin{cases} 1, & if\ (a = \min(k^x_i, r^x_i)) \wedge ((k^x_i = t) \vee (k^x_i \leq r^x_i)) \\ 1 - r, & if\ (a = k^x_i) \wedge (r^x_i < k^x_i < t) \\ r, & if\ (a = r^x_i) \wedge (r^x_i < k^x_i < t) \\ 0, & otherwise \end{cases} \tag{4.2}$$

Since entities discard their data if they transmit it to the RSU, k^x_i is t for every entity in contact range of the RSU. In every other instance, the data being carried will simply age by 1, until it expires, therefore:

$$k^x_i = \begin{cases} t, & if\ (n - l^x_i) \leq c \\ \min(d^x_i + 1,\ t), & otherwise \end{cases} \tag{4.3}$$

When it comes to receiving data, since transmissions are only completed between instants, data received will always be 1 time unit older than at the time of contact. If contact with the DCU is possible, an entity will simply receive new data from it (since it is guaranteed to be the newest data), and therefore r^x_i is 1 for all entities in contact range of the DCU. Otherwise, an entity will receive the newest possible data out of the set C^x_i, which has the data age of all entities in contact range that are not already contacting the RSU, thus:

$$r^x_i = \begin{cases} 1, & if\ (l^x_i - 1) \leq c \\ \min \mathrm{C}^x_i, & otherwise \end{cases} \tag{4.4}$$

with C^x_i being described as

$$\mathrm{C}^x_i = \{d^x_j + 1 : j \in \mathrm{E}, j \neq i, |l^x_i - l^x_j| \leq c,\ (n - l^x_j) > c\} \cup \{t\} \tag{4.5}$$

and with t being part of C^x_i since, in a worst case scenario, nothing will be received.

Transition Probability Matrix

Let's consider $\mathbf{G}^x \equiv [g^x_{ib}]$, a $m \times n$ matrix, and $\mathbf{U}^x \equiv [u^x_{ia}]$, a $m \times t$ matrix, with $\mathbf{g}^x_i$ and $\mathbf{u}^x_i$ representing the ith rows of $\mathbf{G}^x$ and $\mathbf{U}^x$, respectively. For a certain state $x \in \mathcal{X}$, these matrices contain the probability of, at the next instant, each entity being at a certain position and having data with a certain age, respectively. Since the state of the system is described entirely by the positions and data of each entity, these matrices have enough information to enable us to calculate, given the current state, the probability of the system being in any given state at the next instant.

For each entity, the transition of position and data depends solely on the current state, meaning that these transitions are independent from one another. Therefore, the probability of having any combination of positions and data at the next instant can be calculated simply by multiplying the probability of each individual transition occurring. As a result, the probability of state $x \in \mathcal{X}$ transitioning to each state can be calculated as the Kronecker Product between every row of the matrices $\mathbf{G}^x$ and $\mathbf{U}^x$, resulting in a row vector of length s (as defined in Section 4.1.1), that is:

$$\begin{aligned} \mathbf{g}^x_1 \otimes \mathbf{g}^x_2 \otimes ... \otimes \mathbf{g}^x_m \otimes \mathbf{u}^x_1 \otimes \mathbf{u}^x_2 \otimes ... \otimes \mathbf{u}^x_m &= \left(\bigotimes_{i=1}^{m} \mathbf{g}^x_i \right) \otimes \left(\bigotimes_{i=1}^{m} \mathbf{u}^x_i \right) \\ &= \boldsymbol{\gamma}^x \otimes \boldsymbol{\mu}^x \end{aligned} \tag{4.6}$$

This vector contains one element for each state in the state space, meaning that the yth element is the probability of transitioning from state x to state y. Therefore, if we calculate this vector for each $x \in \mathcal{X}$ and stack the results vertically in order, the outcome is the transition probability matrix $\mathbf{P}$, as follows:

$$\mathbf{P} = \begin{bmatrix} \boldsymbol{\gamma}^1 \otimes \boldsymbol{\mu}^1 \\ \boldsymbol{\gamma}^2 \otimes \boldsymbol{\mu}^2 \\ \vdots \\ \boldsymbol{\gamma}^s \otimes \boldsymbol{\mu}^s \end{bmatrix} \tag{4.7}$$

As intuitively expected, $\mathbf{P}$ is a $s \times s$ matrix.

Given that the mobility of entities depends solely on the entities' current position (as described in Section 4.1.2), states in which entities have the same position result in the same probabilities of transitions, that is, $\forall x, y \in \mathcal{X}$:

$$\left(\bigwedge_{i=1}^{m} (l^x_i = l^y_i) \right) \implies (\boldsymbol{\gamma}^x = \boldsymbol{\gamma}^y) \tag{4.8}$$

Since states are first sorted by the entities' position (as described in Section 4.1.1), we know we can divide the state space into n^m groups, each with t^m consecutive states, in which states in the same group have their entities in the same position. Thus, states that are part of the same group have the same γ, and therefore we can simplify the calculation

of $\mathbf{P}$ as such:

$$\mathbf{P} = \begin{bmatrix} \boldsymbol{\gamma}^1 \otimes \begin{bmatrix} \boldsymbol{\mu}^1 \\ \boldsymbol{\mu}^2 \\ \vdots \\ \boldsymbol{\mu}^{t^m} \end{bmatrix} \\ \boldsymbol{\gamma}^{t^m+1} \otimes \begin{bmatrix} \boldsymbol{\mu}^{t^m+1} \\ \boldsymbol{\mu}^{t^m+2} \\ \vdots \\ \boldsymbol{\mu}^{2t^m} \end{bmatrix} \\ \vdots \\ \boldsymbol{\gamma}^{(n^m-1)t^m+1} \otimes \begin{bmatrix} \boldsymbol{\mu}^{(n^m-1)t^m+1} \\ \boldsymbol{\mu}^{(n^m-1)t^m+2} \\ \vdots \\ \boldsymbol{\mu}^{s} \end{bmatrix} \end{bmatrix} \tag{4.9}$$

4.2 Analysis Framework

The system that was previously defined has a set of parameters that allow us to model multiple configurations of a urban sensing scenario. When implementing such a scenario in a real-world setting, there will be some aspects of the system that are inherent to that specific setting (e.g. how the entities move in that location), while other aspects will be entirely dependent on the specific implementation (e.g. how we want the entities to communicate). It follows that, when using our approach to model a setting, some of the parameters will have to be chosen in a way that replicates the physical characteristics and constraints that have been determined to exist, while the other parameters can be freely defined to suit one's needs. Logically, it is interesting to define the latter parameters in a way that somehow optimizes the system, for example, maximizing the "performance" of the network while reducing the "cost" of the implementation.

In this section, we will be mostly interested in issues that relate to the performance of the network. A network's performance can be evaluated in many ways, depending on one's goals. Therefore, we need to define a set of network metrics which will serve as criteria for optimization. Since the urban sensing scenarios being described are intended to be implemented with a fixed configuration that is used continuously, it is intuitive to understand that we are mostly interested in studying how certain configurations behave, on average, over a long period of time. Therefore, in this chapter we will define metrics that measure different aspects of the long-term behaviour of the network.

As previously described in Section 3.1.1, the concept of a stationary state probability vector is extremely important for the study of long-term behaviour of a Markov Chain. However, before using it, we need to ensure that our model allows its calculation. As mentioned previously, the model being studied has a *finite* state space, thus the first condition of being a unichain is met. Let's assume we are using the mobility model described in Equation (4.1) and the communication model and routing strategy described in Equation (4.2). It is clear that, no matter what position or data entities start with, there is a chance that all of them eventually reach the last position and deliver all non-expired data to the RSU. This means that the state in which all entities are at the last location without any data can eventually be reached starting from any state, which is enough to guarantee that we are dealing with a unichain. One way to ensure that a state is aperiodic is to allow it to return to itself in one transition, since 1 would be part of the set of return times (as

described in 3.1.2) and thus the greatest common divisor would necessarily be 1. In the chain being studied, a state transitions to itself if (and only if) all entities stay at the same location, have no data (otherwise it would age) and are outside the contact range of the DCU (otherwise they would receive new data). Given that our model guarantees that there is at least one location outside the contact range of the DCU (due to constraints on c as described in 4.1.2), as long as the mobility model being used has a non-zero probability of the entities staying at such location, then our Markov chain is guaranteed to be an ergodic unichain. This is the case with the mobility model described in Equation (4.1) and therefore we can calculate $\boldsymbol{\pi}$.

In the following sections, we will describe how to calculate performance metrics based on the stationary state probability vector. The analysis techniques presented here were intentionally designed to be applicable for any model as defined in 4.1 and therefore it provides a framework for analysing any system of this kind.

4.2.1 Calculating the Probability of Sequences of States at Steady State

Some interesting metrics regarding the long-term behaviour of the model can be deduced by calculating the probability of a certain sequence of states occurring in the future, under the assumption that the sequence starts at a point where the state probability vector is stationary. We will introduce a method for calculating all metrics of this type.

If we consider k to be a sufficiently large number so that the state probability vector is $\boldsymbol{\pi}$, we have that the probability of the chain being in state x is

$$P[X_k = x] = \pi_x \tag{4.10}$$

for any $x \in \mathcal{X}$. We can generalize this result to obtain the probability of the chain being in any state belonging to a certain set $\mathcal{A}$. Given that the chain cannot be in different states at the same instant, these events are mutually exclusive and thus we have

$$\begin{aligned} P[X_k \in \mathcal{A}] &= \sum_{x \in \mathcal{A}} P[X_k = x] \\ &= \sum_{x \in \mathcal{A}} \pi_x \end{aligned} \tag{4.11}$$

with $\mathcal{A} \subseteq \mathcal{X}$.

Similarly, we can define the probability of the chain having a certain sequence of three states as

$$\begin{aligned} &P[X_k = x \wedge X_{k+1} = y \wedge X_{k+2} = z] \\ &= P[X_k = x] P[X_{k+1} = y | X_k = x] P[X_{k+2} = z | X_{k+1} = y \wedge X_k = x] \\ &= P[X_k = x] P[X_{k+1} = y | X_k = x] P[X_{k+2} = z | X_{k+1} = y] \\ &= \pi_x p_{xy} p_{yz} \end{aligned} \tag{4.12}$$

with $x, y, z \in \mathcal{X}$.[1] This can again be generalized to consider the probability of a sequence in which each element may be any state belonging to a certain set. Since the chain cannot follow different sequences of states simultaneously, the probability of any such sequence

[1] The third line in (4.12) is a result of the Markov property, as introduced in Section 3.1.

occurring is simply the sum of each individual probability, that is

$$\begin{aligned} &P[X_k \in \mathcal{A}_0 \wedge X_{k+1} \in \mathcal{A}_1 \wedge X_{k+2} \in \mathcal{A}_2] \\ &= \sum_{x \in \mathcal{A}_0} \sum_{y \in \mathcal{A}_1} \sum_{z \in \mathcal{A}_2} P[X_k = x \wedge X_{k+1} = y \wedge X_{k+2} = z] \\ &= \sum_{x \in \mathcal{A}_0} \sum_{y \in \mathcal{A}_1} \sum_{z \in \mathcal{A}_2} (\pi_x p_{xy} p_{yz}) \end{aligned} \tag{4.13}$$

with $\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2 \subseteq \mathcal{X}$. The results (4.12) and (4.13) can clearly be generalized for sequences of any length, by simply adding or removing the corresponding terms. Moreover, given that the state space is finite, guaranteeing that the chain is not in a state that belongs to a certain set is equivalent to guaranteeing that the chain is in a state that belongs to the set of all states that do not belong to A, that is, $X_k \notin A \iff X_k \in (\mathcal{X} \setminus A) \iff X_k \in \overline{A}$. Therefore, this approach to calculating probabilities of sequences can also be used in scenarios which involve guaranteeing that, at certain instants, the chain is not at a state belonging to a certain set .

For computational reasons, it would be convenient to represent (4.13) as a result of matrix operations using $\mathbf{P}$ and $\boldsymbol{\pi}$. Let's consider $\mathbf{Z}^{\mathcal{A}} \equiv [z_{xy}^{\mathcal{A}}]$ to be a binary diagonal $s \times s$ matrix such that

$$z_{xy}^{\mathcal{A}} = \begin{cases} 1, & \text{if } x = y \wedge x \in \mathcal{A} \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

The interest in such a matrix lies in the fact that, if a state probability vector is multiplied by $\mathbf{Z}^{\mathcal{A}}$, the result is that same vector but with the elements of index not contained in $\mathcal{A}$ having been replaced by 0. This means that the probability of being in a state belonging to set $\mathcal{A}_0$ is

$$\sum_{x \in \mathcal{A}_0} \pi_x = \boldsymbol{\pi} \cdot \mathbf{Z}^{\mathcal{A}_0} \cdot \mathbf{e}^{\mathsf{T}} \tag{4.15}$$

and, similarly, the probability of being in a state belonging to $\mathcal{A}_0$ and, in the next instant, being in a state belonging to $\mathcal{A}_1$ is

$$\sum_{x \in \mathcal{A}_0} \sum_{y \in \mathcal{A}_1} (\pi_x p_{xy}) = \boldsymbol{\pi} \cdot \mathbf{Z}^{\mathcal{A}_0} \cdot \mathbf{P} \cdot \mathbf{Z}^{\mathcal{A}_1} \cdot \mathbf{e}^{\mathsf{T}} \tag{4.16}$$

Thus, for a sequence of sets $(\mathcal{A}_0, \mathcal{A}_1, ..., \mathcal{A}_v) \subseteq \mathcal{X}$, we have

$$P[X_k \in \mathcal{A}_0 \wedge X_{k+1} \in \mathcal{A}_1 \wedge ... \wedge X_{k+v} \in \mathcal{A}_v] = \mathbf{q}(v) \cdot \mathbf{e}^{\mathsf{T}} \tag{4.17}$$

with $\mathbf{q}$ being a row vector such that:

$$\mathbf{q}(i) = \mathbf{q}(i-1) \cdot \mathbf{P} \cdot \mathbf{Z}^{\mathcal{A}_i} \tag{4.18}$$

$$\mathbf{q}(0) = \boldsymbol{\pi} \cdot \mathbf{Z}^{\mathcal{A}_0} \tag{4.19}$$

Therefore, we now have a framework that allows us to calculate, for any sequence of sets of states, the probability of the chain going through a sequence of states contained in each of those sets, through matrix operations.

Data Loss

As detailed on 4.1.1, at any given instant the DCU generates data, that may be spread and copied between entities. However, a certain data generated at an instant k may not

have any copy delivered to the RSU at any point in time. This is called a *data loss* and the probability of it occurring is a useful metric for the performance of the network.

In order to guarantee that a certain data is never delivered, it is only necessary to guarantee that it is not delivered before it expires. Therefore, this metric can be calculated through the analysis of the sequence of states that occurs after the creation of the data and before its expiry. It is important to note that any data of age $a, 1 \leq a < t-1$ on an instant k is the exact same as any data of age $a+1$ at an instant $k+1$.

Whenever we are in a state in which data of age $a, 1 \leq a < t$ is delivered to the RSU, we say that we are in a delivery state of a and therefore the set of delivery states of a is:

$$\mathcal{D}_a = \{x \in \mathcal{X} : \exists i \in \mathrm{E}, d_i^x = a \wedge (n - l_i^x) \leq c\} \tag{4.20}$$

That is, $\mathcal{D}_a$ contains all the states in which at least one entity has data of age a and is within contact range of the RSU.

We can therefore describe data loss simply as a sequence of states in which, $\forall a : 1 \leq a < t$, the state at instant $k+a$ is not part of $\mathcal{D}_a$, that is:

$$loss = P\left[\bigwedge_{a=1}^{t-1} (X_{k+a} \notin \mathcal{D}_a)\right] \tag{4.21}$$

Therefore, we can calculate this probability as described in 4.2.1, using the sequence $(\overline{\mathcal{D}_1}, \overline{\mathcal{D}_2}, ..., \overline{\mathcal{D}_{t-1}})$.

In general, loss of data can occur in one of two circumstances, namely (i) the DCU does not deliver a copy of that data to any entity at any instant (ii) the DCU delivers a copy of that data to at least one entity, but no copy reaches the RSU. We name the former "loss at the DCU" and the later "loss in transit". While both circumstances contribute to the same final result, they indicate different challenges in the delivery.

As described in 4.1.2, at every instant the DCU generates new data which replaces the data it held previously. If this data being replaced was not received by at least one entity, then it is guaranteed to be lost. Since having entities in range of the DCU does not guarantee that a copy will be delivered (unless $r = 1$), it is necessary to look at the instant following the generation of the data to verify if a copy was delivered. That is, for data generated at an instant k, the probability of it being lost at the DCU is equivalent to the probability of, at instant $k+1$, the system not being in any state in which an entity is carrying data of age 1, which is represented as:

$$loss\_at\_DCU = P\left[X_{k+1} \notin \mathcal{C}_1\right] \tag{4.22}$$

with $\mathcal{C}_a$ being the set of states in which at least one entity has data of age a, which can be described as:

$$\mathcal{C}_a = \{x \in \mathcal{X} : \exists i \in \mathrm{E}, d_i^x = a\} \tag{4.23}$$

and therefore, we can calculate this probability as described in 4.2.1 with simply $(\overline{\mathcal{C}_1})$. Losses at the DCU occur when the DCU is generating data at a faster rate than it has available entities within its contact range. In a scenario where losses at the DCU are excessive, it is logically possible to reduce the losses at the DCU by increasing the probability of having entities in range (either by having more entities or changing the mobility model) or by increasing r and/or c. That said, in a real-world scenario, we would only be able to adapt r to the circumstances and, given that r also significantly affects the losses in transit, this might prove to not be enough to enable us to configure a network efficiently. Therefore,

it would be interesting to be able to configure the rate at which the DCU generates new data; this alternative will be studied later on.

In contrast, a loss in transit occurs when all existing copies of certain data get eventually replaced (with newer data) and/or discarded (due to expiry) before the entities carrying those copies ever get in range of the RSU. Given that losses at the DCU and in transit are mutually exclusive events, we can simply calculate the probability of losses in transit as:

$$loss\_in\_transit = loss - loss\_at\_DCU \tag{4.24}$$

A high probability of losses in transit indicates that, for some reason, the entities are prone to lose the data that they are carrying before getting close enough to the RSU. We believe this to be a good indicator of the performance of the entities as carriers and replicators of data under a given set of parameters. Given that a decrease in this metric might just be a consequence of an increase of losses at the DCU (as that results in less data in transit), it would be interesting to analyse the probability of losses in transit assuming the data was not lost at the DCU, that is:

$$loss\_received = P\left[\bigwedge_{a=1}^{t-1}(X_{k+a} \notin \mathcal{D}_a) \mid X_{k+1} \in \mathcal{C}_1\right] \tag{4.25}$$

$$= \frac{loss\_in\_transit}{1 - loss\_at\_DCU} \tag{4.26}$$

Average Latency

Another way of evaluating the performance of this network is measuring, on average, how much time passes between the creation of data and the first delivery of that same data to the RSU, assuming that data will be delivered. This is essentially the average data age on first delivery, which can be interpreted as the network's delivery latency.

We can describe a first delivery as essentially a delivery occurring at an instant $k+a$ (with $1 \leq a < t$) in which every prior instant $k+q$ (with $0 \leq q < a$) has no deliveries of that same data. Since we are not interested in what may happen after the first delivery, the probability of having a first delivery with age a is given by f_a, a probability mass function such that

$$f_a = P\left[\left(\bigwedge_{q=1}^{a-1}(X_{k+q} \notin \mathcal{D}_q)\right) \wedge (X_{k+a} \in \mathcal{D}_a)\right] \tag{4.27}$$

This is very similar to the calculations in 4.2.1, but instead of having a sequence of non-deliveries throughout the data's entire lifespan, we have a sequence of some non-deliveries followed by one delivery. Therefore, we can calculate this in a similar way; for example, calculating f_3 can be done as described in 4.2.1, using the sequence $(\overline{\mathcal{D}_1}, \overline{\mathcal{D}_2}, \mathcal{D}_3)$.

After calculating f_a for all $a, 1 \leq a < t$, determining the average age on first delivery is achieved by calculating an average weighted by the age of delivery, that is:

$$latency = \frac{\sum_{a=1}^{t-1}(a \cdot f_a)}{\sum_{a=1}^{t-1} f_a} \tag{4.28}$$

$$= \frac{\sum_{a=1}^{t-1}(a \cdot f_a)}{1 - loss} \tag{4.29}$$

It is important to note that under certain conditions deliveries may be impossible and therefore the latency will be undefined.

Probability of Delivering a Single Copy

In an ideal scenario, the RSU would receive exactly one copy of all data generated by the DCU, since extra copies provide no further information and therefore represent wasted resources. As such, it would be interesting to calculate the probability of a data point being delivered just once. Let's consider $\mathcal{O}_a$ which is a set of all delivery states of age a, $1 \leq a < t$, in which only one copy of such age is delivered, that is:

$$\mathcal{O}_a = \{x \in \mathcal{X} : \exists! i \in \mathrm{E}, d_i^x = a \wedge (n - l_i^x) \leq c\} \tag{4.30}$$

Note that $\mathcal{O}_a$ differs from $\mathcal{D}_a$ in that the latter also contains states in which there are multiple deliveries of data of age a (and therefore $\mathcal{O}_a \subseteq \mathcal{D}_a$).

In order for only one copy to be delivered, the chain has to be in a state $x \in \mathcal{O}_a$ at the instant $k + a$ (for any $a \in \{1, ..., t-1\}$), while also not being in any delivery state of that same data at any other instant during the data's lifespan. Since this scenario cannot happen simultaneously for different a, the overall probability of delivering a single copy can be calculated by simply summing the probability for each different a, that is:

$$single = \sum_{a=1}^{t-1} P\left[\left(\bigwedge_{q=1}^{a-1}(X_{k+q} \notin \mathcal{D}_q)\right) \wedge (X_{k+a} \in \mathcal{O}_a) \wedge \left(\bigwedge_{q=a+1}^{t-1}(X_{k+q} \notin \mathcal{D}_q)\right)\right] \tag{4.31}$$

where each term of this sum is essentially a sequence of sets of states, which can be calculated as described in 4.2.1.

Given that we can calculate the probability of no delivery with (4.21), and now the probability of a single copy being delivered with (4.31), we can also calculate the probability of delivering two or more copies of the same data as $1 - (loss + single)$.

4.2.2 Data Age Distribution

Knowing the age distribution of data in the network is important, as it allows us to calculate useful metrics like how much time data tends to stay in the network, or how many entities are carrying data on average. In order to understand the distribution, we will calculate, for all ages $a \in 1, ..., t$, the proportion of entities carrying data of that age at steady state.

This is achieved by calculating, for each state $x \in \mathcal{X}$, the proportion of entities carrying data of age a multiplied by the stationary state probability of being on that state, which is essentially a weighted average as described below:

$$w_a = \sum_{x \in \mathcal{X}} \left(\pi_x \frac{\sum_{i \in \mathrm{E}}[d_i^x = a]}{m}\right) \tag{4.32}$$

This is essentially a probability mass function, with $[p]$ representing the Iverson bracket [16, Section 2.1: Notation], where:

$$[p] = \begin{cases} 1, & \text{if } p \text{ is true} \\ 0, & \text{otherwise} \end{cases} \tag{4.33}$$

It is important to note that w_t is the proportion of entities that do not carry any data (or carry expired data). Therefore, the proportion of entities carrying data (in other words, the overall utilization of the network's storage) is:

$$utilization = 1 - w_t \tag{4.34}$$

We can also calculate the average age of non-expired data, which is essentially a weighted average considering the proportion of each age $a, 1 \leq a < t$, that is:

$$average\_age = \frac{\sum_{a=1}^{t-1}(w_a \cdot a)}{1 - w_t} \tag{4.35}$$

Note that the sum is divided by the utilization, as we are only considering non-expired data.

4.3 Mobility Models

The mobility model previously described in Equation (4.1) served as a simple starting point for the analysis, but it is unrealistic in terms of describing real-world behaviour. This is due to the fact that vehicles usually move with a certain purpose and thus are unlikely to have the wandering path that is common for random walks. Therefore, it may be interesting to design mobility models that better mimic the behaviour of a real-world scenario. If designed according to the principles established in 4.2, these alternatives of g^x_{ib} can still be analysed in the same way.

4.3.1 Forward Walk

One possible approach to describing the mobility of the entities is assuming that they are moving in a one-way street, that is, they can only move in the direction from the DCU to the RSU. In simple terms, entities either move one position forward or stay in the same position. Given that there is a finite number of locations in our model, the entities would have to eventually stop at the last position, which would not create an interesting long-term behaviour. Therefore, we will consider that, if an entity moves forward at the last position, it will go back to the first position. Given that, this model can be described as:

$$g^x_{ib} = \begin{cases} 1 - \alpha^+, & if\ b = l^x_i \\ \alpha^+, & if\ b = (l^x_i \bmod n) + 1 \\ 0, & otherwise \end{cases} \tag{4.36}$$

with $0 < \alpha^+ < 1$.

4.3.2 Cellular Automaton

It is also possible to create a mobility model that describes more complex traffic behaviours. One approach we studied was to use a model akin to a stochastic traffic cellular automata. In this context, we also consider the positions to be cyclic and we suppose that each entity knows the distance to the closest entity ahead of them and uses that information to decide how it moves. In simple terms, an entity will not move forward if there is another entity in the next location, but the probability of moving forward increases with the distance to the entity ahead.

To better mimic cellular automata, each position can only hold one entity and therefore any state in which there are at least two entities at the same location is considered invalid. Since the specification of our model does not enforce this limitation, we have to guarantee

that the mobility model is able to immediately transition from an invalid state to a valid state, making those invalid states transient and consequently not relevant for the long-term behaviour of the chain. This also means that the choice of the valid state to which invalid states transit to does not affect our analysis in any way.

Moreover, given that entities cannot overtake each other in this model, we can have a sparser transition probability matrix without affecting the metrics being analysed by assuming that the entities are always in the same order. In this case, we will impose that for any $i \in \mathrm{E}$, entity i follows entity $(i \bmod m) + 1$. This limitation can similarly be enforced by considering invalid any state in which the entities do not follow this order.

Given the previously mentioned limitations, the set of invalid states $\mathcal{I}$ can be described as:

$$\mathcal{I}_1 = \left\{ x \in \mathcal{X} : \exists i, j \in \mathrm{E}, i \neq j, l_i^x = l_j^x \right\} \tag{4.37}$$

$$\mathcal{I}_2 = \left\{ x \in \mathcal{X} : \sum_{i \in \mathrm{E}} \left((l^x_{(i \bmod m)+1} - l_i^x) \bmod n \right) \neq n \right\} \tag{4.38}$$

$$\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2 \tag{4.39}$$

and thus the mobility model is described as:

$$g_{ib}^x = \begin{cases} \frac{1}{(l_j^x - l_i^x) \bmod n}, & if\ (x \notin \mathcal{I}) \wedge (b = l_i^x) \\ 1 - \frac{1}{(l_j^x - l_i^x) \bmod n}, & if\ (x \notin \mathcal{I}) \wedge (b = (l_i^x \bmod n) + 1) \\ 1, & if\ (x \in \mathcal{I}) \wedge (b = i) \\ 0, & otherwise \end{cases} \tag{4.40}$$

with $j = (i \bmod m) + 1$, that is, j is the entity in front of entity i.

4.4 Sampling Period

As discussed in section 4.2.1, losses at the DCU could be reduced by allowing the DCU to generate data at a slower rate and to keep the same data for some period of time. This period will be referred to as "sampling period", denoted by τ, and the age of the data at the DCU in state $x \in \mathcal{X}$ will be denoted by δ^x. Up until this point, we were considering that new data is generated at every instant, that is, $\tau = 1$; this meant that the data being carried by the DCU in any state $x \in \mathcal{X}$ always had age 0 ($\delta^x = 0$) and therefore entities could only receive data of age 1 from the DCU. As a result, we would lose data every single time that there is no entity in contact range of the DCU. In a decently sized scenario, it is extremely unlikely for entities to always be near the DCU, and thus under such scenarios data loss was extremely prevalent. This is problematic because, unless the scenario was trivial, the network never had an acceptable performance.

Essentially, having a sampling period allows us to age data at a frequency which is different from the positional updates. This allows a finer control of the discretization of space and time. Thus, from now on, we will consider that $\tau \in \mathbb{N}, \tau < t$ and consequently for any state $x \in \mathcal{X}$, $\delta^x \in \mathbb{N}_0, 0 \leq \delta^x < \tau$.

4.4.1 Model Considerations

In order to add this sampling period, it is necessary to change the definition of a state in this chain. Therefore, a state is now defined by a vector $x = (l_1^x, ..., l_m^x, d_1^x, ..., d_m^x, \delta^x)$,

$x \in \mathcal{X}$. The states are still sorted in lexicographic order and mapped onto the set of positive integers. Consequently, there are now $s = (n \cdot t)^m \cdot \tau$ different states.

Since the movement of the entities is completely independent of any data considerations, this change does not affect the mobility models. However, the communication has to be updated in regards to the reception of data from the DCU, that is, we simply need to change (4.4) as such:

$$r_i^x = \begin{cases} \delta^x + 1, & if\ (l_i^x - 1) \leq c \\ \min \mathrm{C}_i^x, & otherwise \end{cases} \tag{4.41}$$

that is, entities in range of the DCU receive whatever data is currently stored there, instead of always assuming the DCU stores data of age 0; everything else can be kept unchanged.

It is also necessary to define the transitions of δ. Given any state $x \in \mathcal{X}$ and a next state $y \in \mathcal{X}$, we know that $\delta^y = (\delta^x + 1) \bmod \tau$. Therefore we have $\boldsymbol{\epsilon}^x$, a vector of size τ, in which the δ^yth element is equal to 1, while all the other elements are equal to 0. We can then describe the Transition Probability Matrix as

$$\mathbf{P} = \begin{bmatrix} \boldsymbol{\gamma}^1 \otimes \begin{bmatrix} \boldsymbol{\mu}^1 \otimes \boldsymbol{\epsilon}^1 \\ \boldsymbol{\mu}^2 \otimes \boldsymbol{\epsilon}^2 \\ \vdots \\ \boldsymbol{\mu}^{\tau t^m} \otimes \boldsymbol{\epsilon}^{\tau t^m} \end{bmatrix} & & \\ & \boldsymbol{\gamma}^{\tau t^m + 1} \otimes \begin{bmatrix} \boldsymbol{\mu}^{\tau t^m + 1} \otimes \boldsymbol{\epsilon}^{\tau t^m + 1} \\ \boldsymbol{\mu}^{\tau t^m + 2} \otimes \boldsymbol{\epsilon}^{\tau t^m + 2} \\ \vdots \\ \boldsymbol{\mu}^{2\tau t^m} \otimes \boldsymbol{\epsilon}^{2\tau t^m} \end{bmatrix} & \\ & \vdots & \\ & & \boldsymbol{\gamma}^{(n^m-1)\tau t^m + 1} \otimes \begin{bmatrix} \boldsymbol{\mu}^{(n^m-1)\tau t^m + 1} \otimes \boldsymbol{\epsilon}^{(n^m-1)\tau t^m + 1} \\ \boldsymbol{\mu}^{(n^m-1)\tau t^m + 2} \otimes \boldsymbol{\epsilon}^{(n^m-1)\tau t^m + 2} \\ \vdots \\ \boldsymbol{\mu}^s \otimes \boldsymbol{\epsilon}^s \end{bmatrix} \end{bmatrix} \tag{4.42}$$

4.4.2 Analysis Considerations

Steady State

Given that the introduction of a sampling period affects the what was described in 4.1.2 (and therefore also the state transitions), not everything described in 4.2 is still applicable. Namely, when $\tau > 1$, the states are guaranteed to have periodic behaviour and thus the chain is no longer ergodic (though it still is a unichain). That said, as explained in Section 3.1.2, this is not necessarily an issue as long as our analysis is simply based on the overall proportion of states. Given that we are only interested in the average behaviour of the system, this result is completely suitable for the metrics that we are analysing. This means that the introduction of periodic behaviours in our system is not sufficient to invalidate the results of our analysis framework.

Metrics

Although the metrics being gathered are not invalidated by the periodic behaviour of our Markov chain (as discussed above), the introduction of a sampling period does affect the definition of some metrics and therefore they may have to be calculated differently.

The loss function defined in Equation (4.21) assumes that there is new data to be received at every instant. This is no longer accurate given that, whenever $\tau > 1$, it would overestimate the amount of data being created and consequently also overestimate the losses. In fact, when $\tau > 1$, the result of that equation is equivalent to calculating the average of $\tau - 1$ times 100% loss and one time the actual loss we are trying to calculate; that is, if we have, for example, $\tau = 4$, under the previous equation we are guaranteed to have at least $\frac{3}{4}$ of lost data, plus whatever data is lost out of the remaining $\frac{1}{4}$. In other words, the probability being calculated would always be at least $\frac{\tau-1}{\tau}$. Therefore, to compensate for this, we can define this function as such:

$$loss = P\left[\bigwedge_{a=1}^{t-1}(X_{k+a} \notin \mathcal{D}_a)\right] \cdot \tau - (\tau - 1) \tag{4.43}$$

Moreover, the calculation of losses at the DCU on (4.22) assumed that entities could only receive data from the DCU in the instant that it was created. Given that the data may now stay on the DCU for longer than one instant, loss at the DCU it can be adapted in a similar way to the previous equation, as shown below:

$$loss\_at\_DCU = P\left[\bigwedge_{a=1}^{\tau}(X_{k+a} \notin \mathcal{C}_a)\right] \cdot \tau - (\tau - 1) \tag{4.44}$$

Finally, the calculation of the probability of delivering a single copy is also affected. Similarly to the situation described for the loss function, whenever $\tau > 1$ we are overestimating the amount of data that is possible to deliver and therefore underestimating the probability of delivery of a single copy. This can be adjusted by multiplying the final result at (4.31) by τ.

All the other metrics can be used without any changes, as either they are not affected by this change or their calculation depends solely on other metrics that have already been adjusted.

4.5 Implementation Aspects

Since the calculations required to analyse this model are far from trivial, and given that we were unable to find any tool or framework that could be easily adapted to our model, we needed to program our own solution. To do this, we opted to use Python 3 [62] as the programming language, particularly due to its inherent rapid prototyping and extensive support for scientific computation libraries.

Due to the way this model was designed, s (the size of $\mathcal{X}$) can increase significantly with a small increase in some parameters; in fact, increasing m (number of entities) results in an exponential increase of s (as seen in 4.1.1). As a consequence, even on a modestly sized scenario, $\mathcal{X}$ can grow to contain millions of elements, especially if more than 2 entities are being considered. On top of that, the number of elements in $\mathbf{P}$ grows quadratically with s (as described in 4.1.2). Therefore, our purpose-built implementation had to assume the necessity of having to deal with large numbers of states efficiently. Moreover, given that this model was designed to handle any choice of parameters, it had to be implemented in a way which allowed a user to make these changes without rewriting code; additionally, it had to allow the implementation of alternatives for g^x_{ib} and u^x_{ia} via some sort of standardized interface.

The Python scripts developed for this implementation can be accessed at `https://github.com/kYwzor/DTNRoutingMarkov`. In the following sections we will go over the main design choices that were taken to deal with the requirements and limitations described above.

4.5.1 Sparse Matrices

As mentioned before, $\mathbf{P}$ is likely to have an extremely large number of elements; in that case, it would be implausible to be able to keep all elements accessible in random-access memory (RAM) without running out of memory. Thankfully, for all g_{ib}^{x} and u_{ia}^{x} that have been described, each state is only able to transition to a maximum of 6^m other states. This means that, in general, the overwhelming majority of elements of $\mathbf{P}$ will be zeros; in fact, even for a relatively small scenario with $m = 2, s = 36000$, $\mathbf{P}$ is guaranteed to have a sparsity of over 99.9% and this value will intuitively increase with larger values of s. Therefore, instead of storing every element of this matrix, we could save a substantial amount of memory by only storing non-zero elements; this is the concept behind sparse matrices. Figure 4.1 shows an example of a sparsity plot for a scenario with 12500 states.

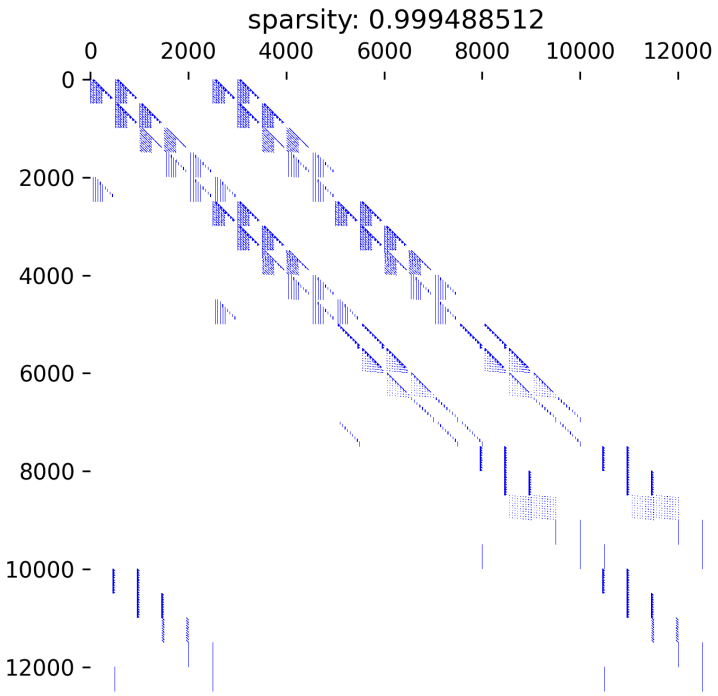

Figure 4.1: Sparsity plot for $\mathbf{P}$ of a scenario with $m = 2, n = 5, t = 10, c = 1, \tau = 5$, Forward Walk ($a^+ = 2/3$), Epidemic Routing ($r = 0.5$). Each point represents a possible transition between two states

While we use NumPy [18] to handle all dense arrays, throughout the program we use two different implementations of sparse matrices by SciPy [64]. Specifically, for all intermediate operations before the construction of $\mathbf{P}$, we are using a *coordinate list* format, since it is relatively efficient for constructing matrices and for calculating Kronecker products. In this format, the matrix is described by an array with the values of all elements, and two more arrays for their corresponding row and column indices, with these values being sorted first by row and then by column. After $\mathbf{P}$ is built, it is converted to a *compressed sparse*

row format which is used for all future operations (mainly matrix multiplications), since it is particularly efficient for them. While the *coordinate list* format has three arrays with a length equal to the number of non-zero elements, *compressed sparse row* manages to reduce the size of the array for the row indices, by creating an array with one element per row where it stores the number of non-zero values "seen" up until that row.

Sparse matrices have properties that allow some operations to be less computationally intensive than they would be for the equivalent dense matrix; for example, multiplying any number by 0 will always return 0 and thus a matrix multiplication algorithm made for sparse matrices can be designed to make use of these shortcuts. SciPy's implementation has multiple optimizations of this kind, and thus we can achieve significant speed ups by making calculations in matrix form whenever possible. This was the goal of expressing most calculations in a matrix form in previous chapters. However, given that the creation of sparse matrix objects is significantly more computationally intensive than the creation of dense arrays, we avoid creating sparse matrix objects in some intermediate calculations that are critical to the performance. We achieve this by directly creating and handling the three dense arrays that internally describe a sparse matrix (as explained previously), allowing us to only create the sparse matrix object after these intermediate steps have been concluded.

4.5.2 Parallelism

When analysing this model, many of the necessary calculations are made on a state-by-state basis, with each calculation being independent of the others; this mainly occurs while building $\mathbf{P}$, but also on the calculation of auxiliary sets for some metrics, for example, (4.20), (4.23), (4.30) and (4.32). Given the previously mentioned large number of states, this type of operations represent a significant computation time. However, given their independence from each other, these calculations can be made in parallel, as long as the system being used has the support for parallelism.

Specifically, we have implemented this through Python's multiprocessing module, which allows us to create a pool of worker processes (one for each available logical core) and then iterate the states by passing the correspondent jobs to the worker processes, as the workers become available. The construction of $\mathbf{P}$ is particularly interesting in that we are able to calculate it by creating a separate job for each different value of $\boldsymbol{\gamma}^x$ in (4.42); as for the previously mentioned auxiliary sets, they can be calculated prior to all metric calculations and then stored in a dense matrix format to be used as needed for each metric.

When using Python's multiprocessing module, communication between processes is typically handled through serialization/de-serialization of data (*pickling*). While this is adequate for small amounts of data, it may represent a significant slowdown when child processes need to access large shared resources. Given that Python creates these processes based on the program's global state, one way to bypass this communication is to create the necessary shared resources globally, prior to the creation of the worker pool. To achieve this, we use singleton-like patterns; while these are typically discouraged, we believe them to be better than the alternative (using global variables), as class attributes provide some degree of encapsulation.

4.5.3 Linear System Solver

The linear system seen in (3.7) can be expressed in matrix form, as described in a prior footnote. In our case, these matrices will be sparse and there are solving algorithms that were designed particularly for them. In general, there are two major types of solvers: solvers based on direct methods and solvers based on iterative methods. The first type in theory is able to calculate the exact solution; to do that it first transforms the matrix (LU decomposition) in order for it to get certain useful properties which then allow it to directly calculate the solution. However, due to the necessary floating-point arithmetic the results are not guaranteed to be exact in practice and this transformation process is not easily parallelized; moreover, for our particular matrices, it is common for the decomposition to significantly reduce the sparsity of the matrix, to the point where system memory is not sufficient. Due to all this, direct methods are not well suited for solving this particular type of linear system. On the other hand, iterative methods are well suited for large sparse matrices and they can usually be easily parallelized. These methods work by starting with some solution which is then improved iteratively until the requested precision is reached; therefore, this allows us to have a trade-off between computation time and precision. With all this in mind we decided to use iterative methods in our calculations, particularly, SciPy's implementation of the LGMRES algorithm with tolerances of 10^{-14}.

4.5.4 Performance

By optimizing many aspects of the implementation we were able to significantly decrease the overall runtime of our program. Our program has 3 main phases: calculation of the Transition Probability Matrix ($\mathbf{P}$), calculation of the stationary state probability vector ($\boldsymbol{\pi}$) and calculation of the performance metrics. While scenarios with more states generally increase the overall runtime, the runtimes of some phases are affected more than others depending on which specific parameter is being changed. For example, on Figure 4.2, we can compare two scenarios with the same number of states (400 000) but with different values of t and n (the sparsity is mostly the same). This plot clearly demonstrates that

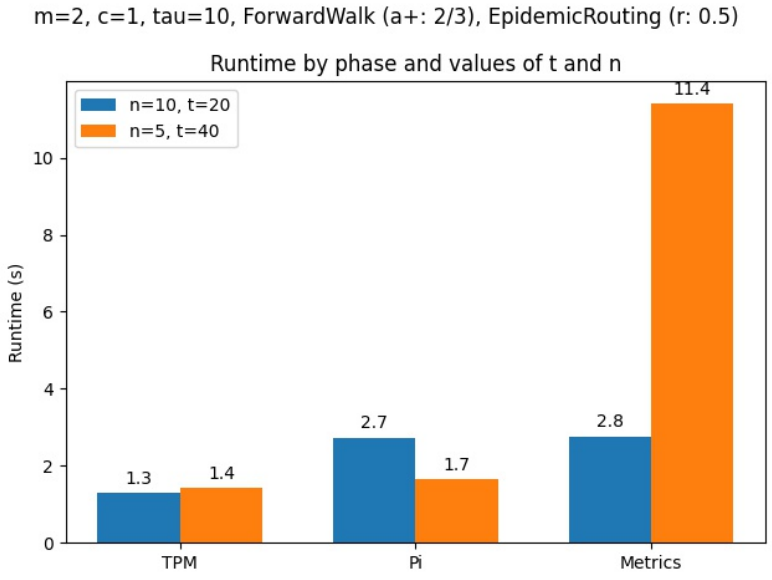

Figure 4.2: Comparison of the runtime of each phase when the values of t and n are changed but s stays the same (average over 5 runs).

the calculation of metrics is particularly affected by t. This is due to the fact that most metrics analyse the chain over the period that a packet of data may exist, which is t, therefore affecting the amount of computation needed; particularly, Equation (4.31) scales quadratically with t. On the other hand, the calculation of $\mathbf{P}$ seems mostly unaffected by these changes. As for the calculation of $\boldsymbol{\pi}$, since we did not implement the algorithm ourselves, we are not sure of what causes the decrease in runtime with a bigger t and smaller n, but it likely is related to the matrix's structure being easier to solve. In fact, due to its iterative nature, the runtime for the calculation of $\boldsymbol{\pi}$ seems to depend heavily on the specific scenario, making it hard to generalize its behaviour. Another interesting way to look at the runtimes is to see how they behave with a linear increase in the number of states. Particularly, we know that τ causes a linear increase in s and thus Figure 4.3 shows its effects. As this figure demonstrates, we believe that our calculations scale mostly

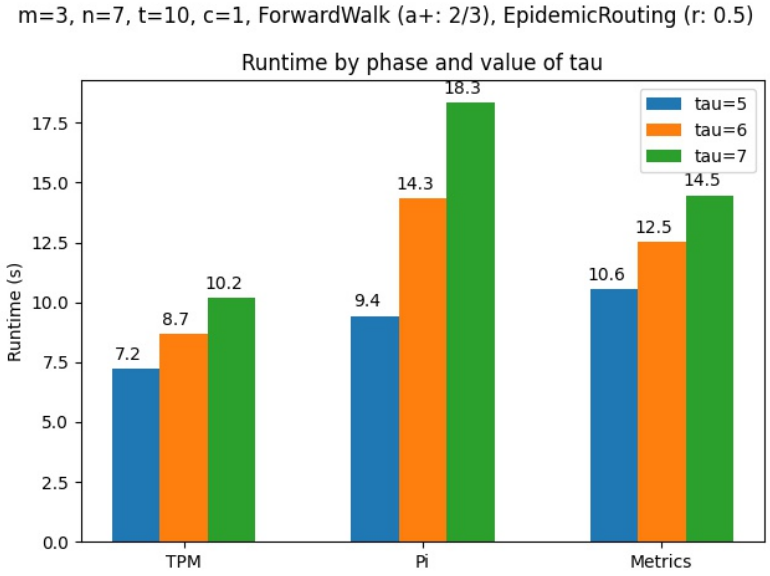

Figure 4.3: Comparison of the runtime of each phase when only τ is changed, representing a linear increase in s (average over 5 runs).

linearly with increases in the number of states (for similar scenarios), though the increase in computation time for $\boldsymbol{\pi}$ is significantly steeper.

Overall, we are satisfied with the performance we achieved, since currently the main limitation is the amount of memory required for the calculation of $\boldsymbol{\pi}$, which we did not implement. For example, we can calculate $\mathbf{P}$ for a scenario with 10 million states in just about 31 seconds, but the iterative algorithm completely exhausts all available memory before it manages to return $\boldsymbol{\pi}$. Therefore, we consider that at this point the biggest improvements to the performance of our program would come from a more careful choice of a system solver and respective tolerances; however, since this is not the focus of our research, we did not pursue it.

4.6 Verifying the Analysis

Given that the approach defined in Section 4.2 was not based on any existing framework, simply verifying that the metrics have been implemented as they have been defined is not sufficient; it is particularly important to verify that these definitions produce results that mean what we intend them to. Therefore, we made an effort to verify our calculations in

a few ways, which will be described in this section.

4.6.1 Simulation based on the Transition Probability Matrix

Firstly, we wanted to verify our calculations with the assumption that the Transition Probability Matrix was correctly defined in (4.42). To do that, we built a simulator which aimed to create a long sequence of states, resulting from successive state transitions. Specifically, the simulation starts in a random state. Then, on every simulation step, we generate a random number between 0 and 1 and start a cumulative sum along the row vector of $\mathbf{P}$ correspondent to the current state. When this cumulative sum surpasses the randomly generated number, the column in which it did corresponds to the state that the simulation will now transition to. The simulation stops after a predefined number of steps.

When this simulation is finished, we can analyse the sequence that was generated and extract metrics from it. Specifically, we can count the occurrences of each state and divide them by the total number of steps, and we expect these results to be a good estimate for the corresponding elements of the calculated $\boldsymbol{\pi}$. Given that Equation (4.35) depends solely on $\boldsymbol{\pi}$, this is enough to verify that calculation. The rest of the calculations depend on analysing the probability of certain sequences of states. To verify them, we can simply calculate how often these sub-sequences occur, by analysing the generated sequence with a sliding window of the appropriate size. Once again, we expect that these results will be good estimates for the equivalent calculations.

Increasing the number of simulations steps has a tendency to approximate the simulated results to the calculated results, and larger scenarios require longer simulations for the same precision. From our testing, with a million simulation steps, the results for a scenario with 7200 states matched with the calculations up to the third decimal place. This gives us confidence that the calculations are defined correctly as long as the transition probability matrix is correct.

4.6.2 Simulation based on Entity Behaviour

We were also interested in verifying our results with the least amount of assumptions regarding our model. To achieve this, we designed a simulation that aimed to describe the exact same behaviour of our model but without any notions of Markov chains, states or transitions. Instead, the simulation is operated from the perspective of the entities, with each moving and communicating independently (although synchronously). Each entity has a certain location and certain data. At the start of the simulation, each entity is placed in a random position without any data. At each simulation step, the age of the data being carried by entities is incremented, then the DCU contacts any entity in range, and then each entity tries to contact the RSU, if it is in range, or all entities in range, otherwise. Every time an entity is contacted, it only takes that contact into consideration if the contacting entity has the newest data it has seen in that step. After all these contacts occur, each entity compares the newest data from contacts with its current data and one of three situations occur: 1) If there is no data from contacts or the data from contacts is not newer than the one it is carrying, the entity keeps its current data; 2) Else, if it is carrying no data, it takes the newest data from contacts; 3) Else, it generates a random number between 0 and 1, taking the newest data from contacts if the generated number is smaller than r, and keeping its current data otherwise. After these comparisons are finished, each entity moves to a new position by randomly choosing between moving forwards, backwards or not moving (the probability of each depends on the mobility model) and a simulation

step is considered to be finished. The simulation runs for a predefined number of steps.

While this simulation is very distant from our model in the ways that it processes changes in the simulation state, its overall behaviour should be exactly the same, as the entities move and communicate with the exact same rules as described in our model. Therefore, this simulation is a great tool for the verification of our model. Particularly, while the simulation is running, it is keeping track of relevant data such as the age of data in all entities, occurrences of first-hits (delivery of a certain data for the first time), lost data, amount of deliveries and repeated deliveries. When the simulation finishes, we can divide these counters by the number of simulation steps, allowing us to calculate all the metrics that we have defined.

Once again, we expected these results to be good estimates for the ones that can be calculated by our model. From our testing, with a million simulation steps, the results for a scenario with 7200 states matched with the calculations up to the second decimal place. This was a very satisfying result given the significant difference in the approach.

4.7 Parameter Sensitivity

Before moving on to analyse a real-world scenario, it is important to understand how the model behaves with changes to its parameters. Therefore, we will explore artificial scenarios were we change some of the parameters and analyse the corresponding results. This allows us to better understand the model, and it will help creating an accurate model for a real scenario and interpreting future results. Additionally, it can help us verify that the changes in the parameters alter the performance of the network in a way that is intuitively expected. We will mostly study the scenarios from the standpoint of minimizing data loss, though obviously there are other possible approaches, such as minimizing delays or the utilization of the network. Every scenario will assume a Forward Walk with $a^+ = 0.5$ and Epidemic Routing.

4.7.1 Number of entities

We will start by comparing the outcome of data in two scenarios that are equal with the exception of the number of entities. We varied the replacing ratio and created a plot of data outcomes for each scenario, which is shown in Figure 4.4. An obvious result is that increasing the number of entities gives the network a better chance of delivering data. More interestingly though, this effect gets much more pronounced with higher values of r. Not only that, but the value of r that minimizes overall data loss is also higher for the scenario with 3 entities. This is due to the fact that higher values of r mean that entities share their data more frequently, and thus the increased number of entities becomes more relevant. However, a clear downside of having more entities is that it disproportionately increases the amount of data that is delivered more than once. This is due to the fact that it is now possible to have more copies of data in the network.

4.7.2 Time-to-live

We will now consider the scenario with 2 entities as our base scenario, on top of which we will change other parameters. In this case, instead of $t = 10$, we will now analyse the cases for $t = 4$ and $t = 16$, the results of which are shown in Figure 4.5.

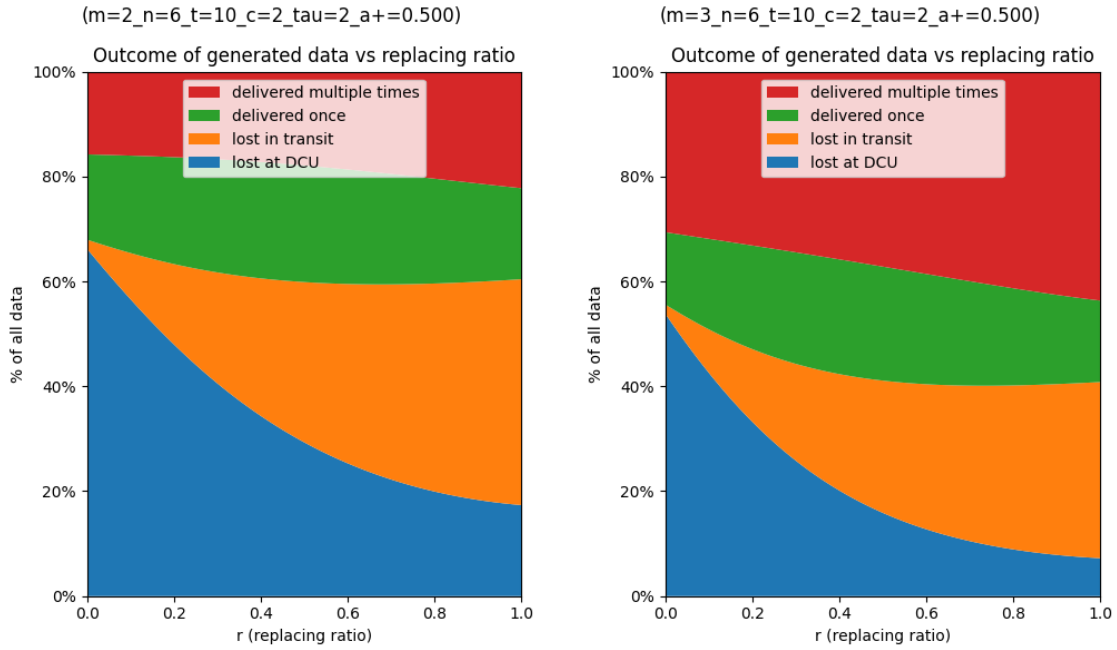

Figure 4.4: Both plots show the outcome of all data that is generated, divided in two types of loss and two types of deliveries. The plot on the left shows the results for $m = 2$, while the one on the right does the same for $m = 3$.

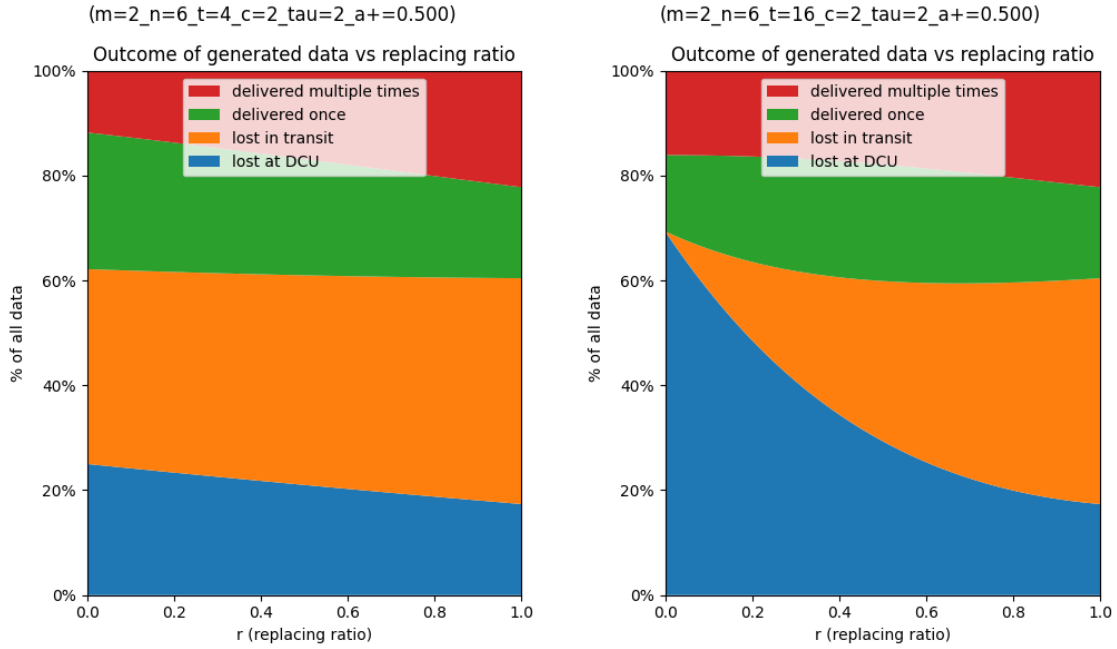

Figure 4.5: Both plots show the outcome of all data that is generated, divided in two types of loss and two types of deliveries. The plot on the left shows the results for $t = 4$, while the one on the right does the same for $t = 16$.

When $t = 4$, it is fairly hard for a single entity to carry data all the way from the DCU to the RSU. Due to this, the importance of sharing data increases, resulting in the optimal r for decreasing data loss being the maximum value possible ($r = 1$). However, given that the value of r is only relevant when entities are carrying data, and given that reducing t also reduces the overall probability of entities carrying data (since it is more likely to expire), when t is smaller the choice of r also has a smaller influence in the overall network performance. This results in the scenario with the smaller t having almost no change to the overall loss when t is changed. Interestingly though, for low values of r, the network has a better performance for the $t = 4$ scenario than for $t = 16$. This result is not very intuitive, but it can be easily explained by analysing the plot. The reduction of t directly results in an increase to the average number of entities that are available to receive data from the DCU (as their data expires more easily), which in turn significantly reduces the losses that occur at the DCU when compared to the scenario with larger t. Although this also results in significantly more losses in transit, the reduction in losses at the DCU is large enough that it is overall an advantageous trade-off.

Looking at the case with $t = 16$, we can make a few interesting conclusions. Firstly, when comparing it to the base scenario ($t = 10$), we notice that their plots are very similar despite the significant change in the parameters. In fact, the results for $t = 16$ are slightly worse for lower values of r, leveling out to be indistinguishable from $t = 10$ with higher values. This can be explained by the fact that most deliveries in the original scenario were already occurring way before data reached its maximum age, due to the relatively small distance the entities had to cover, which can be observed in Figure 4.6. This means that increasing t can only reduce the losses on a network up until a certain point, after which it can actually cause more losses or simply have a negligible effect.

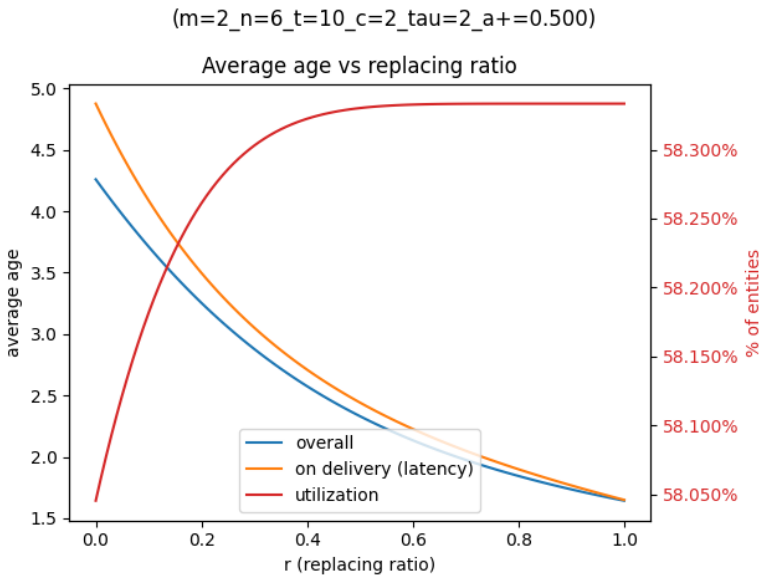

Figure 4.6: This plot shows, for the base scenario, the average age at delivery (latency) and in the network (overall), along with the utilization as a percentage of entities carrying data (different y-axis). The average age on delivery is smaller than 5 at any value of r.

Secondly, its minimum point for data loss (which occurs at $r = 0.68$) is slightly lower than the the minimum point for the $t = 4$ scenario. This is an intuitive result, since we expect that, under the right conditions, a bigger t allows us to deliver more data. Specifically, the data loss for $t = 16$ is lower than the $t = 4$ scenario for all $0.33 < r < 1$, with $r = 1$ resulting in approximately the same loss for both of them; this is due to the fact that,

when $r = 1$, we always replace the data that entities currently have and thus having a larger t is mostly meaningless as data is very likely to be replaced before it expires.

4.7.3 Sampling Period

If we reduce or increase τ by 1, the behaviour of the network changes significantly. This can be seen in Figure 4.7. The results for $\tau = 1$ show very clearly why we opted to introduce a

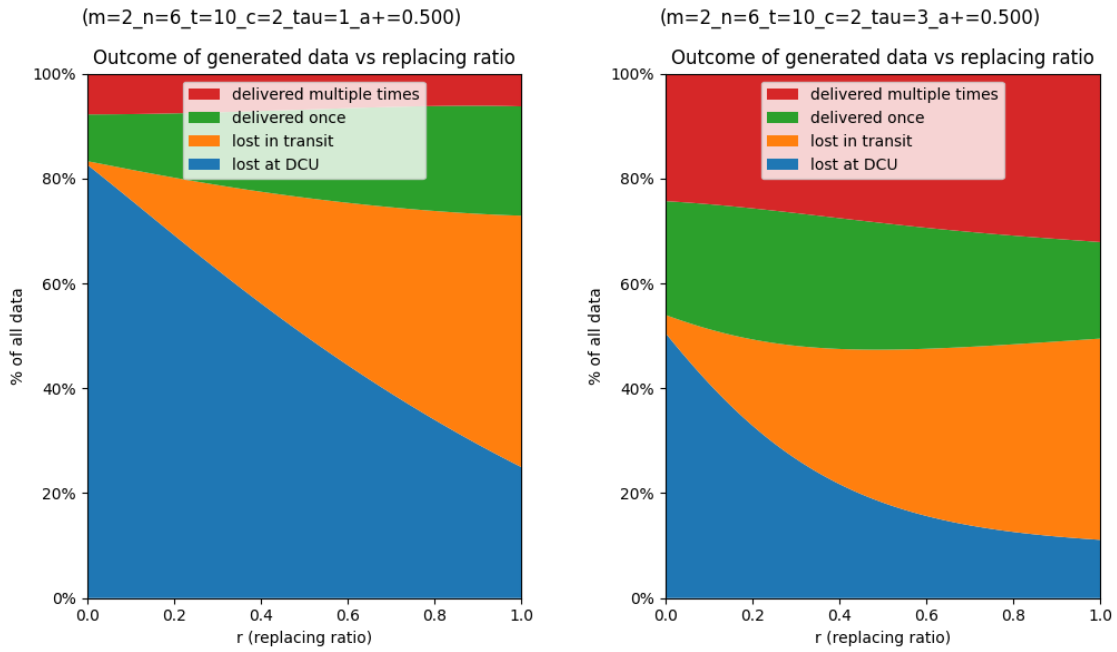

Figure 4.7: Both plots show the outcome of all data that is generated, divided in two types of loss and two types of deliveries. The plot on the left shows the results for $\tau = 1$, while the one on the right does the same for $\tau = 3$.

sampling period in our model; new data is created simply too frequently when compared to how often entities move. Due to this, the loss is extremely high regardless of r, making it hard to create any interesting scenario. As for $\tau = 3$, when compared to the base scenario ($\tau = 2$), the biggest difference lies in a significant reduction in data lost at the DCU, which is intuitive given that it is producing less data. This also shifts the optimal values of r towards 0, because, by increasing the time a certain data stays at the DCU, entities can afford to refuse it more frequently.

4.7.4 Discretization

As previously mentioned in Section 4.4, we can have different levels of discretization for this model, for both space and time. Specifically, we can increase the spatial resolution by increasing the number of locations n and proportionately increasing the contact range c and the speed of the entities in the mobility model. Similarly, we can increase the temporal resolution by increasing the sampling period τ and proportionately increasing the data's time-to-live t and decreasing the speed of the entities in the mobility model (since their position is now being updated more frequently). While the discretization of a real-world scenario always reduces the realism of the model, it is intuitive to think that bigger spatial and temporal resolutions can help approximate the model to the behaviour of the network

in the real-world. Therefore, if we keep increasing the resolution of our model we expect the results to change significantly (but not completely) at first, but we also expect the effects of successive increases to be less notable as they go on.

In order to observe this, we created a small scenario and then we increased its spatial and temporal resolution by 2, 3 and 4 times. It is important to note that if we wanted to double the spatial resolution we would need to halve the length of the intervals between locations, or in other words, double the number of intervals. Given that the number of intervals is $n-1$, this means that multiplying the resolution by x is achieved by using a new value of n such that $new\_n = (old\_n - 1)x + 1$. Moreover, if we increase the spatial and temporal resolution simultaneously, the speed of the entities in the mobility model must be kept unchanged.

The result of this exploration can be seen in Figure 4.8. As we expected, changing the

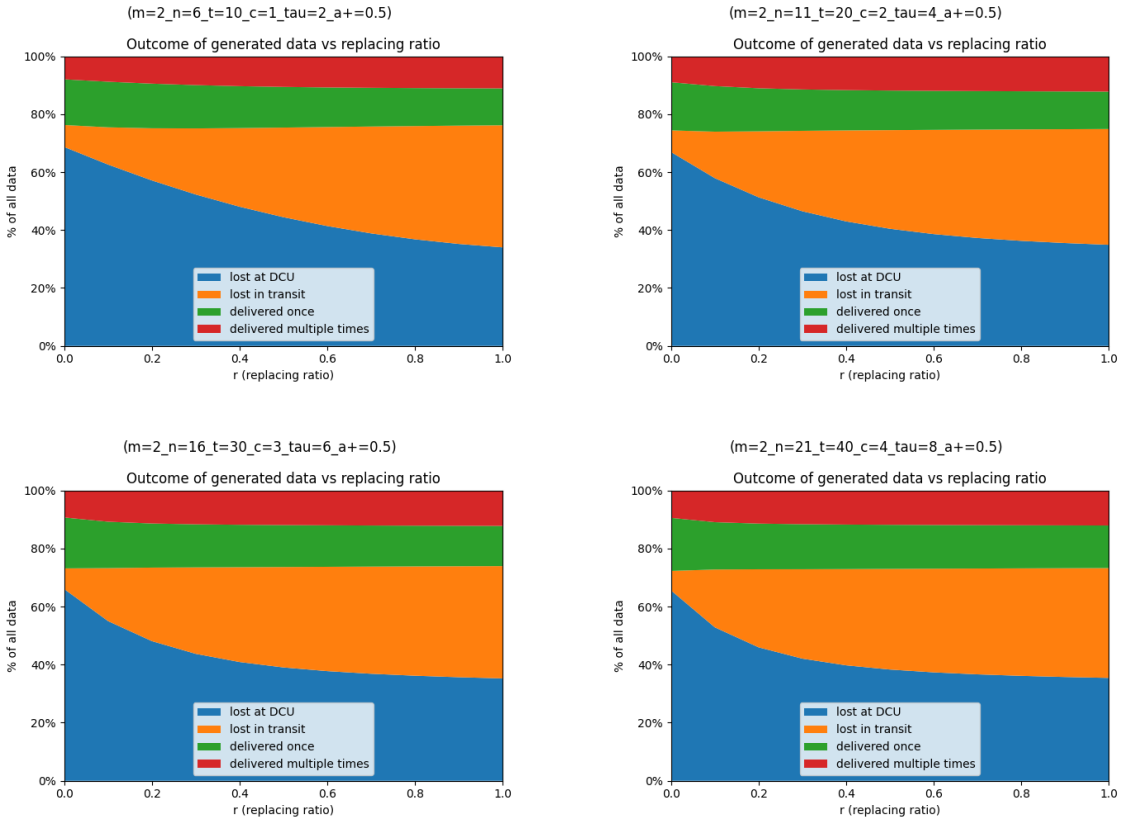

Figure 4.8: These 4 plots show the same scenario, but with increased spatial and temporal resolution. The resolution increases first from left to right and then from top to bottom.

resolution of the model does not change the results completely, which we believe to be a good sign that the interactions between the different parts of this model make sense. Moreover, as we predicted, doubling the base resolution achieved somewhat notable changes in the results, but the changes are less notable with each increase in resolution. We believe that this indicates that the results tend towards a limit, as the resolution increases, but we cannot further verify this experimentally due to model size constraints.

Chapter 5

Analysis of a Real-world Scenario

In this chapter we will apply the model defined in Chapter 4 to a simplified real-world scenario. This will allow us to demonstrate how this conversion can be achieved (Section 5.1) and how the resulting model can be analysed. Particularly, in Section 5.2 we will visualize the behaviour of the model from multiple perspectives and draw conclusions from it.

5.1 Modelling Process

We are interested in providing an example of the process of modelling a system starting from a real-world-like scenario. Let's consider we have the following scenario:

- We have a one-way street in which cars travel at an average speed of 60km/h and are able to overtake each other;
- The DCU and RSU are placed on this street, 500 meters apart from each other, with the cars moving from the DCU to the RSU;
- On average, there are 2 cars moving within these 500 meters;
- The DCU samples its sensors every 20 seconds and the data it generates is only relevant for 40 seconds.
- Communication can occur at a distance of up to 100 meters.

Since our model is only able to deal with discrete time and distance, both will have to discretized. Ideally we would use extremely small discrete units, since by decreasing the scale of the units being used we approximate the system to a continuous behaviour, which is better adjusted to a real-world scenario. However, a decrease in scale also results in a increase in the number of states, which may make the calculations intractable due to the time and memory it requires. Therefore the discretization should find a balance between these situations. With that in mind, we chose 50 meters and 2 seconds as our units of distance and time, respectively.

Given that we will only consider a 500 meter stretch of the road, this means we have 10 different intervals of 50 meters and therefore we have 11 different locations ($n = 11$). Moreover, the entities have a communication range of 2 intervals ($c = 2$). The DCU generates data every 10 units of time ($\tau = 10$), which has a time-to-live of 20 units of time

($t = 20$). Given that the cars only travel in one direction, we will model their movement using the Forwards Walk model. A car travelling at a velocity of 60km/h, covers $50 \times \frac{2}{3}$ meters in 2 seconds, therefore $\alpha^+ = \frac{2}{3}$. Finally, we will consider the average number of cars ($m = 2$). All things considered, this model is composed of 484000 different states.

5.2 Analysis

Let us say that, under this scenario, we decide to use a communication model and routing strategy like the ones described by Equation (4.2). This equation allows us to adjust r according to our needs and therefore we are interested in knowing what is the "best" possible value for it. Ideally, we would like to find a value of r that minimizes data loss and delivery latency while not wasting a lot of resources, that is, also minimizing storage utilization and deliveries of two or more copies of the same data. Intuitively, these goals seem to be hard to reconcile, but some values of r may be more interesting than others. Therefore, we decided to test this network for 101 equally spaced values of r, between 0 and 1. This will allow us to understand how different aspects of the network behave as r varies and ultimately make a decision on what r to pick.

We will start by creating a plot showing the probabilities of delivering different numbers of copies of the same data, as shown in Figure 5.1. By analysing these plots, it is clear that

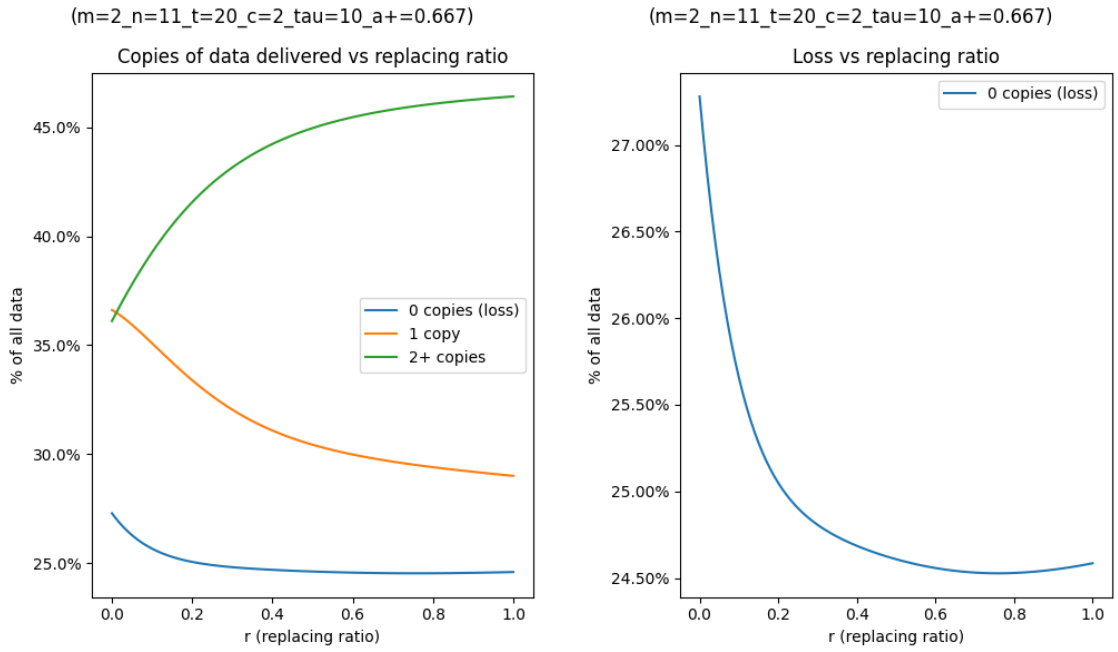

Figure 5.1: Both plots show how data delivery changes with different replacing ratios, but the plot on the right side focuses solely on loss to better highlight its curve.

out of all tested values of r, 0.76 is the optimal choice for reducing the data loss. However, the number of repeated deliveries strictly increases with the increase of r.

We can now try to understand what is causing these changes in the probability of data loss. To do that, we can create a plot distinguishing losses at the DCU from losses in transit, allowing us to evaluate what is causing most losses. Moreover, we can join this data with the data from the previous plots to create a stacked plot which shows a general overview of the outcome of data generated, separated by type of loss and type of delivery. Both of

these plots can be seen in Figure 5.2. As we can see, increasing the replacing ratio results

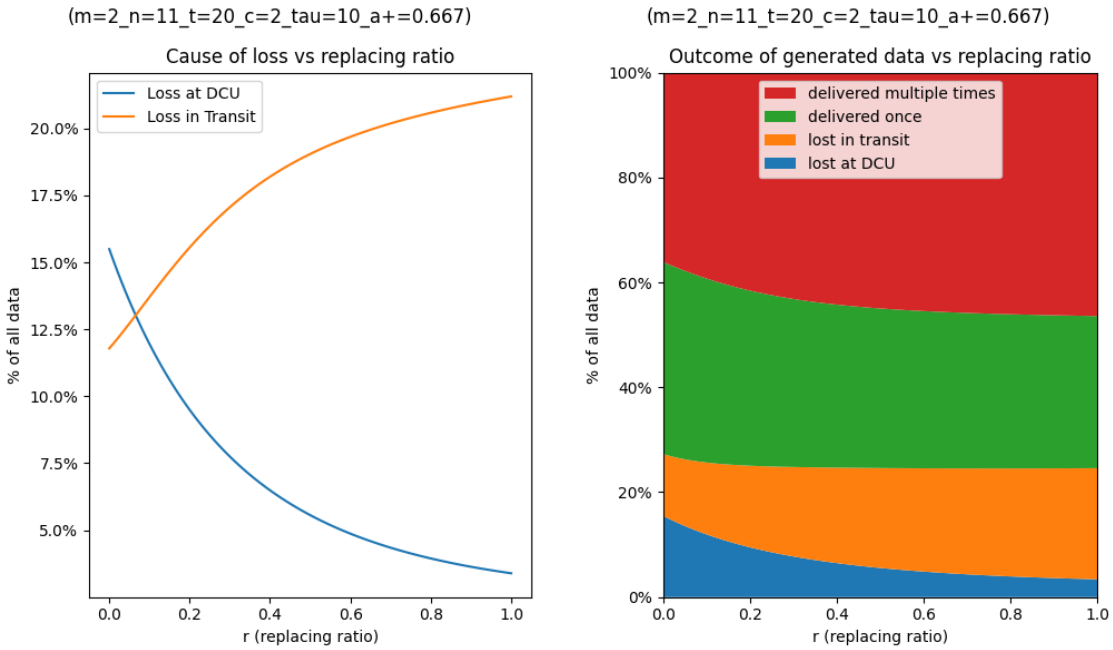

Figure 5.2: The plot on the left shows how the causes of loss change with r, while the plot on the right shows the outcome of all data that is generated, divided in two types of loss and two types of deliveries.

in lower losses at the DCU, which is an intuitive result as it allows the DCU to have more available entities nearby. However, this decrease comes at the cost of an increase in losses in transit, as entities are more likely to lose a packet along the way. Therefore, in this case, the optimal choice for reducing data loss is essentially a point where we are able to decrease the loss at the DCU, while not increasing the loss in transit as much.

Let us now look at the behaviour of the model for other metrics. It is interesting to analyse how r influences the age of the data in the network, as well as the average percentage of entities carrying data (utilization). We can display these values on a plot, shown in Figure 5.3. This plot shows that an increase in r not only strictly decreases the overall age of data in the network, but also specifically of the data being delivered. In fact, these two values seem to be fairly correlated, which we believe to be an intuitive result. However, a clear downside is that the utilization of storage resources rises significantly.

These results are in line with our expectations that it would not be easy to reconcile our multiple interests. However, it now becomes clear that we need to study the problem as a multi-objective optimization problem. We will specifically be looking at data loss, delivery latency and repeated deliveries. We have already established that this problem is non-trivial, that is, there is no value of r that simultaneously optimizes all the metrics we are considering. Therefore, it becomes especially interesting to visualize the *Pareto front* of our problem. This front is the set of all nondominated solutions, that is, solutions to which there is no alternative which improves the value of one of our metrics (objectives) without degrading the value of another metric. In Figure 5.4, we show our objectives in pairs and then simultaneously in a 3D representation. As a reminder, we want to minimize all of our three metrics. Therefore, when analysing loss versus latency, we notice that for any $r < 0.76$ the solutions obtained have larger latency and loss than the solution obtained with $r = 0.76$ and thus they are clearly not interesting for these objectives. On the other hand, solutions obtained for $0.76 \leq r \leq 1$ improve the value of one of our objectives while

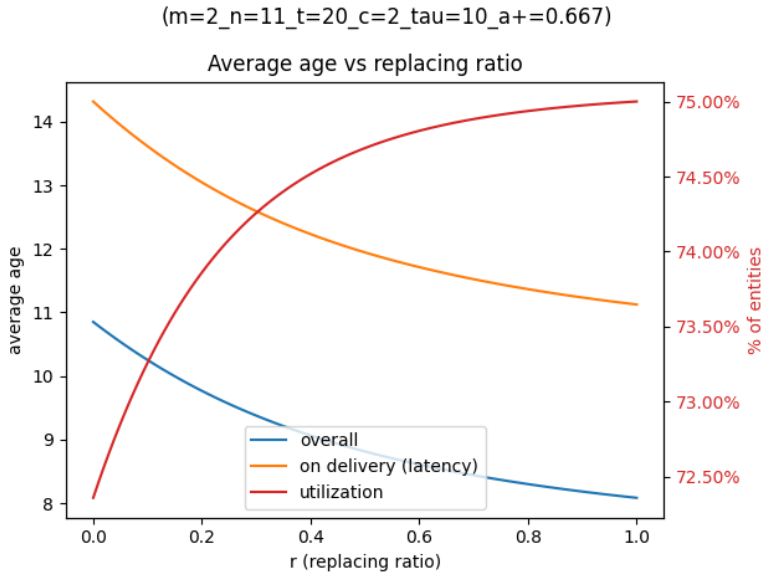

Figure 5.3: This plot shows the average age at delivery (latency) and in the network (overall), along with the utilization as a percentage of entities carrying data (different y-axis).

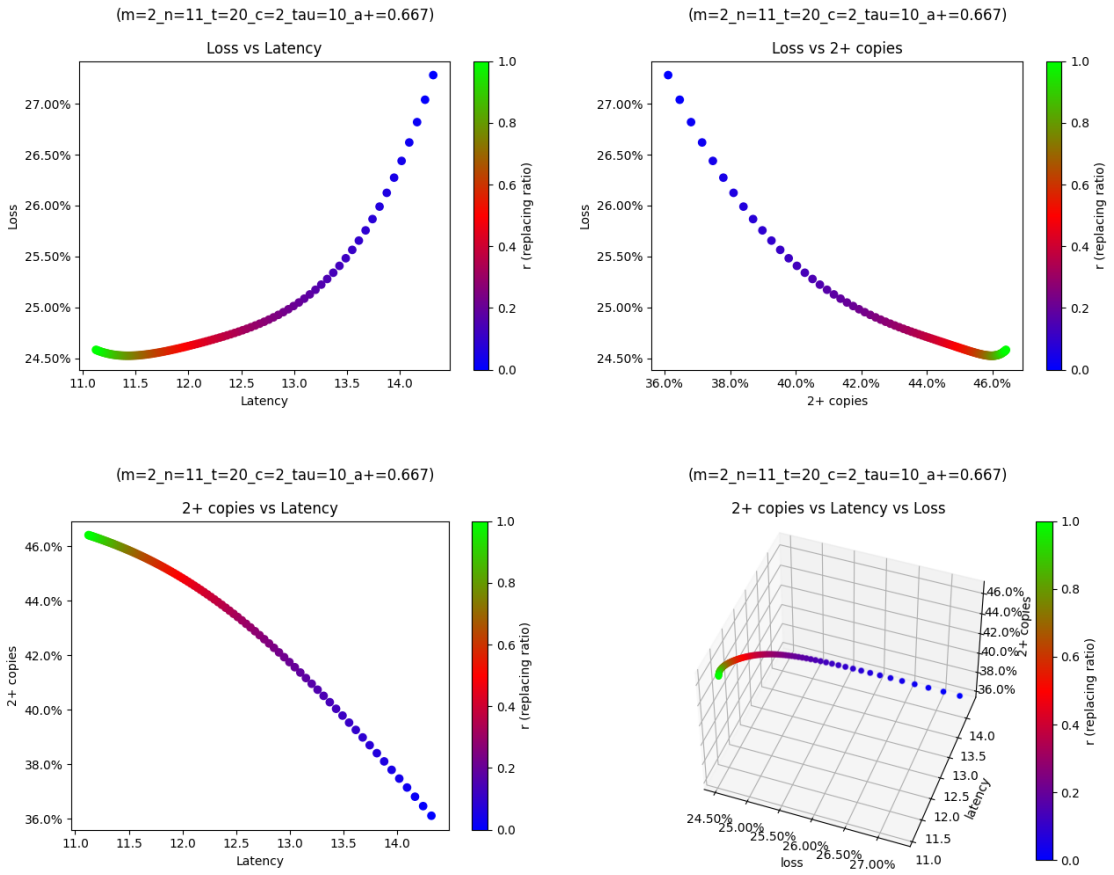

Figure 5.4: The first three plots represent the pairwise Pareto fronts between our optimization goals, while the last plot (lower right) is a 3D representation of the Pareto front between these goals.

degrading the value of the other, and thus these solutions are nondominated. Similarly, when taking into consideration loss and repeated deliveries, the solutions obtained for $0 \leq r \leq 0.76$ constitute the Pareto front. As for the last pair-wise comparison, all solutions are nondominated. This means that, when considering all of our three objectives, all solutions are nondominated.

The biggest challenge in this optimization has now become clear: in this scenario it is impossible to lower latency without causing an increase in the deliveries of repeated copies. Therefore, any value of r is a potentially good choice depending on our goals and, consequently, the decision falls squarely on the decision maker's shoulders.

For example, upon analysing all this data, the decision maker may conclude that, when it comes to latency, it is only fundamental to keep it below a average of 12, but further decreases are not relevant. In that case, we would now only be interested in analysing the relation between data loss and repeated deliveries, but the solution space would be reduced, as the latency became a constraint on the possible solutions; this is represented in Figure 5.5. With this, the decision maker has reduced the set of nondominated solutions

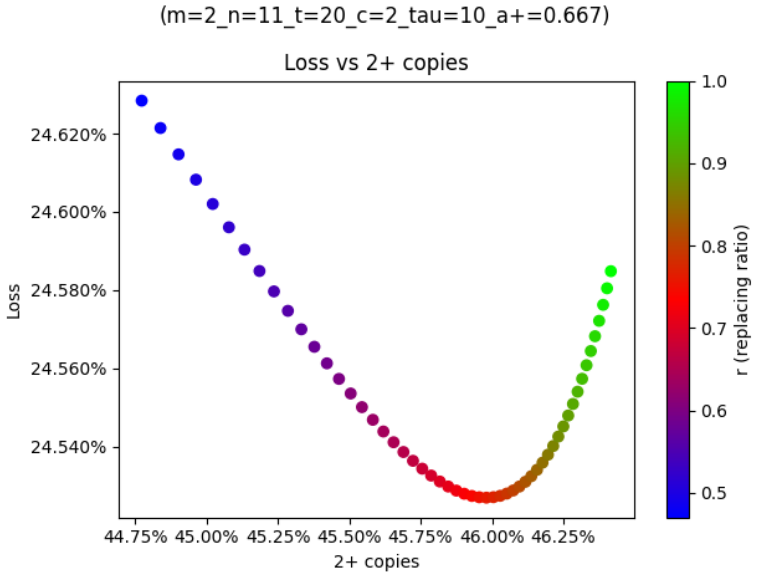

Figure 5.5: This plot compares Loss and probability of repeating deliveries, with the restriction of keeping latency under 12.

to values of r between 0.47 (minimum value of r that keeps the average latency below 12) and 0.76 (optimal value of r for minimum loss), with any choice between these values being a simple trade-off between our two objectives.

Chapter 6

Conclusion

The mobility of nodes in a network provides both challenges and opportunities for communication. While traditional networking approaches struggle to deal with the resulting disruptions in connectivity, DTN approaches are designed to resist them and also adopt SCF paradigms which allow the performance of the network to benefit from the mobility. In this context, DTNs open the possibility of establishing communication in scenarios with little to no infrastructure, enabling a wide range of new applications.

While interesting, DTNs also come with challenges of their own, having particular difficulty in providing guarantees about data delivery. On top of that, they are complex to study and predict, given that their performance depends heavily on the interactions between the way nodes move, communicate and route messages. At the same time, such analysis is crucial to allow the implementation of real-world systems that are well-optimized, and thus there is a significant interest in exploring ways of assisting it.

We proposed a model based on a finite-state discrete-time homogeneous Markov chain that aims to describe the behaviour of a DTN in a urban sensing scenario. This model separately describes a mobility model, a communication model and a routing strategy, which together form the behaviour of the network, allowing us to easily change each of these independent components. Although the model simplifies some aspects of a real-world scenario, we have shown that we can still extract many relevant network performance metrics from it, such as delivery ratios and delays, which can help us make decisions regarding optimal routing strategies. We have also implemented a program that handles the necessary calculations for the analysis of this model, carefully designing it so that its computation is viable. On top of that, we verified the proposed calculations by comparing our results with the results of two different simulation-based models.

We then applied the proposed model to a real-world scenario and extracted all the previously described metrics, visualizing them in a way that facilitates the analysis. The model proved to be robust enough to provide results that are interesting from an optimization perspective (non-trivial). We approached the results from a multi-objective optimization standpoint, exemplifying how a decision maker might decide on a optimal parameter for the routing strategy.

We believe our work provides a good foundation for future research in this topic, as the model was designed in a way that can easily accommodate extensions, allowing it to have increased realism and adaptability to different scenarios.

Future Work

One of the biggest limitations of our model is that it requires an extremely large number of states to properly analyse a non-trivial real-world scenario. One possible way of avoiding this would be by changing from a mobility model based on discrete values for locations, to a continuous model that is based on the estimated time between entity contacts. This could potentially reduce the amount of information that is needed to describe each entity and, consequently, significantly reduce the overall number of discrete states.

Another limitation of our model is that we consider the existence of a single DCU and a single RSU. In practice, roads may contain several of each, and may even contain some other units that behave as throwboxes. This would greatly increase the complexity of the system being studied, but also its realism.

A future work opportunity that would be worth exploring is the possibility of creating mobility models and routing strategies with increased realism. For example, a second-order Markov chain could allow for interesting properties such as inertia in entity movement and knowledge of an entity's current direction, which could allow for more realistic movement and better routing decisions. Another possibility is to base a routing strategy on an entity's knowledge of its current location. Moreover, our current mobility models are based on a one-dimensional space, but a two-dimensional (or even three-dimensional) space would allow for an increase in degrees of freedom of movement, which has the potential to increase realism. The system could also be expanded to consider the possibility of all units having larger buffers, though that would significantly increase the state space and require a change in the current routing strategies.

Finally, it would be extremely interesting to build these scenarios in the real-world and gather data for the relevant network metrics. This would allow us to properly validate the results of our model and have a better grasp on how closely it can describe reality.

References

[1] Ian Akyildiz, Dario Pompili, and Tommaso Melodia. Underwater acoustic sensor networks: Research challenges. *Ad Hoc Networks*, 3:257–279, 05 2005.

[2] Zigbee Alliance. Smart energy. `https://zigbeealliance.org/solution/smart-energy/`. Accessed: 2020-01-18.

[3] B. Baron, P. Spathis, M. Dias de Amorim, Y. Viniotis, and M. H. Ammar. Mobility as an alternative communication channel: A survey. *IEEE Communications Surveys Tutorials*, 21(1):289–314, Firstquarter 2019.

[4] A. Bujari, S. Gaito, D. Maggiorini, C. E. Palazzi, and C. Quadri. Delay tolerant networking over the metropolitan public transportation. *Mobile Information Systems*, 2016:8434109, 09 2016.

[5] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–11, April 2006.

[6] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: sensor networks in agricultural production. *IEEE Pervasive Computing*, 3(1):38–45, Jan 2004.

[7] Christos G. Cassandras and Stéphane Lafortune. Markov chains. In *Introduction to Discrete Event Systems*, chapter 7, pages 369–428. Springer US, Boston, MA, 2 edition, 2008.

[8] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838, RFC Editor, April 2007.

[9] Google Cloud. Data transfer appliance. `https://cloud.google.com/transfer-appliance/`. Accessed: 2020-01-18.

[10] James A Davis, Andrew H Fagg, and Brian Neil Levine. Wearable computers as packet transport mechanisms in highly-partitioned ad-hoc networks. In *Proceedings Fifth International Symposium on Wearable Computers*, pages 141–148. IEEE, 2001.

[11] M. Delorme. An introduction to cellular automata. In *Cellular Automata: A Parallel Model*, pages 5–49. Springer Netherlands, Dordrecht, 1999.

[12] 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*, volume 33 of *SIGCOMM '03*, page 27–34, New York, NY, USA, 12 2003. Association for Computing Machinery.

[13] Kevin R Fall. Comparing information-centric and delay-tolerant networking. In *LCN*, 2012.

[14] Robert G. Gallager. Finite state markov chains. In *Discrete Stochastic Processes*, pages 103–147. Springer US, Boston, MA, 1996.

[15] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40–54, 2011.

[16] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., USA, 1994.

[17] Khaled A. Harras, Kevin C. Almeroth, and Elizabeth M. Belding-Royer. Delay tolerant mobile networks (dtmns): Controlled flooding in sparse mobile networks. In Raouf Boutaba, Kevin Almeroth, Ramon Puigjaner, Sherman Shen, and James P. Black, editors, *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, pages 1180–1192, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[18] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[19] S. Hayat, E. Yanmaz, and R. Muzaffar. Survey on unmanned aerial vehicle networks for civil applications: A communications viewpoint. *IEEE Communications Surveys Tutorials*, 18(4):2624–2661, Fourthquarter 2016.

[20] C. Hedrick. Routing Information Protocol. RFC 1058, RFC Editor, June 1988.

[21] Sean Hollister. Google announces nest wifi, a mesh router system with smart speakers inside. *The Verge*, October 2019. Accessed: 2020-01-18.

[22] Sushant Jain, Kevin Fall, and Rabin Patra. Routing in a delay tolerant network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, page 145–158, New York, NY, USA, 2004. Association for Computing Machinery.

[23] Sweta Jain. Buffer management in delay-tolerant networks. In Khaleel Ahmad, Nur Izura Udzir, and Ganesh Chandra Deka, editors, *Opportunistic Networks: Mobility Models, Protocols, Security, and Privacy*, chapter 3. CRC Press, 2018.

[24] David Johnson and David Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Comput.*, 353, 05 1999.

[25] Evan Jones and Paul Ward. Routing strategies for delay-tolerant networks. *Submitted to ACM Computer Communication Review (CCR)*, 01 2006.

[26] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, page 96–107, New York, NY, USA, 2002. Association for Computing Machinery.

[27] Tomotaka Kimura, Takahiro Matsuda, and Tetsuya Takine. Location-aware store-carry-forward routing based on node density estimation. *IEICE Transactions on Communications*, 98(1):99–106, 2015.

[28] Tomotaka Kimura and Masahiro Muraguchi. Buffer management policy based on message rarity for store-carry-forward routing. In *2017 23rd Asia-Pacific Conference on Communications (APCC)*, pages 1–6. IEEE, 2017.

[29] I. Leontiadis and C. Mascolo. Geopps: Geographical opportunistic routing for vehicular networks. In *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, June 2007.

[30] Lexico. Sneakernet. `https://www.lexico.com/definition/sneakernet`. Accessed: 2020-01-18.

[31] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. In Petre Dini, Pascal Lorenz, and José Neuman de Souza, editors, *Service Assurance with Partial and Intermittent Resources*, pages 239–254, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[32] Jiajia Liu, Xiaohong Jiang, Hiroki Nishiyama, and Nei Kato. A general model for store-carry-forward routing schemes with multicast in delay tolerant networks. In *2011 6th International ICST Conference on Communications and Networking in China (CHINACOM)*, pages 494–500. IEEE, 2011.

[33] Sven Maerivoet and Bart De Moor. Cellular automata models of road traffic. *Physics Reports*, 419(1):1 – 64, 2005.

[34] J. McQuillan, I. Richer, and E. Rosen. The new routing algorithm for the arpanet. *IEEE Transactions on Communications*, 28(5):711–719, May 1980.

[35] Microsoft. Azure data box. `https://azure.microsoft.com/en-us/services/databox/`. Accessed: 2020-01-18.

[36] NASA. Disruption tolerant networking. `https://www.nasa.gov/content/dtn`. Accessed: 2020-01-18.

[37] Yasuharu Ohta, Tomoyuki Ohta, Eitaro Kohno, and Yoshiaki Kakuda. A store-carry-forward-based data transfer scheme using positions and moving direction of vehicles for vanets. In *2011 Tenth International Symposium on Autonomous Decentralized Systems*, pages 131–138. IEEE, 2011.

[38] Jim Partan, Jim Kurose, and Brian Neil Levine. A survey of practical issues in underwater networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(4):23–33, October 2007.

[39] Karl Pearson. The Problem of the Random Walk. *Nature*, 72(1865):294–294, July 1905.

[40] A. Pentland, R. Fletcher, and A. Hasson. Daknet: rethinking connectivity in developing nations. *Computer*, 37(1):78–83, Jan 2004.

[41] Agoston Petz, Justin Enderle, and Christine Julien. A framework for evaluating dtn mobility models. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, Simutools '09, Brussels, BEL, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[42] K. Poularakis, G. Iosifidis, and L. Tassiulas. Sdn-enabled tactical ad hoc networks: Extending programmable control to the edge. *IEEE Communications Magazine*, 56(7):132–138, July 2018.

[43] Sulma Rashid, Qaisar Ayub, M Soperi Mohd Zahid, and Abdul Hanan Abdullah. Message drop control buffer management policy for dtn routing protocols. *Wireless personal communications*, 72(1):653–669, 2013.

[44] E. M. Royer and Chai-Keong Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, 6(2):46–55, April 1999.

[45] Amazon Web Services. Snowball. `https://aws.amazon.com/snowball/`. Accessed: 2020-01-18.

[46] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, MobiCom '06, page 334–345, New York, NY, USA, 2006. Association for Computing Machinery.

[47] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: modeling a three-tier architecture for sparse sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, pages 30–41, May 2003.

[48] Rostam Shirani and Faramarz Hendessi. A markov chain model for evaluating performance of store-carry-forward procedure in vanets. In *2008 11th IEEE Singapore International Conference on Communication Systems*, pages 593–598. IEEE, 2008.

[49] Mihail L. Sichitiu. Mobility models for ad hoc networks. In *Guide to Wireless Ad Hoc Networks*, pages 237–254. Springer London, London, 2009.

[50] Tara Small and Zygmunt J. Haas. The shared wireless infostation model: A new ad hoc networking paradigm (or where there is a whale, there is a way). In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, MobiHoc '03, page 233–244, New York, NY, USA, 2003. Association for Computing Machinery.

[51] Tara Small and Zygmunt J. Haas. Resource and performance tradeoffs in delay-tolerant wireless networks. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking*, WDTN '05, page 260–267, New York, NY, USA, 2005. Association for Computing Machinery.

[52] Itu Snigdh and K Sridhar Patnaik. Energy management in oppnets. In Khaleel Ahmad, Nur Izura Udzir, and Ganesh Chandra Deka, editors, *Opportunistic Networks: Mobility Models, Protocols, Security, and Privacy*, chapter 8. CRC Press, 2018.

[53] C.C. Sobin. An efficient buffer management policy for dtn. *Procedia Computer Science*, 93:309 – 314, 2016. Proceedings of the 6th International Conference on Advances in Computing and Communications.

[54] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*, pages 79–85, March 2007.

[55] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S. Raghavendra. Spray and wait: An efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking*, WDTN '05, page 252–259, New York, NY, USA, 2005. Association for Computing Machinery.

[56] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and C.S. Raghavendra. Single-copy routing in intermittently connected mobile networks. In *Proceedings of IEEE Conference of Sensor and Ad Hoc Communications and Networks*, pages 235 – 244, 11 2004.

[57] Zhenxi Sun, Yuebin Bai, Rui Wang, and Weitao Wang. Comsp: correlated contact and message scheduling policy in dtn. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 595–602. IEEE, 2013.

[58] Chai K Toh. *Wireless ATM and Ad-Hoc Networks: Protocols and Architectures.* Kluwer Academic Publishers, USA, 1996.

[59] Chai K Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems.* Prentice Hall, 1 edition, 2001.

[60] Amin Vahdat and David Becker. Epidemic routing for partially-connected ad hoc networks. Technical report, Internet Systems and Storage Group, 2000.

[61] Charles F Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

[62] Guido van Rossum and Fred L. Drake. *The Python Language Reference Manual.* Network Theory Ltd., 2011.

[63] Athanasios Vasilakos, Yan Zhang, and Thrasyvoulos Spyropoulos, editors. *Delay tolerant networks: protocols and applications.* Number 19 in Wireless networks and mobile communications. CRC Press, Boca Raton, 2012.

[64] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[65] John Von Neumann et al. The general and logical theory of automata. *1951*, pages 1–41, 1951.

[66] En Wang, Yongjian Yang, Bing Jia, and Tingting Guo. The dtn routing algorithm based on markov meeting time span prediction model. *International Journal of Distributed Sensor Networks*, 9(9):736796, 2013.

[67] H. Wang, B. Crilly, W. Zhao, C. Autry, and S. Swank. Implementing mobile ad hoc networking (manet) over legacy tactical radio links. In *MILCOM 2007 - IEEE Military Communications Conference*, pages 1–7, Oct 2007.

[68] Wirepas. Applications in smart lighting. `https://wirepas.com/applications/smart-lighting/`. Accessed: 2020-01-18.

[69] J. Wu and F. Dai. Mobility-sensitive topology control in mobile ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(6):522–535, June 2006.

[70] Jun Wu, Shuhui Yang, and Fei Dai. Logarithmic store-carry-forward routing in mobile ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):735–748, 06 2007.

[71] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '04, page 187–198, New York, NY, USA, 2004. Association for Computing Machinery.

[72] Wenrui Zhao, Yang Chen, Mostafa H. Ammar, Mark D. Corner, Brian Neil Levine, and Ellen W. Zegura. Capacity enhancement using throw-boxes in mobile delay tolerant networks. Technical report, Georgia Institute of Technology, 2006.

[73] Reinholds Zviedris, Atis Elsts, Girts Strazdins, Artis Mednis, and Leo Selavo. Lynxnet: Wild animal monitoring using sensor networks. In Pedro J. Marron, Thiemo Voigt, Peter Corke, and Luca Mottola, editors, *Real-World Wireless Sensor Networks*, pages 170–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.